Chapter 5

Hodgkin and Huxley Model

5.1 Vocabulary

- Depolarization
- Positive Feedback
- Hyperpolarization
- Negative Feedback
- Membrane Potential
- Sodium-Potassium Pump
- Nernst Potential
- Reversal Potential
- Equilibrium Potential
- Driving Force
- Conductance
- Leak Current
- Leaky Integrate and Fire Model
- Absolute Refractory Period
- Gating variable

5.2 Introduction

Before you read this chapter, we would like to draw your attention to this video. We call this a Zombie Squid because the squid is in fact dead; however, it is recently deceased. Since the squid passed shortly before, Adenosine triphosphate (ATP) energy stores are still available to the squid's muscles. When the soy sauce, which has a lot of sodium chloride (salt) in it, is poured onto the squid, the salt in the soy sauce causes a voltage change which causes the squid's muscles to contract. Thus, we have a Zombie Squid.

So why is the Zombie Squid important? The Hodgkin-Huxley Model, said to

have started the field of computational neuroscience, all hinges on the giant axons of squid. In the 1950s Alan Hodgkin and Andrew Huxley built a model that shows us how computers can successfully predict certain aspects of the brain that cannot be directly studied. The two even won a Nobel Prize in Physiology or Medicine in 1963 with Sir John Carew Eccles for their model. The Hodgkin-Huxley Model is now the basis of all conductance-based models. As a result, we can now understand how an action potential works, and why it is an all-or-none event.

While Hodgkin and Huxley created their model in the 1950s, the first recording of an action potential was done by Edgar Adrian in the 1920s. However, the first person to realize that neurons communicate via electrical signals came much earlier in 1791, when Luigi Galvani found that electricity from lightning or primitive batteries can cause a dead frog's leg muscle to contract. This led to a good amount of Frankenstein-like science with interested parties running electricity through dead bodies in an attempt to bring them back to life. However, the next truly scientific discovery came from Hermann Helmholtz in the 19th century. Helmholtz found that he could measure the speed of muscles contracting when he stimulated the nerve linked to that specific muscle. The Hodgkin-Huxley Model was then created once Adrian noted that not only were action potentials discrete, but the firing rate (spike per second) increased as stimulation to the nerve increased.

5.3 The Hodgkin and Huxley model

Alan Hodgkin (pictured left) and Andrew Huxley (pictured right) were two Cambridge University undergraduates who eventually found themselves working in a marine biology laboratory with the axon of a giant squid. The two men were able to derive the necessary information for their influential model of an action potential using the massive axon of the giant squid.

Hodgkin and Huxley developed a series of equations that could accurately predict and depict action potentials. Their work is a cornerstone for computational modeling as computer modelling can now be used to mimic the biological properties of a neuron that we are unable to directly observe.

Really the Hodgkin-Huxley Model is just an elaboration on the Integrate and Fire Model. The Integrate and Fire model was generated by French neuroscientist Louis Lapicque, who in 1907 sought to generate a mathematical model that could be used to predict and graph an action potential. In his efforts to understand action potentials, Lapicque chose to model the flow of ions as a single **leak current**.

Hodgkin and Huxley took the single conductance term from the Integrate and Fire Model is broken up into three separate conductance terms, each relating to a different ion channel. These conductance terms are known as **gating variables** and are labeled m, n, and h. Voltage-gated sodium channel activation is modeled

