**Name:** Britt Ahn

**Course:** CASMA665 *Mathematical Neuroscience*

**Date:** October 1, 2024

**Assignment:** Homework 2, *Hodgkin-Huxley Neuron*

---

## Document Description

Address Challenges 1-4 in the following module: mark-kramer.github.io/BU-MA665-MA666/HH.html

---

## Challenge 1

**This section will address the following questions:**

1. Describe the dynamics during an action potential in the HH model.
2. How does the voltage change?
3. How do the gates open and close?
4. How do the ions flow?

---

### First, define the HH model.

Define functions for the main HH equations.

In [3]:
```python
# Import libraries

import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import find_peaks
```

In [4]:
```python
# Define alpha and beta functions for gating variables m, h, n.

def alphaM(V):
    return (2.5-0.1*(V+65)) / (np.exp(2.5-0.1*(V+65)) -1)

def betaM(V):
    return 4*np.exp(-(V+65)/18)

def alphaH(V):
    return 0.07*np.exp(-(V+65)/20)

def betaH(V):
    return 1/(np.exp(3.0-0.1*(V+65))+1)

def alphaN(V):
    return (0.1-0.01*(V+65)) / (np.exp(1-0.1*(V+65)) -1)

def betaN(V):
    return 0.125*np.exp(-(V+65)/80)
```

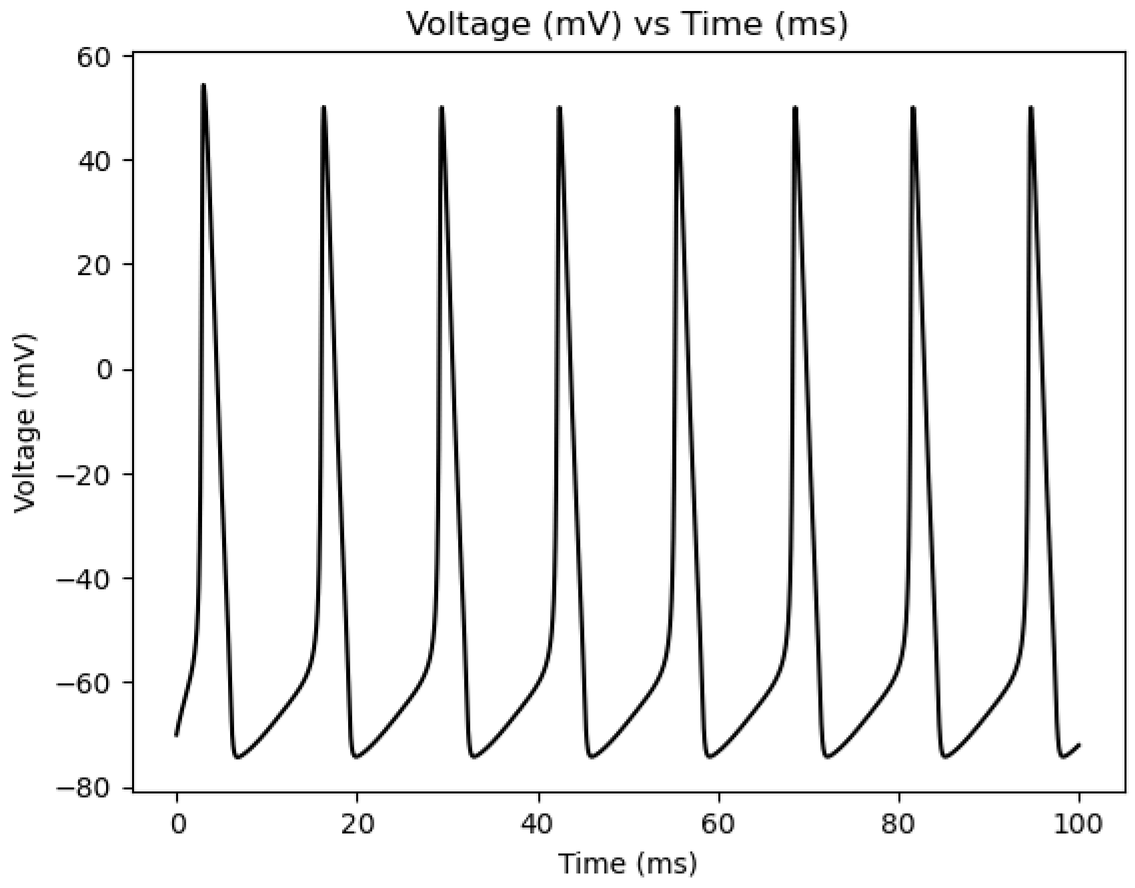
In [5]:
```python
"""
Define a function for the Hodgkin-Huxley equations.

Function name: HH

Args:
I0 -- input current
T0 -- duration of simulation, to be converted to milliseconds by time step dt
gK0_i -- max potassium conductance
gNa0_i -- max sodium conductance

Returns:
V -- voltage array
m -- sodium activation variable array
h -- sodium inactivation variable array
n -- potassium activation variable array
t -- time array (milliseconds)
"""

def HH(I0,T0, gK0_i, gNa0_i):
    # Define biological constants
    gNa0 = gNa0_i   # sodium conductance [mS/cm^2]
    ENa  = 125;     # equilibrium or Nernst potential for sodium [mV]
    gK0  = gK0_i;   # potassium conductance [mS/cm^2]
    EK   = -12;     # equilibrium or Nernst potential for potassium [mV]
    gL0  = 0.3;     # leak conductance [mS/cm^2]
    EL   = 10.6;    # equilibrium or Nernst potential for leak [mV]

    # Define a maximum duration of the simulation in milliseconds
    dt = 0.01;                # time step
    T  = int(np.ceil(T0/dt))

    # Initialize an array of times, voltages, activation/inactivation variables (m, h, n)
    t = np.arange(0,T)*dt  # [ms]
    V = np.zeros([T,1])
    m = np.zeros([T,1])
    h = np.zeros([T,1])
    n = np.zeros([T,1])

    # Define initial values for voltages, activation/inactivation variables (m, h, n)
    V[0]=-70.0  # mV
    m[0]=0.05   # m, h, n have no units because they are probabilities!! from [0,1]
    h[0]=0.54
    n[0]=0.34

    # Apply a numerical method (i.e. forward Euler) to compute the next value in the arrays for:
    for i in range(0,T-1):

        # Voltage
        V[i+1] = V[i] + dt*(gNa0*m[i]**3*h[i]*(ENa-(V[i]+65)) + gK0*n[i]**4*(EK-(V[i]+65)) + gL0*(EL-(V[i]+65)) + I0);

        # Sodium activation
        m[i+1] = m[i] + dt*(alphaM(V[i])*(1-m[i]) - betaM(V[i])*m[i]);

        # Sodium inactivation
        h[i+1] = h[i] + dt*(alphaH(V[i])*(1-h[i]) - betaH(V[i])*h[i]);

        # Potassium activation
        n[i+1] = n[i] + dt*(alphaN(V[i])*(1-n[i]) - betaN(V[i])*n[i]);

    return V,m,h,n,t
```

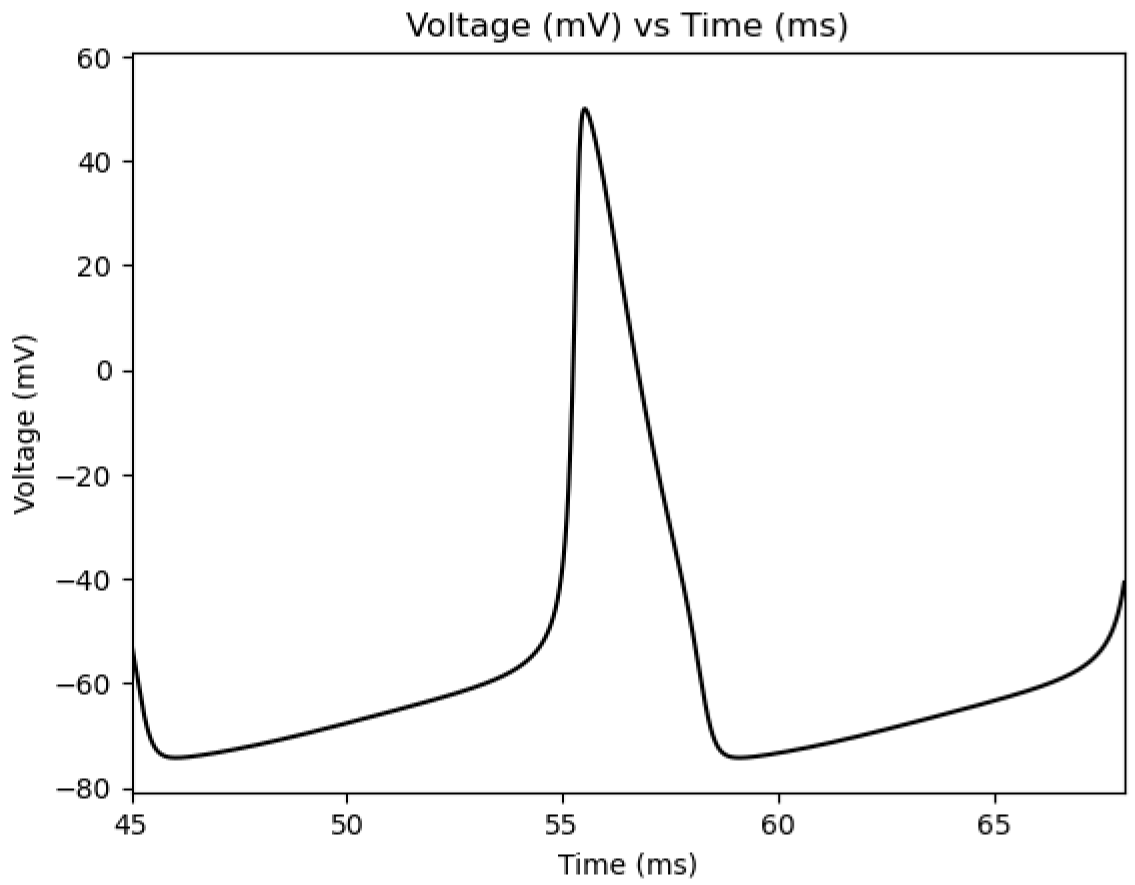
**Note:** *In the function above, it is possible to make the constants which are defined inside the function (i.e. gNa0, ENa, gK0, EK, gL0, EL, dt) arguments of the function, rather than defining them inside the function. However, later Challenges 3-4 interrogate what happens when potassium conductance is increased; or what happens when sodium conductance is increased. Hence I chose to make only gK0 and gNa0 into additional arguments for this HH function. The HH function takes in I0, T0, gK0, and gNa0 as arguments.*