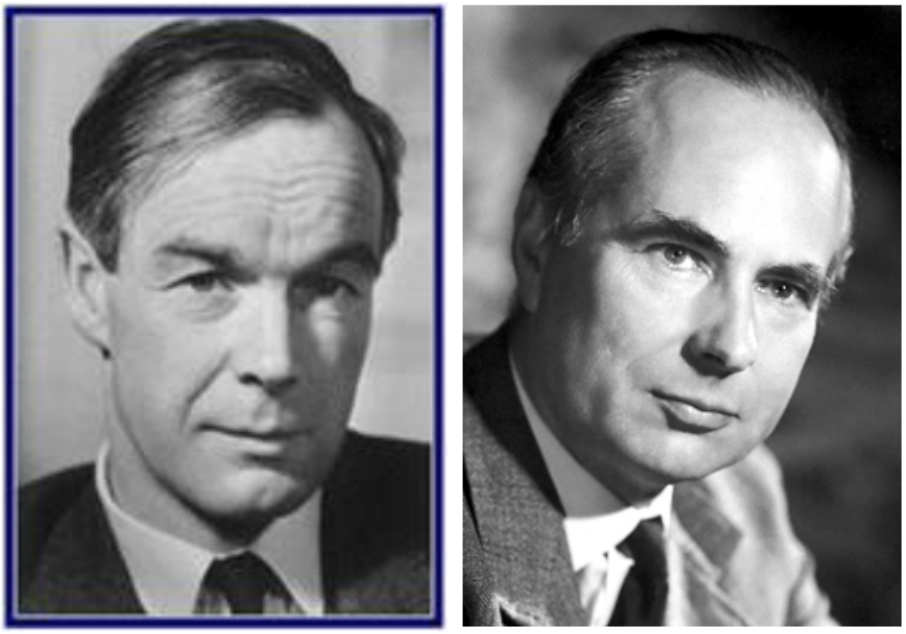

Figure 5.1: Alan Hodgkin (left) and Andrew Huxley (right).

by the letter *ms*. Voltage-gated sodium channels have three subunits, as these three subunits are involved in the channels activation, m is raised to the third power. Voltage-gated sodium channel also inactivate at the peak of the action potential and this variable is modeled by the letter h. The combination of m and h gives rise to the conductance of Voltage-gated sodium channel which is modeled below:

$$\bar{g}_{Na} m^3 h(V(t) - E_{Na})$$

Voltage-gated potassium channels are modeled by the letter n. Voltage-gated potassium channels have four subunits, and thus the gating variable, n, is raised to the fourth power. The conductance of Voltage-gated potassium channels is modeled below:

$$\bar{g}_K n^4 (V(t) - E_K$$

The final conductance taken into account by Hodgkin and Huxley is the leak potential of all the ions. The Leak conductance is taken into account for the instance when all ion channels are open. This conductance is represented below:

$$\bar{g}_L (V(t) - E_L)$$

These three conductance variables are combined together to form the Hodgkin-Huxley equation which is written as follows:

$$C\frac{dV}{dt} = I_e(t) - [(\bar{g}_{Na} m^3 h(V(t) - E_{Na})) + (\bar{g}_K n^4 (V(t) - E_K)) + (\bar{g}_L (V(t) - E_L))]$$

Expression	Meaning
n	Potassium gating variable
m	Sodium activation gating variable
h	Sodium inactivation gating variable
C_m	Specific membrane capacitance
I_e	Injected current
$\bar{g}_{Na}$	Maximum Na+ conductance
$\bar{g}_K$	Maximum K+ conductance
$\bar{g}_L$	Maximum leak conductance
V_m	Membrane potential
E_{Na}	Sodium Nernst potential
E_K	Potassium Nernst potential
E_L	Leak Nernst potential

Additionally, we can calculate the value of each gating variable over different voltages and times:

$$m\frac{dV}{dt} = \alpha_m(V)(1-m) - \beta_m(V)m$$

Both n and h can be substituted for m in the above equation in order to calculate values for each gating variable. Additionally, note the α and β in the equation are rate constants that govern the opening and closing (respectively), of their channels. Here are their values:

$$\alpha_n(V_m) = \frac{0.01(V_m+55)}{1-exp(-0.1(V_m+55))}$$

$$\alpha_m(V_m) = \frac{0.1(V_m+40)}{1-exp(-0.1(V_m+40))}$$

$$\alpha_h(V_m) = 0.07exp(-0.05(V_m+65))$$

$$\beta_n(V_m) = 0.125exp(-0.0125(V_m+65))$$

$$\beta_m(V_m) = 4exp(-0.0556(V_m+65))$$

$$\beta_h(V_m) = \frac{1}{1+exp(-0.1(V_m+35))}$$

Expression	Meaning
α_n	Rate constant for K+ channel opening
α_m	Rate constant for Na+ activation gate opening
α_h	Rate constant for Na+ inactivation gate opening
β_n	Rate constant for K+ channel closing
β_m	Rate constant for Na+ activation gate closing
β_h	Rate constant for Na+ inactivation gate closing

Worked Example:
Have you ever wondered how anesthesia makes a tooth extraction painless? It's because anesthesia works by blocking the activation of voltage-dependent Na+ channels. This prevents the propagation of the action potentials that carry that awful pain sensation.