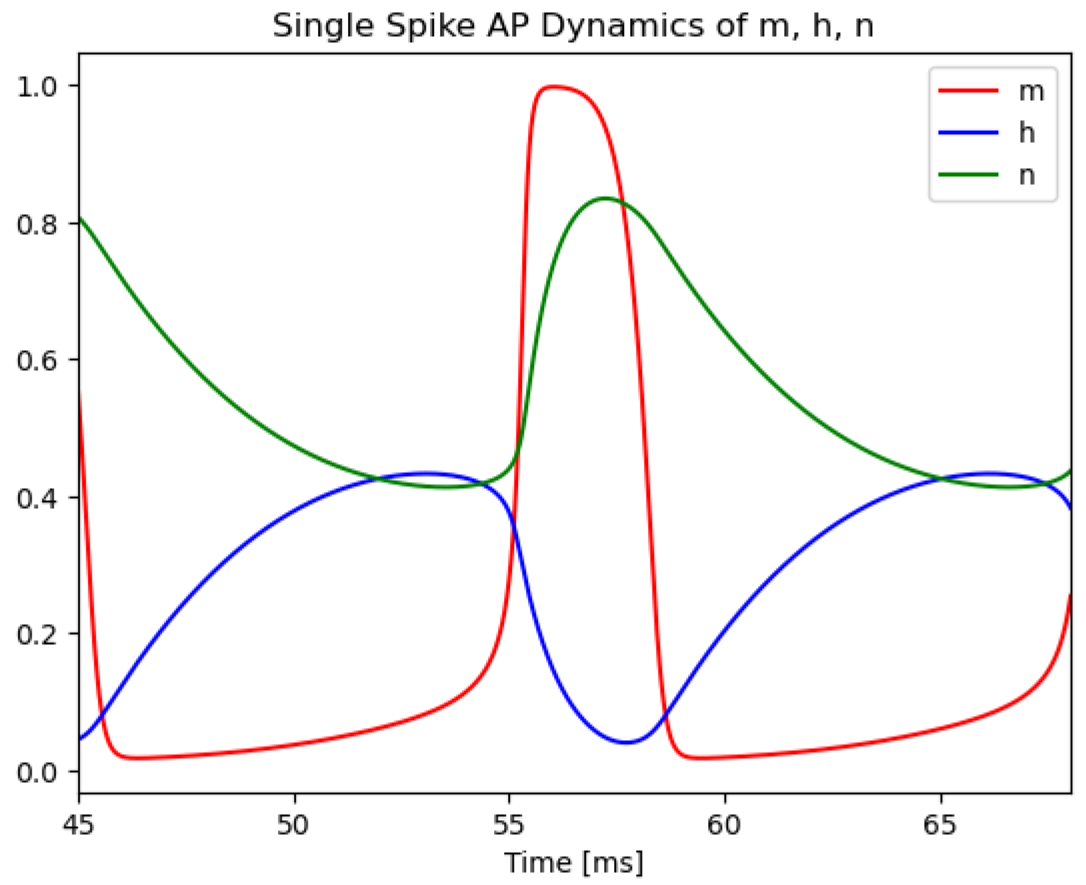
**Next, call the function HH defined above.**

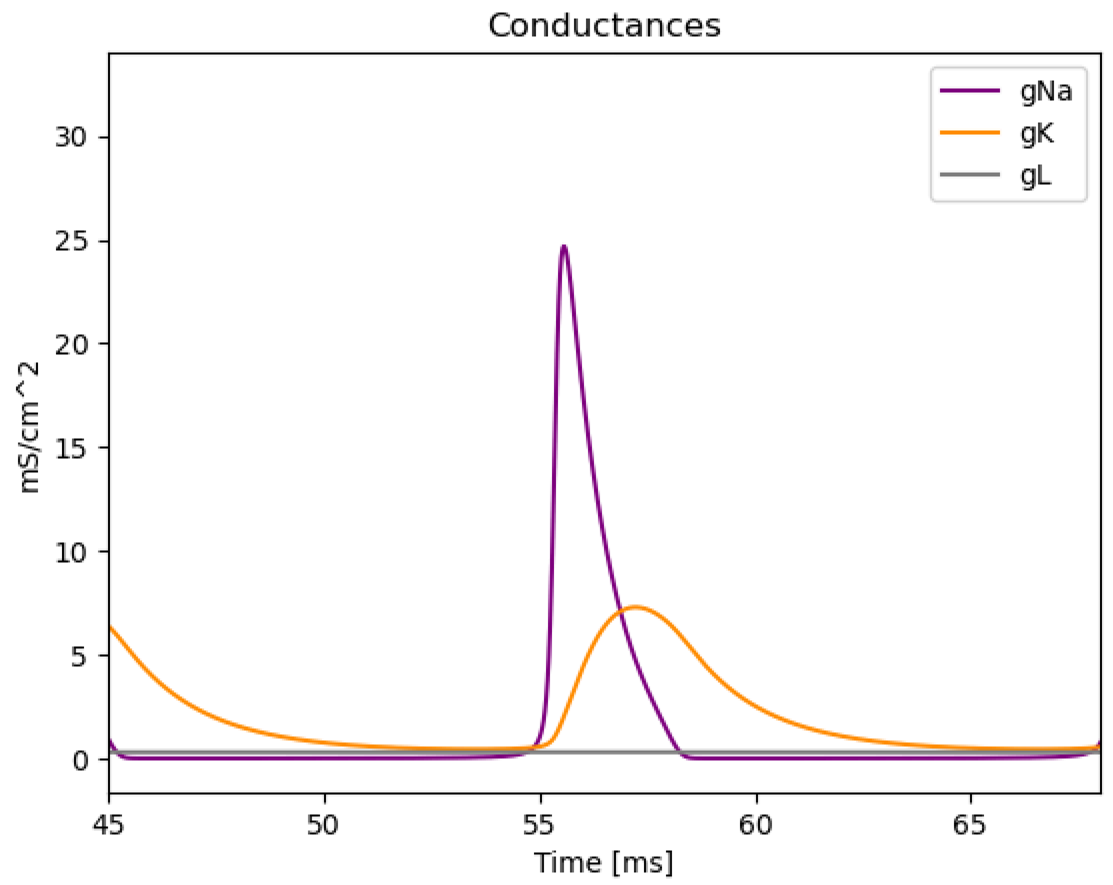
In [7]:
```
# Define the following "default" values
I0 = 5    # input current of 5
T0 = 100  # simulation duration of 100
gK0=15    # max potassium conductance of 15
gNa0=120  # max potassium conductance of 120

# Call the function HH
[V,m,h,n,t]=HH(I0,T0,gK0,gNa0)

# Plot voltages
plt.figure()
plt.title("Voltage (mV) vs Time (ms)")
plt.plot(t,V, 'black')
plt.xlabel("Time (ms)")
plt.ylabel("Voltage (mV)")
plt.show()

# Plot single spike
plt.figure()
plt.title("Voltage (mV) vs Time (ms)")
plt.plot(t,V, 'black')
plt.xlim([45, 68])
plt.xlabel("Time (ms)")
plt.ylabel("Voltage (mV)")
plt.show()

# Plot gating variables m, h, n
plt.figure()
plt.title("Single Spike AP Dynamics of m, h, n")
plt.plot(t,m,'red', label='m')
plt.plot(t,h,'blue', label='h')
plt.plot(t,n,'green', label='n')
plt.xlim([45, 68])
plt.xlabel('Time [ms]')
plt.legend()
plt.show()



####################
# Plot conductances

# Define conductances
gNa  = gNa0*(m**3)*h                # Sodium conductance array
```

```
gK   = gK0*(n**4)                     # Potassium conductance array
gL  = 0.3*(n/n)                     # Leak conductance array; multiply by 1 (n/n) to create array of equal shape as gK, gNa

# Plot 3 conductance traces in 1 graph
plt.figure()
plt.title("Conductances")
plt.plot(t,gNa,'purple', label='gNa')#... and plot the sodium conductance,
plt.plot(t,gK, 'darkorange', label='gK') #... and plot the potassium conductance,
plt.plot(t,gL, 'gray', label='gL') #... and plot the leak conductance.
plt.xlim([45, 68])
plt.xlabel('Time [ms]')             #... label the x-axis.
plt.ylabel('mS/cm^2')               #... and label the y-axis.
plt.legend()                        #... make a legend.
plt.show()
```




Voltage (mV) vs Time (ms)
Voltage (mV)
Time (ms)
60
40
20
0
−20
−40
−60
−80
45
50
55
60
65


Single Spike AP Dynamics of m, h, n
m
h
n
1.0
0.8
0.6
0.4
0.2
0.0
45
50
55
60
65
Time [ms]



---

## Revisit Challenge 1 Questions:

**1. Describe the dynamics during an action potential in the HH model.**

From the action potentials (APs) produced by the Hodgkin-Huxley (HH) model, we see persistent firing activity over a duration of 100 milliseconds (ms). Here we used an input current of 5, maximum potassium conductance of 15, and maximum sodium conductance of 120. During a single action potential, the cell depolarizes to approximately -50 mV, then hyperpolarizes to approximately -75 mV before returning to resting potential. The neuron spikes at a rate of ~80 spikes per second.

**2. How does the voltage change?**

Due to the input current, the voltage increases (becomes less negative) during the depolarization phase of the AP to approximately -50mV, then hyperoplarizes to approximately -75mV.

**3. How do the gates open and close?**

As the neuron depolarizes, we see a rise in m (sodium activation variable). This causes sodium gates to open, causing sodium ions to rush into the cell. Since sodium (Na+) is a positively charged ion, this causes the intracellular environment to become more positive and the neuron depolarizes (membrane potential becomes less negative). During this time, we also see a sharp decrease in the sodium inactivation variable, h, indicating that sodium channels are not inactivated during depolarization.

Shortly after the cell spikes, we see a sharp decrease in m and increase in h, in which sodium channels become temporarily inactivated. We also see a rise in the potassium gating variable. Notably, potassium channels do not inactivate (only sodium channels do) during the AP. This rise in variable n occurs shortly after the peak of the AP, following the peak of variable m. This indicates the role of potassium in reactivation, following the depolarization phase, in which the activation of potassium channels causes potassium to flood out of the cell. This causes the cell to become more negative, since potassium (K+) ions are positively charged. Since the cell loses positive ions, the intracellular environment becomes more negative, shown by a decrease in membrane potential (ie voltage; mV).