Using the equations below, calculate the maximum conductances of each ion in the resting state.

Useful parameters:

- V_m = -68 mV
- C_m = -20.1 nF
- E_L = -54 mV
- E_{Na} = 50 mV
- E_K = -77 mV
- $R_L = 1/3M\Omega$
- $g_{Na} = 1200mS/mm^2$

Equations:

$$I_{ion} = g_{Na} \cdot m^3h(V_m - E_{Na}) + g_K \cdot n^4(V_m - E_K) + g_L \cdot (V_m - E_L)$$

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

$$\frac{dm}{dt} = \alpha_m(V) \cdot (1 - m) - \beta_m(V) \cdot (m)$$

$$\frac{dh}{dt} = \alpha_h(V) \cdot (1 - h) - \beta_h(V) \cdot (h)$$

Step 1: Understand the question The resting potential can be considered to be a steady state because the voltage is not changing. Therefore, $\frac{dp}{dt} = 0$ for p={n, m, h} and therefore the last three equations will not be used.

Step 2: Calculate n, m, and h We need to now use the resting potential to solve for the steady state values of the gating variables.

$$p_\infty = \frac{\alpha_p}{\alpha_p + \beta_p}$$

for p = {n, m, h}

Therefore, the very first step is to calculate each α and β.

$$\alpha_n(V_m) = \frac{0.01(V_m + 55)}{1 - exp(-0.1(V_m + 55))}$$

$$\alpha_n(V_{rest}) = \frac{0.01(-68 + 55)}{1 - exp(-0.1(-68 + 55))}$$

$$\alpha_n = 0.049$$

$$\alpha_m(V_m) = \frac{0.1(V_m + 40)}{1 - exp(-0.1(V_m + 40))}$$
$$\alpha_m(V_{rest}) = \frac{0.1(-68 + 40)}{1 - exp(-0.1(-68 + 40))}$$
$$\alpha_m(V_{rest}) = 0.18$$

$$\alpha_h(V_m) = 0.07exp(-0.05(V_m + 65))$$
$$\alpha_h(V_{rest}) = 0.07 \cdot exp(-0.05 \cdot (-68 + 65))$$
$$\alpha_h(V_{rest}) = 0.08$$

$$\beta_n(V_m) = 0.125exp(-0.0125(V_m + 65))$$
$$\beta_n(V_{rest}) = 0.125 \cdot exp(-0.0125 \cdot (-68 + 65))$$
$$\beta_n(V_{rest}) = 0.13$$

$$\beta_m(V_m) = 4exp(-0.0556(V_m + 65))$$
$$\beta_m(V_{rest}) = 4 \cdot exp(-0.0556 \cdot (-68 + 65))$$
$$\beta_m(V_{rest}) = 4.73$$

$$\beta_h(V_m) = \frac{1}{1 + exp(-0.1(V_m + 35))}$$
$$\beta_h(V_{rest}) = \frac{1}{1 + exp(-0.1(-68 + 35))}$$
$$\beta_h(V_{rest}) = 0.036$$

Now, with each of our α and β values, we can calculate our gating variables:

$$n_\infty = \frac{\alpha_n}{\alpha_n + \beta_n}$$
$$n_\infty = \frac{0.049}{0.049 + 0.13} = 0.274$$

$$m_\infty = \frac{\alpha_m}{\alpha_m + \beta_m}$$
$$m_\infty = \frac{0.18}{0.18 + 4.73} = 0.037$$

$$h_\infty = \frac{\alpha_h}{\alpha_h + \beta_h}$$
$$h_\infty = \frac{0.08}{0.08 + 0.036} = 0.690$$

Step 3: Calculate g_L from resistance units. Remember that $g = \frac{1}{R}$!

$$g_L = (1/R) = (1/(1/3)) = 3mS/mm^2$$

Step 4: Solve for g_K Remember that from Kirchhoff's Law that the algebraic sum of all the currents entering and leaving a junction must be equal to 0. Therefore:

$$0 = g_{Na} \cdot m^3 h(V_m - E_{Na}) + g_K \cdot n^4(V_m - E_K) + g_L \cdot (V_m - E_L)$$

And we may plug in the values that we already have:

$$0 = 1200 \cdot 0.037^3 0.69 \cdot (-68 - 50) + g_K \cdot 0.274^4 \cdot (-68 - -77) + 3 \cdot (-68 - -54.387)$$
$$0 = 4.956 + g_K \cdot 0.051 + (-40.84)$$
$$g_K = 703.6$$

Worked Example: The voltage of a neuron is clamped at -20 mV, depolarized from its resting potential of -65 mV. The steady-state values of the gating variables in the two conditions are shown below. Comment on what these changes mean for the neuron's behavior.

V=-65 mV	V=-20mV
m=0.0529	m=0.875
n=0.3177	n=0.820
h=0.5961	h=0.009

Answer The value of m represents the probability of voltage-gated Na+ channels to be open. This probability increases as the cell depolarizes. The n value represents the probability that the voltage-gated K+ channel is open. Like the Na+ channels, this probability increases with depolarization, but not to the same extent. The h values represent the probability of Na+ channel inactivation. This decreases significantly with depolarization because we have not hit the peak of the action potential. It is fair to assume that the h value will increase as we near the peak.

5.4 Summary

An action potential is the electro-chemical signal that propagates down a neuron. Action potentials are facilitated by the electrochemical gradient that is maintained through the action of the Sodium-Potassium Pump. The role that each ion plays within the action potential can be determined through the use of the Nernst Equation, which allows us to understand the movement of a specific ion at a specific membrane voltage. Understanding the role of ions, is important, but this is not how we actually graph an action potential. One of the first modules developed to graph an action potential was the integrate and fire module. This equation disregarded all of the biomechanical features of an action potential and focused on the subthreshold membrane dynamics of a neuron. This model was relatively effective, until Hodgkin and Huxley reassessed it and changed the single leak resistance in integrate and fire to three separate resistance terms. Hodgkin and Huxley in doing this created an equation that more thoroughly analyzed and depicted the very action potential that is displayed to this day.

5.5 Exercises:

5.5.1 Conceptual Exercises:

1. Briefly explain how the Zombie Squid moves. Why is this important to understand?

2. Action potentials are characterized by both negative and positive feedback. Provide one example of each that takes place during an action potential.

3. Describe one difference between the leaky integrate and fire model and the Hodgkin and Huxley model.

4. Why does depolarization not continue indefinitely once voltage-gated Na+ channels have opened?

5. Describe the specific stages of the action potential. How are these stages relate to positive/negative feedback? What kinds of channels are open/closed during the respective stages?