**4. How do the ions flow?**

The channel conductances affect the flow of ions during an AP produced by the HH model. With the depolarization of the cell, we see a sharp increase in sodium conductance gNa. The peak voltage / spike of the AP is followed shortly by a relatively smaller and slower increase in potassium conductance gK. During repolarization, the potassium conductance increases as the sodium conductance falls quickly. In this model, the leak conductance is a constant variable and remains the same value during the AP. gNa and gK change duirng the AP dependent on gating variables m, h, and n which are dependent on voltage and time.

During depolarization, sodium channels open causing positive sodium ions to flow into the cell. This results in the intracellular environment to become more positive. Shortly following the peak of depolarization, potassium channels open, causing potassium to rush out of the cell, resulting in an outward current of K+, causing the intracellular environment to become more negative. After depolarization, sodium channels are also temporarily inactivated. The gating (inactivation/activation) of sodium channels and the open/closing of potassium channels allows the neuron to spike and reset to resting potential.

---

## Bonus Challenge

### For ~ *fun* ~ , let's look at the bonus question before we do the rest of the challenges.

**In the bonus question, we will address:**

- Can we use forward and backward rate functions to plot the functions for m/h/n infinity and tau m/h/n?
- Can we use the alpha and beta functions defined above in Challenge 1 to do this?

*In other words, create **Figure 1** from https://mark-kramer.github.io/BU-MA665-MA666/Readings/Hodgkin-Huxley_Cheat_Sheet.pdf*

In [104…
```
# Bonus Problem

# Figure 1A: plot the steady states

m_inf=[]  # create an empty array for m steady state at varying voltages
```

```python
for v in V:  # use the voltages from the voltage array created in Challenge 1
    m_inf.append((alphaM(v)/(alphaM(v)+betaM(v))))  # append the value of m_infinity for each voltage in the voltage array from Challenge 1

h_inf=[]
for v in V:
    h_inf.append((alphaH(v)/(alphaH(v)+betaH(v))))

n_inf=[]
for v in V:
    n_inf.append((alphaN(v)/(alphaN(v)+betaN(v))))

# Plot Figure 1A
plt.figure()
print("Figure 1A of HH Cheat Sheet:")
plt.title("Steady State (m, h, n) vs Voltage")
plt.plot(V,m_inf, color="darkgreen", label='m_infinity')
plt.plot(V,h_inf, color="purple", label='h_infinity')
plt.plot(V,n_inf, color="skyblue", label='n_infinity')
plt.ylabel("Probabilities (no units)")
plt.xlabel("Voltage (mV)")
plt.legend()
plt.show()

# Figure 1B
tau_M=[]
for v in V:
    tau_M.append(1/(alphaM(v) + betaM(v)))

tau_H=[]
for v in V:
    tau_H.append(1/(alphaH(v) + betaH(v)))

tau_N=[]
for v in V:
    tau_N.append(1/(alphaN(v) + betaN(v)))

# Plot Figure 1B
plt.figure()
print("Figure 1B of HH Cheat Sheet:")
plt.title("Time Constants (m, h, n) vs Voltage")
plt.ylabel("Time (ms)")
plt.xlabel("Voltage (mV)")
plt.plot(V,tau_M, color="darkgreen", label='tau_m')
plt.plot(V,tau_H, color="purple", label='tau_h')
plt.plot(V,tau_N, color="skyblue", label='tau_n')
plt.legend()
plt.show()
```

Figure 1A of HH Cheat Sheet:

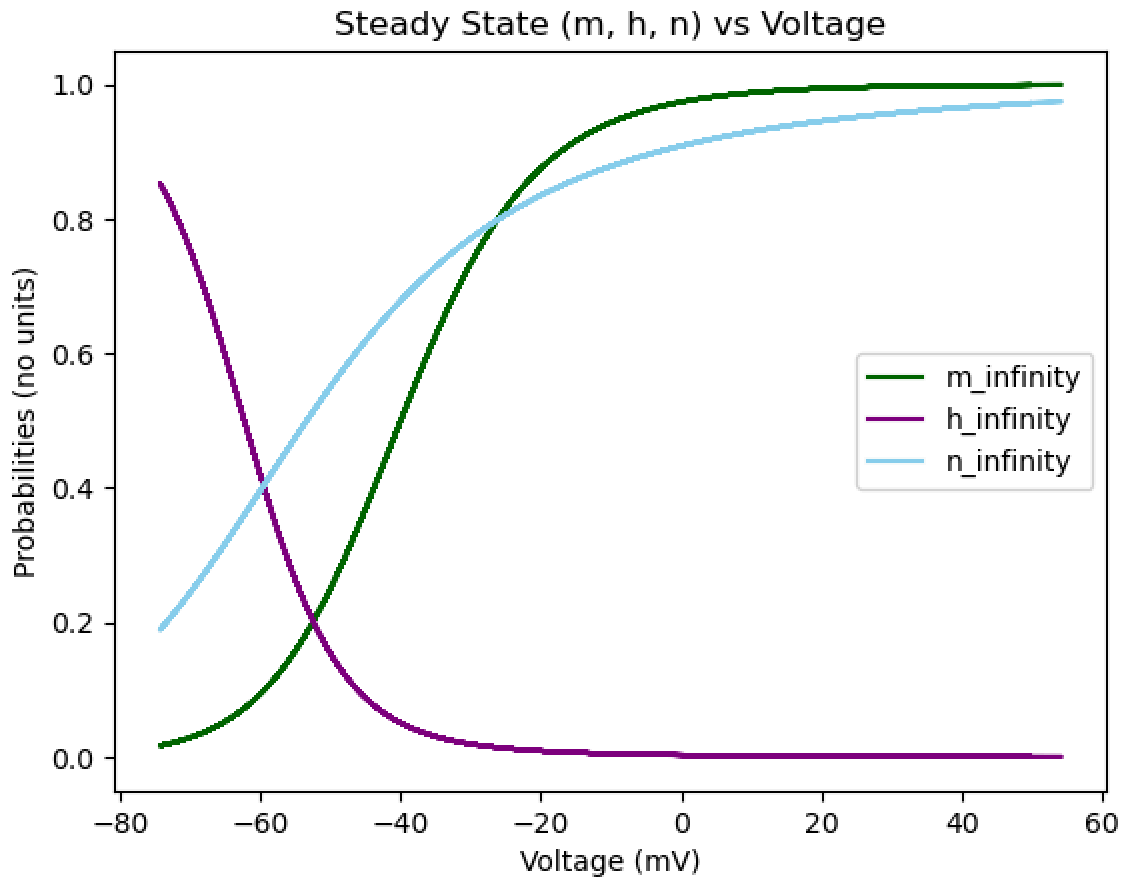
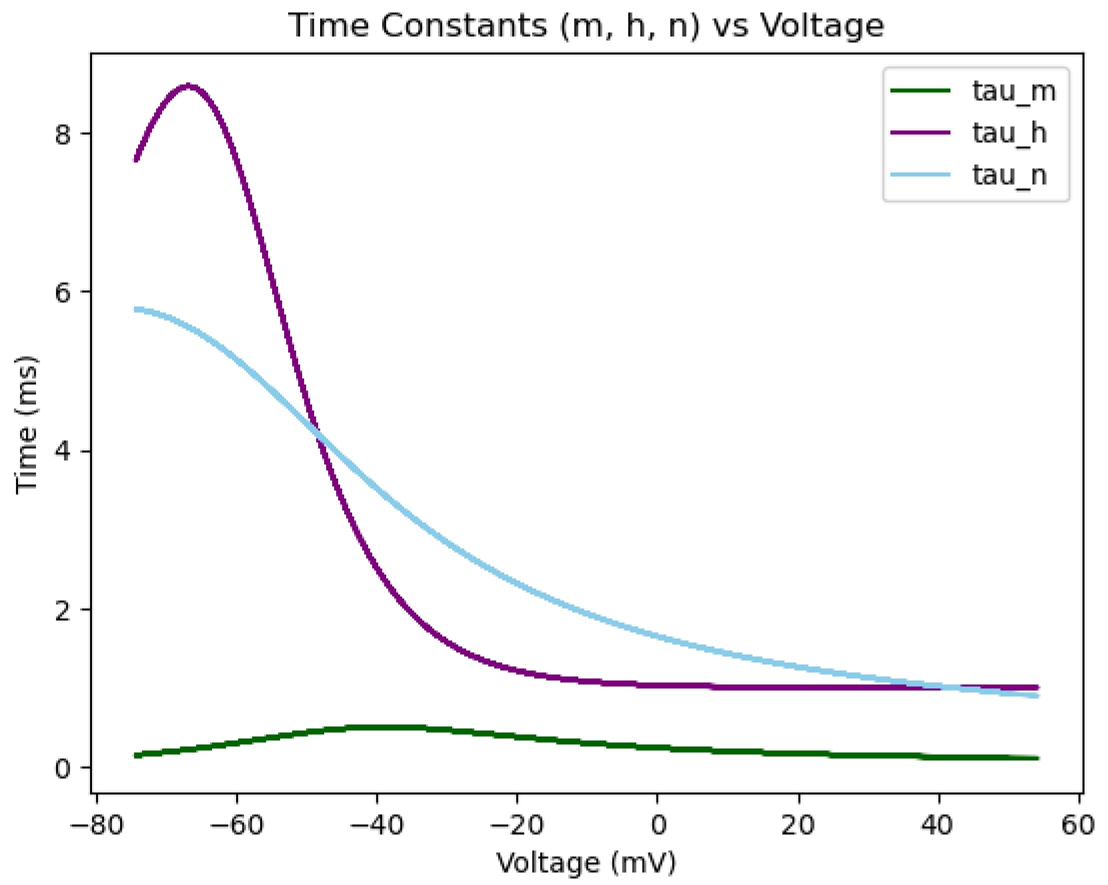


Figure 1B of HH Cheat Sheet:



---

## Reflections on Bonus Challenge

The graphs I created don't perfectly recreate Figure 1 because of my model's default args (which causes my model to run at voltage values [-80,60] mV, rather than [-100,40] mV as in ***Figure 1***.

However, these graphs do mimic the overall trend in the change in steady state and time constant for m, h, n as in Figure 1 at https://mark-kramer.github.io/BU-MA665-MA666/Readings/Hodgkin-Huxley_Cheat_Sheet.pdf.