5.5.2 Coding Exercises:

Welcome to the coding problems for the Hodgkin and Huxley model! These exercises will help you get comfortable with using the Hodgkin and Huxley Model in different settings. To get started, simply run the following cell to create a few custom functions that we will be using for these exercises. No changes are needed, but feel free to look through the code and descriptive comments to see how each aspect of the model is implemented!

```
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt

#Creates a custom Hodgkin and Huxley function
def HHmodel(Ie, dt, tFinal, tStimStart, tStimEnd, RestingPotential):
    import matplotlib.pyplot as plt
    # Defines the model parameters
    c      =  10 # capacitance per unit area (nF/mm^2)
    gMaxL =  3 # leak maximal conductance per unit area (mS/mm^2)
    EL     =  -54.387 # leak conductance reversal potential (mV)
    gMaxK =  360 # maximal K conductance per unit area (mS/mm^2)
    EK     =  -77 # K conductance reversal potential (mV)
    gMaxNa=  1200 # maximal Na conductance per unit area (mS/mm^2)
    ENa    =  50 # Na conductance reversal potential (mV)

    # sets up data structures to hold relevant variable vectors
    timeVec = np.arange(0,tFinal, dt)
    voltageVector = np.zeros(len(timeVec))
    Ivector = np.zeros(len(timeVec))
    mVec = np.zeros(len(timeVec))
    hVec = np.zeros(len(timeVec))
    nVec = np.zeros(len(timeVec))
    tauVec = np.zeros(len(timeVec))

    # assigns the initial value of each variable
    voltageVector[0] =  RestingPotential
    mVec[0] = 0.0529
    hVec[0] = 0.5961
    nVec[0] = 0.3177

    # defines the stimulus
    tStimStart = int(tStimStart/dt)
    tStimEnd = int(tStimEnd/dt)
    Ivector[tStimStart:tStimEnd] = Ie

    # For-loop integrates equations into model
    for v in range(len(timeVec)-1):
        # Calculates alpha values for m, h, and n
        alpha_n = 0.01*(voltageVector[v] + 55) /
          (1-np.exp(-0.1*(voltageVector[v]+55)))
        alpha_m = (0.1*(voltageVector[v]+40))/
          (1-np.exp(-0.1*(voltageVector[v]+40)))
        alpha_h = 0.07*np.exp(-.05*(voltageVector[v]+65))
```

```
        # Calculates beta values for m, h, and n
        beta_n = 0.125*np.exp(-.01125*(voltageVector[v]+55))
        beta_m = 4*np.exp(-.05556*(voltageVector[v]+65))
        beta_h = 1 / (1 + np.exp(-0.1*(voltageVector[v]+35)))

        # Calculates tau values for m, h, and n
        tau_n = 1 / (alpha_n + beta_n)
        tau_m = 1 / (alpha_m + beta_m)
        tau_h = 1 / (alpha_h + beta_h)

        # Calculates inf values for m, h, and n
        pm = alpha_m/(alpha_m + beta_m)
        pn = alpha_n/(alpha_n+ beta_n)
        ph = alpha_h/(alpha_h + beta_h)

        # Calculates and store values in m, h, and n vectors
        mVec[v+1] = pm + (mVec[v] - pm)*np.exp(-dt/tau_m)
        nVec[v+1] = pn + (nVec[v] - pn)*np.exp(-dt/tau_n)
        hVec[v+1] = ph + (hVec[v] - ph)*np.exp(-dt/tau_h)

        # Updates Vinf and tauV
        denominator = gMaxL + gMaxK*(nVec[v]**4) +
          gMaxNa*(mVec[v]**3)*hVec[v]
        tauV = c / denominator
        Vinf = ((gMaxL)*EL + gMaxK*(nVec[v]**4)*EK +
          gMaxNa*(mVec[v]**3)*hVec[v]*ENa + Ivector[v])/denominator

        # Calculates and store next voltage value in vector
        voltageVector[v+1] = Vinf + (voltageVector[v]-Vinf)*np.exp(-dt/tauV)

    # Plotting
    plt.figure(1, figsize=(10,10))
    plt.subplot(4,1,1)
    plt.plot(timeVec,voltageVector)
    plt.title('Hodgkin and Huxley Simulation')
    plt.ylabel('Voltage in mV')
    plt.subplot(4,1,2)
    plt.plot(timeVec, mVec)
    plt.ylabel('g_Na activation variable m')
    plt.subplot(4,1,3)
    plt.plot(timeVec, hVec)
    plt.ylabel('g_Na inactivation variable h')
    plt.subplot(4,1,4)
    plt.plot(timeVec, nVec)
    plt.ylabel('g_K activation variable')
```

```
    plt.xlabel('Time in ms')


#creates a custom function that will determine the minimum Ie
# required to surpass the threshold potential

def HHmodel_threshold(Ie):
    tStimStart = 250
    tStimEnd   = 750
    dt          = 0.1 # time step (ms)
    tFinal      = 1000 # total time of run (ms)
    RestingPotential = -65
    c     = 10 # capacitance per unit area (nF/mm^2)
    gMaxL = 3 # leak maximal conductance per unit area (mS/mm^2)
    EL    =  -54.387 # leak conductance reversal potential (mV)
    gMaxK =  360 # maximal K conductance per unit area (mS/mm^2)
    EK    =  -77 # K conductance reversal potential (mV)
    gMaxNa=  1200 # maximal Na conductance per unit area (mS/mm^2)
    ENa   =  50 # Na conductance reversal potential (mV)


    for current in range(len(Ie)):

        timeVec = np.arange(0,tFinal, dt)
        voltageVector = np.zeros(len(timeVec))
        Ivector = np.zeros(len(timeVec))
        mVec = np.zeros(len(timeVec))
        hVec = np.zeros(len(timeVec))
        nVec = np.zeros(len(timeVec))
        tauVec = np.zeros(len(timeVec))
        voltageVector[0] =  RestingPotential
        mVec[0] = 0.0529
        hVec[0] = 0.5961
        nVec[0] = 0.3177

        Ivector[2499:7499] = Ie[current]

        for v in range(len(timeVec)-1):
            alpha_n = 0.01*(voltageVector[v] + 55) /
              (1-np.exp(-0.1*(voltageVector[v]+55)))
            alpha_m = (0.1*(voltageVector[v]+40))/
              (1-np.exp(-0.1*(voltageVector[v]+40)))
            alpha_h = 0.07*np.exp(-.05*(voltageVector[v]+65))
            beta_n = 0.125*np.exp(-.01125*(voltageVector[v]+55))
            beta_m = 4*np.exp(-.05556*(voltageVector[v]+65))
```

```
            beta_h = 1 / (1 + np.exp(-0.1*(voltageVector[v]+35)))
            tau_n = 1 / (alpha_n + beta_n)
            tau_m = 1 / (alpha_m + beta_m)
            tau_h = 1 / (alpha_h + beta_h)
            pm = alpha_m/(alpha_m + beta_m)
            pn = alpha_n/(alpha_n+ beta_n)
            ph = alpha_h/(alpha_h + beta_h)
            mVec[v+1] = pm + (mVec[v] - pm)*np.exp(-dt/tau_m)
            nVec[v+1] = pn + (nVec[v] - pn)*np.exp(-dt/tau_n)
            hVec[v+1] = ph + (hVec[v] - ph)*np.exp(-dt/tau_h)
            denominator = gMaxL + gMaxK*(nVec[v]**4) +
              gMaxNa*(mVec[v]**3)*hVec[v]
            tauV = c / denominator
            Vinf = ((gMaxL)*EL + gMaxK*(nVec[v]**4)*EK +
              gMaxNa*(mVec[v]**3)*hVec[v]*ENa + Ivector[v])/ denominator
            voltageVector[v+1] = Vinf +
              (voltageVector[v]-Vinf)*np.exp(-dt/tauV)

            # Checks to see if the given current resulted in an
            # action potential
            # Values around 25 mv is only reached if the neuron spikes.
            # A value of 20 would similarly be appropriate
            if voltageVector[v] > 25:
                return(print("With an external current of", Ie[current]-1 ,
                  "nA/mm^2 threshold potential was finally reached!"))

    return(print("Looks like you didn't provide a large value in your range
      caused the neuron to spike. Try again!"))


# plots the behavior of a neuron when injected with different
# currents on the same graphs
def HHmodel_compare(Ie, RestingPotential):
    terminate = 0
    tStimStart  =  10
    tStimEnd    =  30
    tFinal      =  50
    dt          =  0.002
    c           =  10
    gMaxL       =  0.003e03
    EL          =  -54.387
    gMaxK       =  0.36e03
    EK          =  -77
    gMaxNa      =  1.2e03
    ENa         =  50
```

```
    for current in range(len(Ie)):
        timeVec = np.arange(0,tFinal, dt)
        voltageVector = np.zeros(len(timeVec))
        Ivector = np.zeros(len(timeVec))
        mVec = np.zeros(len(timeVec))
        hVec = np.zeros(len(timeVec))
        nVec = np.zeros(len(timeVec))
        tauVec = np.zeros(len(timeVec))

        voltageVector[0] = RestingPotential

        mVec[0] = 0.0529
        hVec[0] = 0.5961
        nVec[0] = 0.3177

        Ivector[5000:15000] = Ie[current]

        for v in range(len(timeVec)-1):
            alpha_n = 0.01*(voltageVector[v] + 55) /
              (1-np.exp(-0.1*(voltageVector[v]+55)))
            alpha_m = (0.1*(voltageVector[v]+40))/
              (1-np.exp(-0.1*(voltageVector[v]+40)))
            alpha_h = 0.07*np.exp(-.05*(voltageVector[v]+65))
            beta_n = 0.125*np.exp(-.01125*(voltageVector[v]+55))
            beta_m = 4*np.exp(-.05556*(voltageVector[v]+65))
            beta_h = 1 / (1 + np.exp(-0.1*(voltageVector[v]+35)))
            tau_n = 1 / (alpha_n + beta_n)
            tau_m = 1 / (alpha_m + beta_m)
            tau_h = 1 / (alpha_h + beta_h)
            pm = alpha_m/(alpha_m + beta_m)
            pn = alpha_n/(alpha_n+ beta_n)
            ph = alpha_h/(alpha_h + beta_h)
            mVec[v+1] = pm + (mVec[v] - pm)*np.exp(-dt/tau_m)
            nVec[v+1] = pn + (nVec[v] - pn)*np.exp(-dt/tau_n)
            hVec[v+1] = ph + (hVec[v] - ph)*np.exp(-dt/tau_h)
            denominator = gMaxL + gMaxK*(nVec[v]**4) +
              gMaxNa*(mVec[v]**3)*hVec[v]
            tauV = c / denominator
            Vinf = ((gMaxL)*EL + gMaxK*(nVec[v]**4)*EK +
              gMaxNa*(mVec[v]**3)*hVec[v]*ENa + Ivector[v])/
                denominator
            voltageVector[v+1] = Vinf +
              (voltageVector[v]-Vinf)*np.exp(-dt/tauV)

            # stores values so the two plots can be superimposed
```

```
            if terminate ==0:
                Mv = mVec
                Nv = nVec
                Hv = hVec
                Vv = voltageVector
                terminate = 1

    #plotting
    plt.figure(1, figsize=(10,10))
    plt.subplot(4,1,1)
    plt.plot(timeVec,voltageVector, timeVec, Vv)
    plt.title('Hodgkin and Huxley Simulation')
    plt.ylabel('Voltage in mV')
    plt.subplot(4,1,2)
    plt.plot(timeVec, mVec, timeVec, Mv)
    plt.ylabel('g_Na activation variable m')
    plt.subplot(4,1,3)
    plt.plot(timeVec, hVec, timeVec, Hv)
    plt.ylabel('g_Na inactivation variable h')
    plt.subplot(4,1,4)
    plt.plot(timeVec, nVec, timeVec, Nv)
    plt.ylabel('g_K activation variable')
    plt.xlabel('Time in ms')


print("Cell run successfully! Please proceed to the next part")
```