- The steady state variables generally follow a sigmoid function in the first graph.
- The second graph shows a relatively lower time constant for m, with a higher time constant for variables h and n with peaks around -60mV.

---

## Challenge 2

**This section will address:**

1. Determine how the firing rate of the HH model varies with input current I.

2. Plot the firing rate versus I (i.e., plot the “f-I curve”).

---

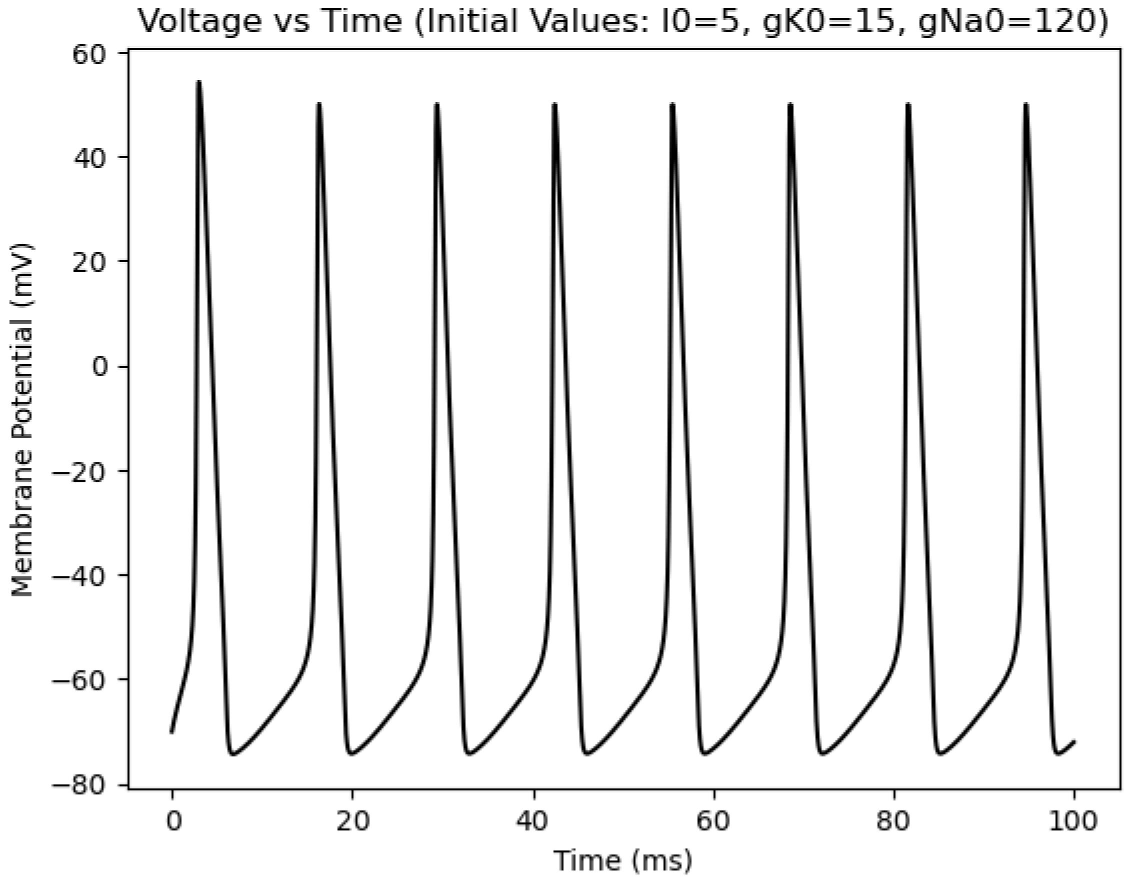
### Define a function which computes the firing rate using HH model.

In [13]:
```
"""
Define a function to compute the firing rate of a HH model.

Args:
I0 -- input current
T0 -- duration of simulation, to be converted to milliseconds by time step dt

Returns:
firing_rate --- firing rate (spikes per second)
t --- time array (ms)
V --- voltage array
"""

def HH_firingrate(I0, T0, gK0_i, gNa0_i):

    # Define biological constants
    gNa0 = gNa0_i   # sodium conductance [mS/cm^2]
    ENa  = 125;     # equilibrium or Nernst potential for sodium [mV]
    gK0  = gK0_i;   # potassium conductance [mS/cm^2]
    EK   = -12;     # equilibrium or Nernst potential for potassium [mV]
    gL0  = 0.3;     # leak conductance [mS/cm^2]
    EL   = 10.6;    # equilibrium or Nernst potential for leak [mV]

    # Define a maximum duration of the simulation in milliseconds
    dt = 0.01;   # time step
    T  = int(np.ceil(T0/dt))  # [ms]

    # Initialize an array of times, voltages, activation/inactivation variables (m, h, n)
    t = np.arange(0,T)*dt
    V = np.zeros([T,1])
    m = np.zeros([T,1])
    h = np.zeros([T,1])
```

```python
    n = np.zeros([T,1])

    # Define initial values for voltages, activation/inactivation variables (m, h, n)
    V[0]=-70.0
    m[0]=0.05
    h[0]=0.54
    n[0]=0.34

    # Apply a numerical method (forward Euler) to compute the next value in the arrays for:
    for i in range(0,T-1):

        # Voltage
        V[i+1] = V[i] + dt*(gNa0*m[i]**3*h[i]*(ENa-(V[i]+65)) + gK0*n[i]**4*(EK-(V[i]+65)) + gL0*(EL-(V[i]+65)) + I0);

        # Sodium activation
        m[i+1] = m[i] + dt*(alphaM(V[i])*(1-m[i]) - betaM(V[i])*m[i]);

        # Sodium inactivation
        h[i+1] = h[i] + dt*(alphaH(V[i])*(1-h[i]) - betaH(V[i])*h[i]);

        # Potassium activation
        n[i+1] = n[i] + dt*(alphaN(V[i])*(1-n[i]) - betaN(V[i])*n[i]);

    # Create voltage array (list of voltages)
    voltages=[]
    for i in range(V.size):
        voltages.append(V[i][0])


    ###############################

    # Compute the firing rate

    # initialize spike count at 0
    spike_count=0

    # find_peaks(voltages)[0] gives the indices in which there are peaks in the voltage array
    # find_peaks(voltages)[0].size gives the number of peaks in the voltage array
    for i in find_peaks(voltages)[0]:  # for each index in which there is a peak in the voltage array,
        if voltages[i] >= 0:           # if the peak is a positive voltage, then this is a spike
            spike_count+=1             # increase spike count by 1.

    firing_rate = spike_count/.1  # spikes/second

    return firing_rate,t,V
```

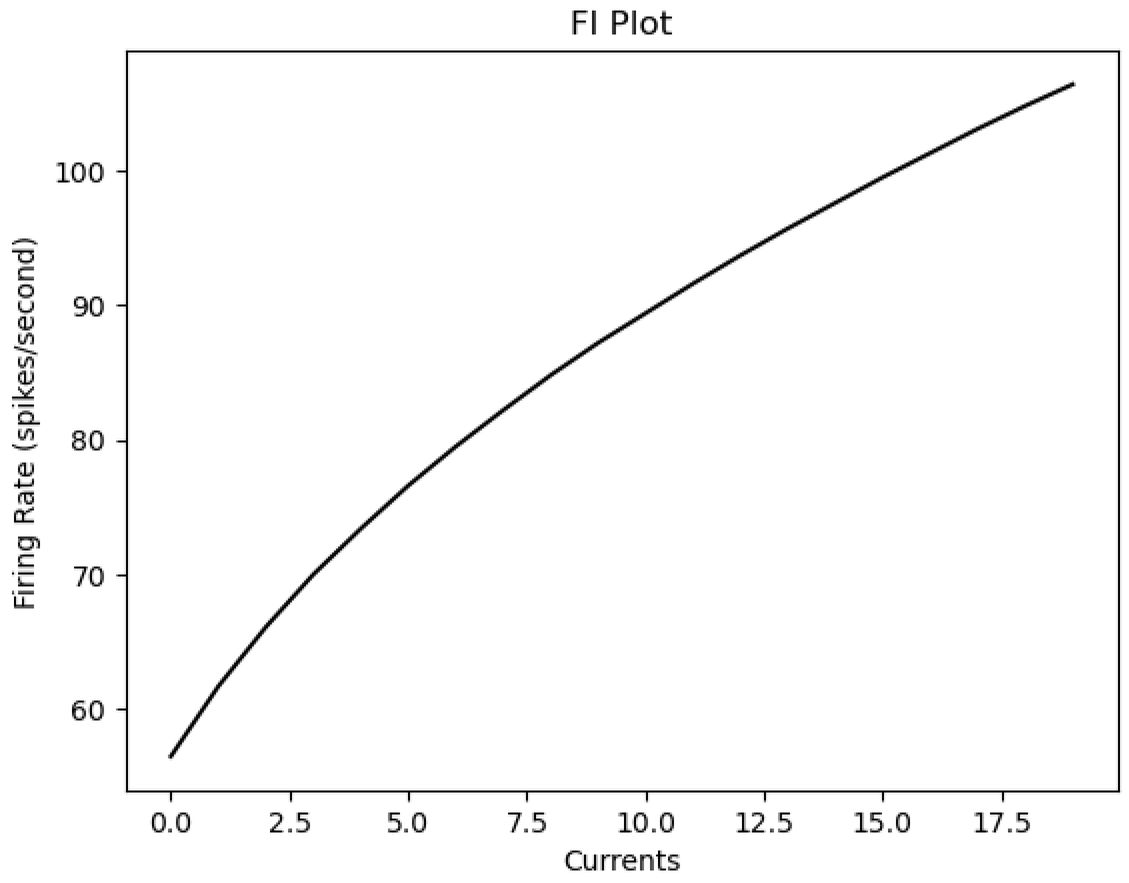
```
#################
# Call function HH_firingrate

# frate_i represents the firing rate for iniital input values: gK0=15, gNa0=120, I0=5, T0=100
# t_i represents time array for iniital input values: gK0=15, gNa0=120, I0=5, T0=100
# V_i represents voltage array for for iniital input values: gK0=15, gNa0=120, I0=5, T0=100
frate_i,t_i,V_i = HH_firingrate(5,100,15,120)

print("Firing rate: ", frate_i, " (initial  vals I0=5, gK0=15, gNa0=120)")

plt.figure()
plt.title("Voltage vs Time (Initial Values: I0=5, gK0=15, gNa0=120)")
plt.xlabel("Time (ms)")
plt.ylabel("Membrane Potential (mV)")
plt.plot(t_i,V_i, 'black')
plt.show()
```

```
Firing rate:  80.0  (initial  vals I0=5, gK0=15, gNa0=120)
```



```
In [99]: # Initialize arrays of currents, firing rates
arr_I = []
```

```
arr_frates = []
T0 = 10000

for i in range(0,20):
    arr_I.append(i)

# Recursively call the HH_firingrate(I0,T0) function.
for i in arr_I:
    arr_frates.append((HH_firingrate(i,T0,15,120)[0])/100)
    # print(arr_frates)

plt.figure()
plt.title("FI Plot")
plt.xlabel("Currents")
plt.ylabel("Firing Rate (spikes/second)")
plt.plot(arr_I, arr_frates, 'black')
plt.show()
```



### Answer questions from Challenge 2:

**1. Determine how the firing rate of the HH model varies with input current I.**

Overall, the firing rate increases with increased input current I. At lower input currents, change in input current results in a steeper increase in firing rate. At higher input currents, the change in input current resutls in relatively shallower increases in firing rate and the firing rate begins to level out at very high input currents (in which the neuron is still showing persistent firing).