Problem 1: Exploring the Model *Part 1* What can the model do? The function HHmodel() provides insight by simulating a neuron firing and the behavior of its channel gating variables under various conditions. Simulate an experiment that runs for 500 ms in which a neuron is exposed to a current of 100 nA/mm^2 from 100-400 ms. Use time steps of 0.1 ms and set the resting potential to -65 mV.

Note, the code has been started for you - set the variables to their specified values and run the code.

```
tStimStart  =           # time to start injecting current (ms)
tStimEnd    =           # time to end injecting current (ms)
tFinal      =           # total time of run (ms)
dt          =           # time step (ms)
Ie          =           # nA/mm^2
RestingPotential =      #mv

# Run the custom function
HHmodel(Ie, dt, tFinal, tStimStart, tStimEnd, RestingPotential)
```

Part 2 Nice work! Does increasing the injected current increasing the firing rate? Using the same code, increase the injected current to 250 nA/mm^2 to find out!

Part 3, optional Play around with the parameters to see how changing the neuron's starting conditions will affect its behavior!

Problem 2: Threshold Potential *Part 1* Neurons operate on a "all or nothing" basis. The function HHmodel_threshold() takes different values of injected currents (I_e) and returns the minimum value that elicited an action potential. Using *np.arange()*, set I_e equal to a vector of increasing voltages to determine the minimum injected current required for the neuron to cross its threshold potential. Hint: first define your external stimulation, I_e then use the HHmodel_threshold() function.

Part 2 The function HHmodel_compare() will plot the behavior of a neuron and its gating variables for two different injected current values. The function's input is the array I_e that contains two values. Using the external current determined in part one and the preceding integer, run the code to see the difference between just missing and passing the threshold potential!

Part 3 Pretty striking difference right? Now imagine there is a neuron where the resting potential is -60 mV rather than -65. What would happen if all else remained equal? Change the resting potential to -60 mV and rerun the code to find out!(Note: HHmodel_compare() also accepts an input for the resting potential.)

Problem 3: Pick your Poison(s) *Part 1* The poisonous dart frog of Central and South America secretes the neurotoxin batrachotoxin (BTX). Exposure to BTX results causes paralysis by irreversibly binding to the sodium channels and preventing them from closing. In the following code, simulate a neuron's exposure to BTX by setting the relevant variable to the appropriate value. Note: the only code that should be altered has been sectioned off using hashtags.

```
# Define input parameters
c           =  10
gMaxL       =  0.003e03
EL          =  -54.387
gMaxK       =  0.36e03
EK          =  -77
gMaxNa      =  1.2e03
ENa         =  50
Ie = 200
tFinal = 1000

# Set up data structures
timeVec = np.arange(0,tFinal, dt)
voltageVector = np.zeros(len(timeVec))
Ivector = np.zeros(len(timeVec))
mVec = np.zeros(len(timeVec))
```

```
hVec = np.zeros(len(timeVec))
nVec = np.zeros(len(timeVec))
tauVec = np.zeros(len(timeVec))

# Initialize starting values
voltageVector[0] =  -65
mVec[0] = 0.0529
hVec[0] = 0.5961
nVec[0] = 0.3177
Ivector[1999:7999] = Ie

# Loop to calculate next values
for v in range(len(timeVec)-1):
    alpha_n = 0.01*(voltageVector[v] + 55) /
      (1-np.exp(-0.1*(voltageVector[v]+55)))
    alpha_m = (0.1*(voltageVector[v]+40))/
      (1-np.exp(-0.1*(voltageVector[v]+40)))
    alpha_h = 0.07*np.exp(-.05*(voltageVector[v]+65))
    beta_n = 0.125*np.exp(-.01125*(voltageVector[v]+55))
    beta_m = 4*np.exp(-.05556*(voltageVector[v]+65))
    beta_h = 1 / (1 + np.exp(-0.1*(voltageVector[v]+35)))


    ##############
    # if timeVec[v] >= 300:
        #  =    # Part one
    # if timeVec[v] > 600:
       #   =    # Part two
    ###############

    # Update time constants
    tau_n = 1 / (alpha_n + beta_n)
    tau_m = 1 / (alpha_m + beta_m)
    tau_h = 1 / (alpha_h + beta_h)

    # Update gating variables
    pm = alpha_m/(alpha_m + beta_m)
    pn = alpha_n/(alpha_n+ beta_n)
    ph = alpha_h/(alpha_h + beta_h)
    mVec[v+1] = pm + (mVec[v] - pm)*np.exp(-dt/tau_m)
    nVec[v+1] = pn + (nVec[v] - pn)*np.exp(-dt/tau_n)
    hVec[v+1] = ph + (hVec[v] - ph)*np.exp(-dt/tau_h)

    # Update voltage
    denominator = gMaxL + gMaxK*(nVec[v]**4) + gMaxNa*(mVec[v]**3)*hVec[v]
```

```
    tauV = c / denominator
    Vinf = ((gMaxL)*EL + gMaxK*(nVec[v]**4)*EK +
      gMaxNa*(mVec[v]**3)*hVec[v]*ENa + Ivector[v])/ denominator
    voltageVector[v+1] = Vinf +
      (voltageVector[v]-Vinf)*np.exp(-dt/tauV)

# Plot the results
plt.figure(1, figsize=(10,10))
plt.subplot(4,1,1)
plt.plot(timeVec,voltageVector)
plt.title('Hodgkin and Huxley Simulation')
plt.ylabel('Voltage in mV')
plt.subplot(4,1,2)
plt.plot(timeVec, mVec)
plt.ylabel('g_Na activation variable m')
plt.subplot(4,1,3)
plt.plot(timeVec, hVec)
plt.ylabel('g_Na inactivation variable h')
plt.subplot(4,1,4)
plt.plot(timeVec, nVec)
plt.ylabel('g_K activation variable')
```

Part 2 Unfortunately, there is no effective treatment to BTX poisoning. However, in theory, the membrane depolarization can be reversed using tetrodotoxin (TTX), which is produced by the pufferfish, blue ringed octopus, and other deadly creatures. This toxin also causes paralysis. However, unlike BTX, TTX prevents action potentials by binding to voltage-gated sodium channels and preventing the movement of sodium ions into the cell. Revise the above code to simulate exposure to TTX 300 ms after the BTX poisoning.

Part 3 Observe the behavior of the neuron between 600 and 800 ms. While this looks like more ideal behavior than when the neuron was exposed to only BTX, what should the behavior of the neuron be if neither poison had been administered? (Note, while this is a conceptual question, feel free to alter the code to see what should occur).