**2. Plot the firing rate versus I (i.e., plot the "f-I curve").**

See the plot above.

---

## Challenge 3

**For Challenge 3, address:**

1. How does the firing rate of the HH model change as you increase the potassium conductance?
2. Provide a "simulation" explanation and a "physical" explanation.

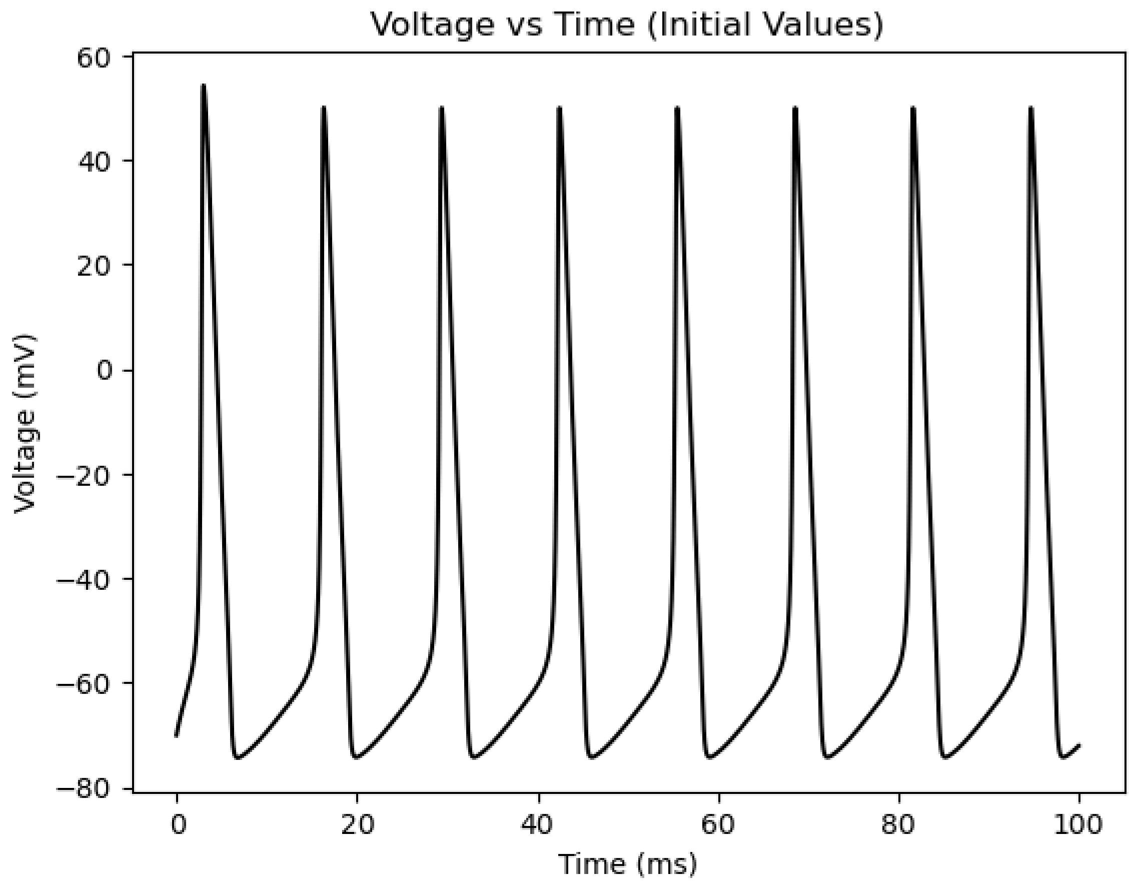
---

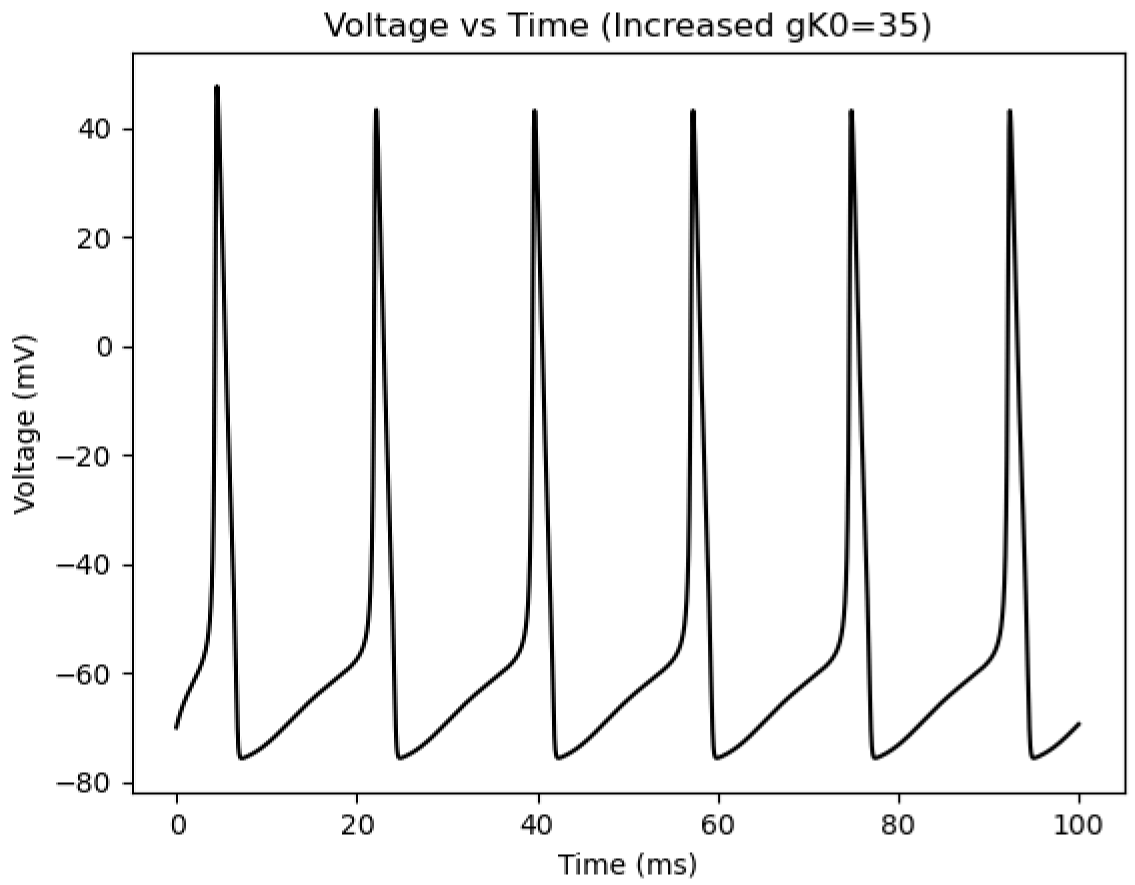
### Begin by calling HH_firingrate function using an increased value of gK0=35 (rather than gK0=15).

In [21]:
```python
# Call HH_firingrate(I0,T0,gK0_i,gNa0_i) function
# Increase gK0 from 15 to 35.

frate_gK0_35,t_gK0_35,V_gK0_35 = HH_firingrate(5,100,35,120)

print("Firing rate: ", frate_i, " (gK0=15) (initial values)")
print("Firing rate: ", frate_gK0_35, " (gK0=35) (increased gK0) \n")

plt.figure()
plt.title("Voltage vs Time (Initial Values)")
plt.plot(t_i,V_i, 'black')
plt.xlabel("Time (ms)")
plt.ylabel("Voltage (mV)")
plt.show()
```

```python
# Plot voltage vs time for increased gK0=35
plt.figure()
plt.title("Voltage vs Time (Increased gK0=35)")
plt.plot(t_gK0_35,V_gK0_35, 'black')
plt.xlabel("Time (ms)")
plt.ylabel("Voltage (mV)")
plt.show()
```

```
Firing rate:  80.0  (gK0=15) (initial values)
Firing rate:  60.0  (gK0=35) (increased gK0)
```





---

### Revisit Challenge 3 Questions:

**1. How does the firing rate of the HH model change as you increase the potassium conductance?**

From the simulation above, at relatively lower levels of potassium conductance (i.e. gK0=15), we can observe a higher firing rate of approximately 80 spikes per second. By keeping all other biological parameters the same but only changing gK0 to 35, we see a slower firing rate of approximately 60 spikes per second. With increased potassium conductance, the firing rate decreases. We also observe more depolarization in the lower potassium conductance spike train, in which APs depolarize to ~+50mV, whereas higher potassium conductance causes the neuron to depolarize to a lower voltage at ~+45mV.

**2. Provide a “simulation” explanation and a “physical” explanation.**

*Simulation explanation:* From the simulation above, the only changed values between the two plots above are the maximum conductance for potassium, gK0. Comparing the two plots, we see a slower firing rate in the plot with increased gK0, a steeper repolarization, and a slower recovery time. This is due to how gK0 is applied in the potassium component of the HH voltage equation, and the potassium gating variable, n, expressed as the probability n^4. Notably, n is a differential equation that is dependent on the voltage and time.

*Physical explanation:* From the plot titled "Conductances" above, the sodium conductance gNa peaks with the depolarization of the neuron, briefly followed by a rise in potassium conductance gK. In our initial Voltage vs Time plot, the maximum potassium conductance is much lower than the maximum sodium conductance. Potassium conductance represents the ease in which potassium ions can flow through potassium ion channels on the cell membrane. High potassium conductance indicates low potassium resistance, indicating a higher probability of open potassium channels. Higher potassium conductance also suggests potassium ions can flow more easily across the membrane. With increased conductance, it makes sense to see a sharper repolarization phase (when the n gating variable is highest), due to potassium ions flowing out of the cell more quickly. Since potassium ions are rushing out of the cell during repolarization/hyperpolarization, a higher potassium conductance also suggests a slower recovery time, causing the neuron to more slowly reach threshold potential to fire, causing a slower firing rate.

---

## Challenge 4

**In Challenge 4, we will address:**

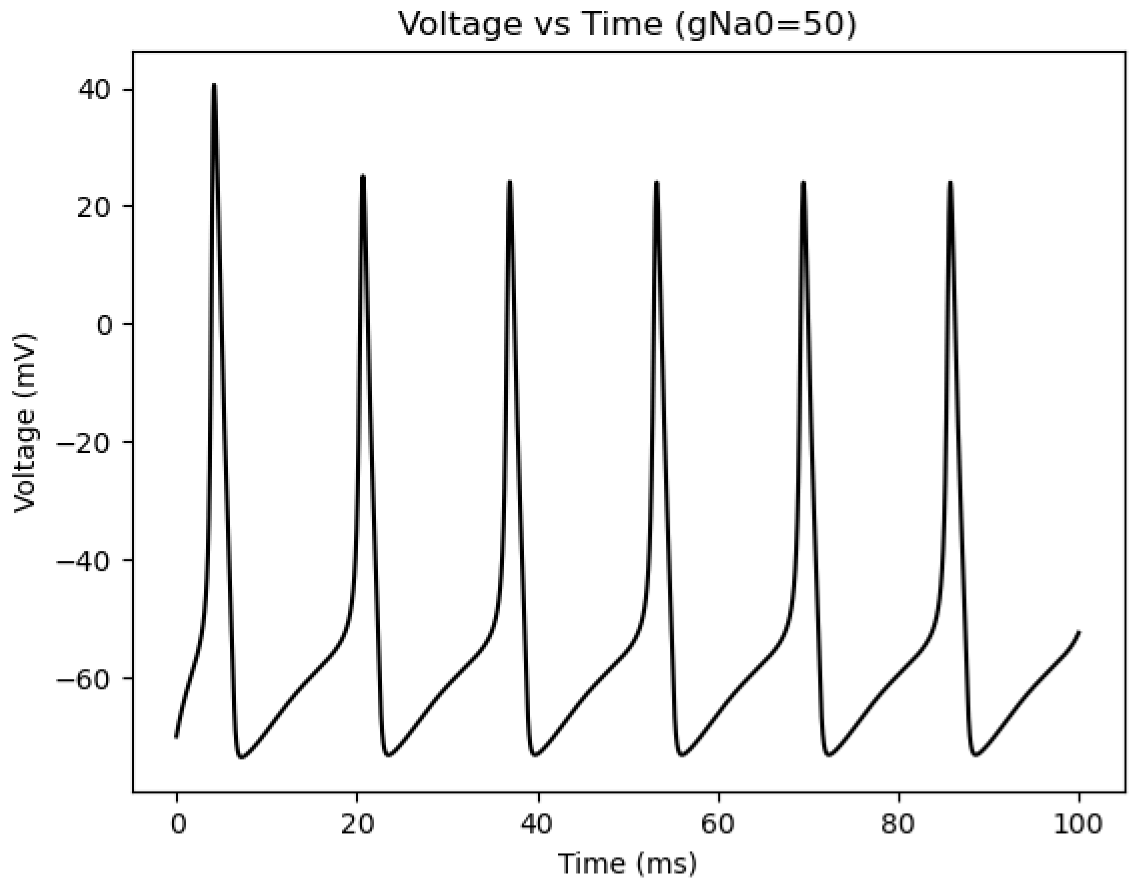
1. How does the firing rate of the HH model change as you increase sodium conductance?
2. Provide a “simulation” explanation and a “physical” explanation.

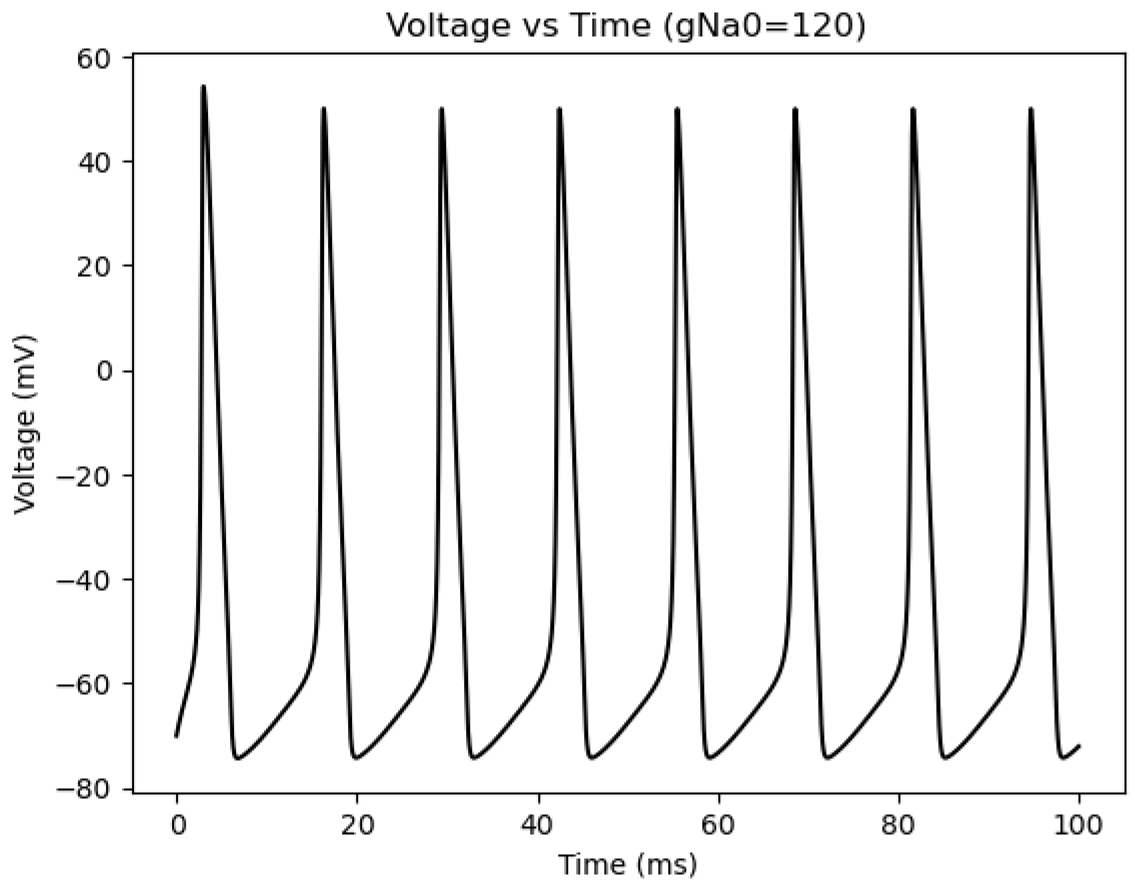
---

### Start by calling the HH_firingrate function with increased gNa0

In [15]:
```
# Call HH_firingrate(I0,T0,gK0_i,gNa0_i) function
# Increase gNa0 from 50 to 120.

frate_gNa0_50,t_gNa0_50,V_gNa0_50 = HH_firingrate(5,100,15,50)

print("Firing rate: ", frate_gNa0_50, " (gNa0=50) (decreased gNa0)")
print("Firing rate: ", frate_i, " (gNa0=120) (initial values)")

# Plot voltage vs time
plt.figure()
plt.title("Voltage vs Time (gNa0=50)")
plt.plot(t_gNa0_50,V_gNa0_50, 'black')
plt.xlabel("Time (ms)")
plt.ylabel("Voltage (mV)")
plt.show()

plt.figure()
plt.title("Voltage vs Time (gNa0=120)")
plt.plot(t_i,V_i, 'black')
plt.xlabel("Time (ms)")
plt.ylabel("Voltage (mV)")
plt.show()
```

```
Firing rate:  60.0  (gNa0=50) (decreased gNa0)
Firing rate:  80.0  (gNa0=120) (initial values)
```





---

### Revisit Challenge 4 Questions:

1. How does the firing rate of the HH model change as you increase sodium conductance?

   Increased sodium conductance causes the neuron to fire at a higher rate. Additionally, with increased sodium conductance, the neuron depolarizes to a more positive voltage at ~+50mV, compared to ~+25mV in the model with lower sodium conductance. The threshold is also lower for the model with higher sodium conductance.

2. Provide a “simulation” explanation and a “physical” explanation.

   *Simulation Explanation:* Increasing sodium conductance (variable gNa0 in the simulation) increases the firing rate. In the simulation, the sodium conductance, gNa, is equal to the (maximum sodium conductance; gNa0; a constant) times the (sodium activation gating probability; m^3) times the (sodium inactivation gating variable; h). m and h are also differential equations and dependent on voltage and time. The increase in the maximum sodium conductance, gNa0, causes an increase in the sodium conductance gNa, which increases the change in voltage over time during the repolarization and depolarization phases of an AP.

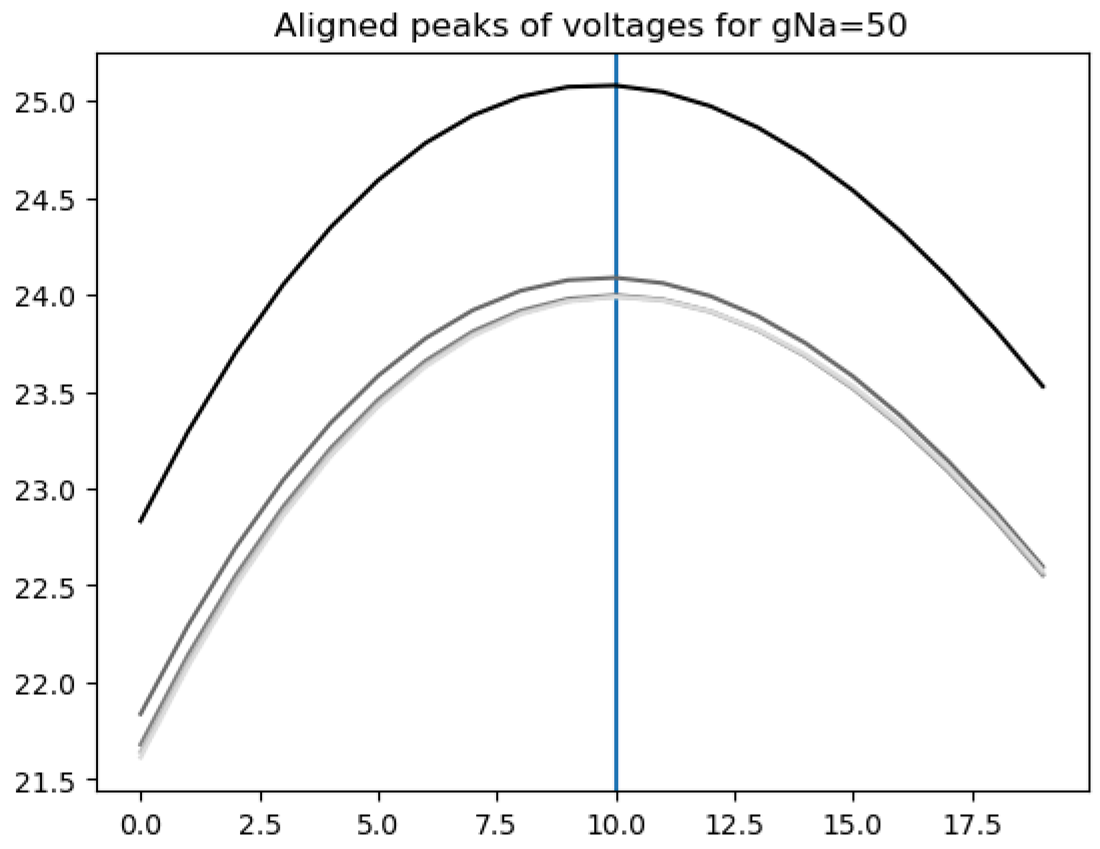
   *Physical Explanation:* During an AP, sodium conductance (gNa) peaks with the depolarization of the cell (see plot titled "Conductances" above). With higher sodium conductance, this causes the neuron to become more easily excitable, causing the neuron to fire at a lower threshold and to depolarize and repolarize at a faster rate.

Let's look closer at the spikes for gNa=50 and gNa=120.

In [91]:
```python
# Align the peak height of all spikes for gNa=50.

# Convert to lists
v1=[]
v2=[]
v3=[]
v4=[]
v5=[]
t1=[]
t2=[]
t3=[]
t4=[]
t5=[]

# V
for v in V_gNa0_50[1500:3000]:
    v1.append(v)

for v in V_gNa0_50[3000:4500]:
    v2.append(v)

for v in V_gNa0_50[4500:6000]:
    v3.append(v)

for v in V_gNa0_50[6000:7500]:
    v4.append(v)

for v in V_gNa0_50[7500:9000]:
    v5.append(v)

# Times
for t in t_gNa0_50[1500:3000]:
    t1.append(t)

for t in t_gNa0_50[3000:4500]:
    t2.append(t)

for t in t_gNa0_50[4500:6000]:
    t3.append(t)
```

```python
for t in t_gNa0_50[6000:7500]:
    t4.append(t)

for t in t_gNa0_50[7500:9000]:
    t5.append(t)

# Compute index of max val in voltage array
max1i = (v1).index(max(v1))
max2i = (v2).index(max(v2))
max3i = (v3).index(max(v3))
max4i = (v4).index(max(v4))
max5i = (v5).index(max(v5))

# Make arrays the same length
bounds=10
separate=1500
v1_align = v1[(max1i-bounds):(max1i+bounds)]
t1_align = t1[(max1i-bounds):(max1i+bounds)]
v2_align = v2[(max2i-bounds):(max2i+bounds)]
t2_align = t2[(max2i-bounds):(max2i+bounds)]
v3_align = v3[(max3i-bounds):(max3i+bounds)]
t3_align = t3[(max3i-bounds):(max3i+bounds)]
v4_align = v4[(max4i-bounds):(max4i+bounds)]
t4_align = t4[(max4i-bounds):(max4i+bounds)]
v5_align = v5[(max5i-bounds):(max5i+bounds)]
t5_align = t5[(max5i-bounds):(max5i+bounds)]

print("Blue line in plot below shows alignment of max peaks \n")
print("Max voltages: ", max(v1),max(v2),max(v3),max(v4),max(v5))

# Plot aligned
plt.figure()
plt.title("Aligned peaks of voltages for gNa=50")
plt.axvline(x=bounds)
plt.plot(v1_align,'black')
plt.plot(v2_align,'dimgray')
plt.plot(v3_align,'gray')
plt.plot(v4_align,'silver')
plt.plot(v5_align,'gainsboro')
plt.show()
```

```
Blue line in plot below shows alignment of max peaks

Max voltages:  [25.07971512] [24.08787838] [23.9979107] [23.98978336] [23.98895519]
```



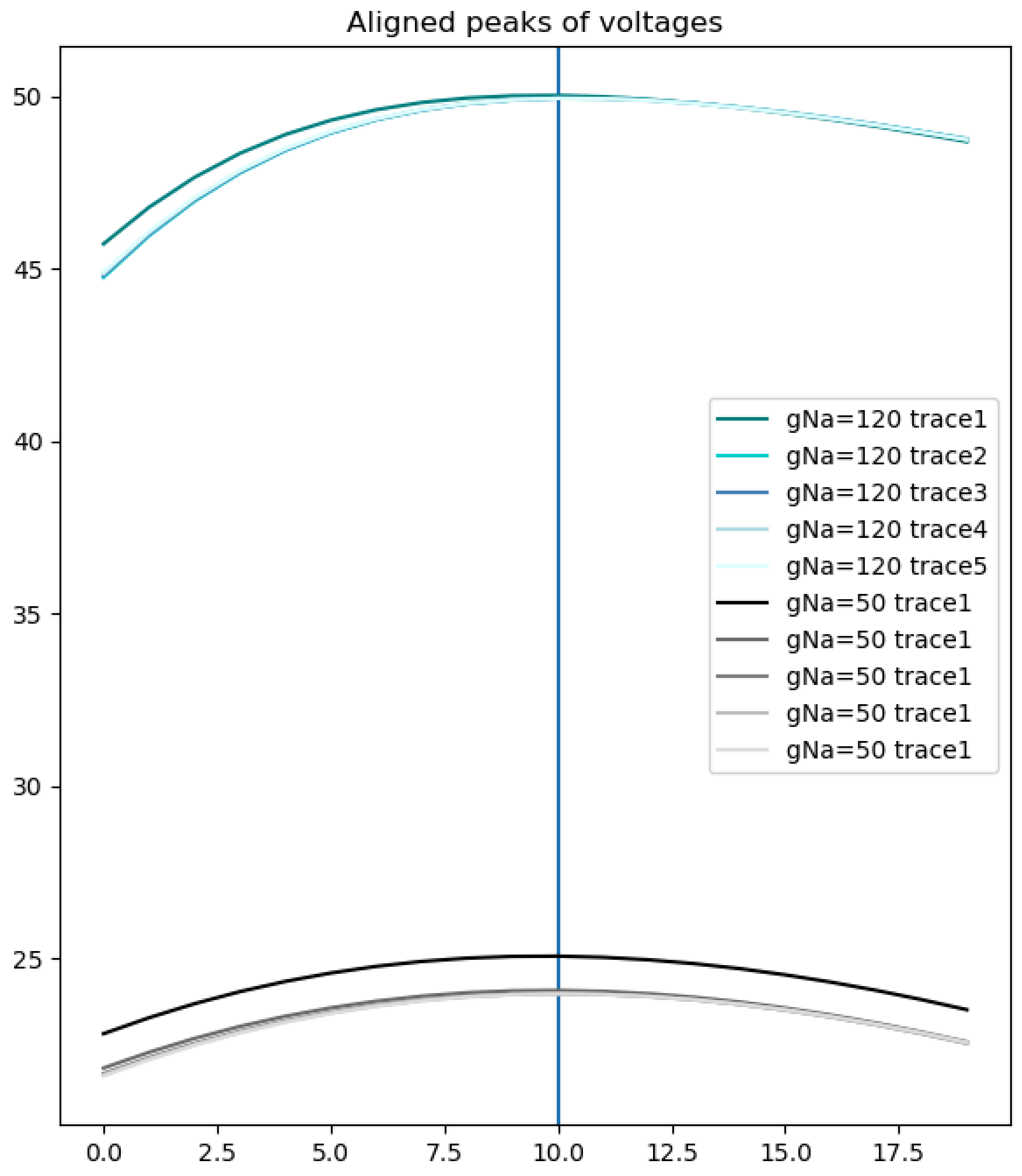
In [100…

```
# Align peak spikes for gNa=120.

# Convert to lists
v1h=[]
v2h=[]
v3h=[]
v4h=[]
v5h=[]
t1h=[]
t2h=[]
t3h=[]
t4h=[]
t5h=[]

# V
for v in V_i[1500:3000]:
    v1h.append(v)

for v in V_i[3000:4500]:
    v2h.append(v)
```

```
for v in V_i[4500:6000]:
    v3h.append(v)

for v in V_i[6000:7500]:
    v4h.append(v)

for v in V_i[7500:9000]:
    v5h.append(v)

# Times
for t in t_i[1500:3000]:
    t1h.append(t)

for t in t_i[3000:4500]:
    t2h.append(t)

for t in t_i[4500:6000]:
    t3h.append(t)

for t in t_i[6000:7500]:
    t4h.append(t)

for t in t_i[7500:9000]:
    t5h.append(t)

# Compute index of max val in voltage array
max1ih = (v1h).index(max(v1h))
max2ih = (v2h).index(max(v2h))
max3ih = (v3h).index(max(v3h))
max4ih = (v4h).index(max(v4h))
max5ih = (v5h).index(max(v5h))

# Make arrays the same length
boundsh=10
separateh=1500
v1_alignh = v1h[(max1ih-bounds):(max1ih+bounds)]
t1_alignh = t1h[(max1ih-bounds):(max1ih+bounds)]
v2_alignh = v2h[(max2ih-bounds):(max2ih+bounds)]
t2_alignh = t2h[(max2ih-bounds):(max2ih+bounds)]
v3_alignh = v3h[(max3ih-bounds):(max3ih+bounds)]
t3_alignh = t3h[(max3ih-bounds):(max3ih+bounds)]
v4_alignh = v4h[(max4ih-bounds):(max4ih+bounds)]
t4_alignh = t4h[(max4ih-bounds):(max4ih+bounds)]
v5_alignh = v5h[(max5ih-bounds):(max5ih+bounds)]
t5_alignh = t5h[(max5ih-bounds):(max5ih+bounds)]

print("Vertical blue line in plot below shows alignment of max peaks \n")
```

```
# Plot aligned
plt.figure(figsize=(7, 8))
plt.title("Aligned peaks of voltages")
plt.axvline(x=bounds)
plt.plot(v1_alignh,'teal',label='gNa=120 trace1')
plt.plot(v2_alignh,'darkturquoise',label='gNa=120 trace2')
plt.plot(v3_alignh,'steelblue',label='gNa=120 trace3')
plt.plot(v4_alignh,'powderblue',label='gNa=120 trace4')
plt.plot(v5_alignh,'lightcyan',label='gNa=120 trace5')
plt.plot(v1_align,'black',label='gNa=50 trace1')
plt.plot(v2_align,'dimgray',label='gNa=50 trace1')
plt.plot(v3_align,'gray',label='gNa=50 trace1')
plt.plot(v4_align,'silver',label='gNa=50 trace1')
plt.plot(v5_align,'gainsboro',label='gNa=50 trace1')
plt.legend()
plt.show()
```

```
Vertical blue line in plot below shows alignment of max peaks
```

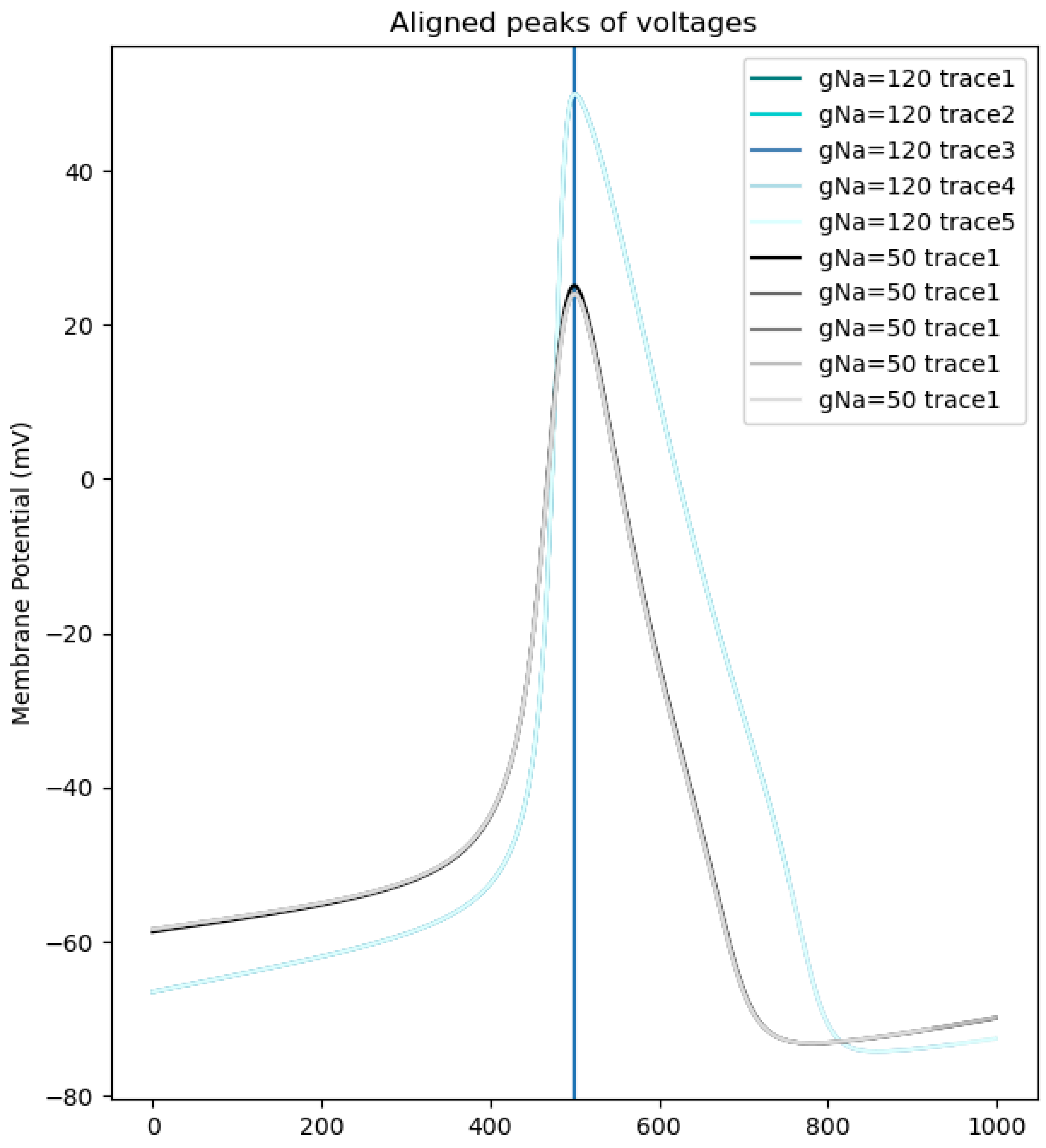


In [129…
```
# Align the peak height of all spikes for gNa=50.

# Convert to lists
v1=[]
v2=[]
v3=[]
```

```
v4=[]
v5=[]
t1=[]
t2=[]
t3=[]
t4=[]
t5=[]

# V
for v in V_gNa0_50[1500:3000]:
    v1.append(v)

for v in V_gNa0_50[3000:4500]:
    v2.append(v)

for v in V_gNa0_50[4500:6000]:
    v3.append(v)

for v in V_gNa0_50[6000:7500]:
    v4.append(v)

for v in V_gNa0_50[7500:9000]:
    v5.append(v)

# Times
for t in t_gNa0_50[1500:3000]:
    t1.append(t)

for t in t_gNa0_50[3000:4500]:
    t2.append(t)

for t in t_gNa0_50[4500:6000]:
    t3.append(t)

for t in t_gNa0_50[6000:7500]:
    t4.append(t)

for t in t_gNa0_50[7500:9000]:
    t5.append(t)

# Compute index of max val in voltage array
max1i = (v1).index(max(v1))
max2i = (v2).index(max(v2))
max3i = (v3).index(max(v3))
max4i = (v4).index(max(v4))
max5i = (v5).index(max(v5))

# Make arrays the same length
```

```
bounds=500
separate=1500
v1_align = v1[(max1i-bounds):(max1i+bounds)]
t1_align = t1[(max1i-bounds):(max1i+bounds)]
v2_align = v2[(max2i-bounds):(max2i+bounds)]
t2_align = t2[(max2i-bounds):(max2i+bounds)]
v3_align = v3[(max3i-bounds):(max3i+bounds)]
t3_align = t3[(max3i-bounds):(max3i+bounds)]
v4_align = v4[(max4i-bounds):(max4i+bounds)]
t4_align = t4[(max4i-bounds):(max4i+bounds)]
v5_align = v5[(max5i-bounds):(max5i+bounds)]
t5_align = t5[(max5i-bounds):(max5i+bounds)]

# Align peak spikes for gNa=120.

# Convert to lists
v1h=[]
v2h=[]
v3h=[]
v4h=[]
v5h=[]
t1h=[]
t2h=[]
t3h=[]
t4h=[]
t5h=[]

# V
for v in V_i[1500:3000]:
    v1h.append(v)

for v in V_i[3000:4500]:
    v2h.append(v)

for v in V_i[4500:6000]:
    v3h.append(v)

for v in V_i[6000:7500]:
    v4h.append(v)

for v in V_i[7500:9000]:
    v5h.append(v)

# Times
for t in t_i[1500:3000]:
    t1h.append(t)

for t in t_i[3000:4500]:
    t2h.append(t)

for t in t_i[4500:6000]:
    t3h.append(t)

for t in t_i[6000:7500]:
    t4h.append(t)

for t in t_i[7500:9000]:
    t5h.append(t)

# Compute index of max val in voltage array
max1ih = (v1h).index(max(v1h))
max2ih = (v2h).index(max(v2h))
max3ih = (v3h).index(max(v3h))
max4ih = (v4h).index(max(v4h))
max5ih = (v5h).index(max(v5h))

# Make arrays the same length
boundsh=500
separateh=1500
v1_alignh = v1h[(max1ih-bounds):(max1ih+bounds)]
t1_alignh = t1h[(max1ih-bounds):(max1ih+bounds)]
v2_alignh = v2h[(max2ih-bounds):(max2ih+bounds)]
t2_alignh = t2h[(max2ih-bounds):(max2ih+bounds)]
v3_alignh = v3h[(max3ih-bounds):(max3ih+bounds)]
t3_alignh = t3h[(max3ih-bounds):(max3ih+bounds)]
v4_alignh = v4h[(max4ih-bounds):(max4ih+bounds)]
t4_alignh = t4h[(max4ih-bounds):(max4ih+bounds)]
v5_alignh = v5h[(max5ih-bounds):(max5ih+bounds)]
t5_alignh = t5h[(max5ih-bounds):(max5ih+bounds)]

print("Vertical blue line in plot below shows alignment of max peaks \n")

# Plot aligned
plt.figure(figsize=(7, 8))
plt.title("Aligned peaks of voltages")
plt.axvline(x=bounds)
plt.ylabel('Membrane Potential (mV)')
plt.plot(v1_alignh,'teal',label='gNa=120 trace1')
plt.plot(v2_alignh,'darkturquoise',label='gNa=120 trace2')
plt.plot(v3_alignh,'steelblue',label='gNa=120 trace3')
plt.plot(v4_alignh,'powderblue',label='gNa=120 trace4')
plt.plot(v5_alignh,'lightcyan',label='gNa=120 trace5')
plt.plot(v1_align,'black',label='gNa=50 trace1')
plt.plot(v2_align,'dimgray',label='gNa=50 trace1')
plt.plot(v3_align,'gray',label='gNa=50 trace1')
plt.plot(v4_align,'silver',label='gNa=50 trace1')
plt.plot(v5_align,'gainsboro',label='gNa=50 trace1')
plt.legend()
plt.show()
```

```
Vertical blue line in plot below shows alignment of max peaks
```


Aligned peaks of voltages
Membrane Potential (mV)
gNa=120 trace1
gNa=120 trace2
gNa=120 trace3
gNa=120 trace4
gNa=120 trace5
gNa=50 trace1
gNa=50 trace1
gNa=50 trace1
gNa=50 trace1
gNa=50 trace1