BE 521: Homework 2 Questions

Modeling Neurons

Spring 2025

47 points

Due: Feb 6th, 2025

Objective: Computational modeling of neurons. We gratefully acknowledge Dr. Vijay Balasubramanium (UPenn) for many of the questions in this homework.

AI Usage Notice

The use of artificial intelligence tools (e.g., large language models, code assistants) is permitted. However, students must explicitly state the specific ways AI was used in completing their work. Failure to disclose AI usage may result in an oral examination to assess understanding, at the discretion of Dr. Litt.

If AI was used in the completion of this assignment, please provide a statement below:

[Enter your statement here]

In [63]:
```
!jupyter nbconvert --to html Prakriti_HW2.ipynb
```

```
[NbConvertApp] Converting notebook Prakriti_HW2.ipynb to html
[NbConvertApp] WARNING | Alternative text is missing on 7 image(s).
[NbConvertApp] Writing 899526 bytes to Prakriti_HW2.html
```

In [73]:
```
#Set up the notebook environment
import numpy as np
import matplotlib.pyplot as plt

!pip install git+https://github.com/ieeg-portal/ieegpy.git # Install ieegpy toolbox directly from github
from ieeg.auth import Session
```

```
Collecting git+https://github.com/ieeg-portal/ieegpy.git
  Cloning https://github.com/ieeg-portal/ieegpy.git to /tmp/pip-req-build-5s5trk2k
  Running command git clone --filter=blob:none --quiet https://github.com/ieeg-portal/ieegpy.git /tmp/pip-req-bu
ild-5s5trk2k
  Resolved https://github.com/ieeg-portal/ieegpy.git to commit 080bfa42a8503380ef164b5e7b116613f75073bb
  Preparing metadata (setup.py) ... done
Requirement already satisfied: deprecation in /usr/local/lib/python3.11/dist-packages (from ieeg==1.6) (2.1.0)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from ieeg==1.6) (2.32.3)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from ieeg==1.6) (1.26.4)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (from ieeg==1.6) (2.2.2)
Requirement already satisfied: pennprov==2.2.4 in /usr/local/lib/python3.11/dist-packages (from ieeg==1.6) (2.2.
4)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from pennprov==2.2
.4->ieeg==1.6) (2025.1.31)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.11/dist-packages (from pennprov==2
.2.4->ieeg==1.6) (2.8.2)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.11/dist-packages (from pennprov==2.2.4->ieeg=
=1.6) (1.17.0)
Requirement already satisfied: urllib3>=1.23 in /usr/local/lib/python3.11/dist-packages (from pennprov==2.2.4->i
eeg==1.6) (2.3.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from deprecation->ieeg==1.6
) (24.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas->ieeg==1.6)
(2025.1)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas->ieeg==1.6
) (2025.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from request
s->ieeg==1.6) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->ieeg==1.6
) (3.10)
```

1. Basic Membrane and Equilibrium Potentials (9 pts)

Before undertaking this section, you may find it useful to read pg. 153-161 of Dayan & Abbott's *Theoretical Neuroscience* (the relevant section of which, Ch. 5, is posted with this homework).

Recall that the potential difference V_T when a mole of ions crosses a cell membrane is defined by the universal gas constant R = 8.31 J/molꞏK, the temperature T (in Kelvin), and Faraday's constant F = 96,480 C/mol.

$$V_T = \frac{RT}{F}$$

Calculate V_T at human physiological temperature (37°C). (1 pt)

In [74]:
```
#Your code here
r =  8.31 #J/mol
t = 310.15 #37 degC - convet to kelvin
f = 96480 #C/mol

vt = (r*t)/f
print(vt)
```

```
0.026713790422885575
```

Your answer here

2

Use this value V_T to calculate the Nernst equilibrium potentials (in mV) for the K^+, Na^+, and Cl^- ions, given the following cytoplasm and extracellular concentrations in the squid giant axon:

- K^+: (120, 4.5)
- Na^+: (15, 145)
- Cl^-: (12, 120)

The first number is the cytoplasmic concentration and the second number is the extracelluar concentration (in mM). (2pt)

In [75]:
```
# which number is inside and outside?????????????
# z is the charge of the ion
#nersnt equation
# Potassium in mM

inside_k, outside_k = 120, 4.5  # K+
inside_na, outside_na = 15, 145  # Na+
inside_cl, outside_cl = 12, 120  # Cl-

# Charges
z_k, z_na, z_cl = 1, 1, -1

# Compute Nernst potentials (in volts)
E_k = (vt / z_k) * np.log(outside_k / inside_k)
E_na = (vt / z_na) * np.log(outside_na / inside_na)
E_cl = (vt / z_cl) * np.log(outside_cl / inside_cl)

# Convert to mV
E_k_mV = E_k * 1000
E_na_mV = E_na * 1000
E_cl_mV = E_cl * 1000

E_k_mV, E_na_mV, E_cl_mV
```

Out[75]:
```
(-87.7124427106941, 60.60513665862865, -61.51077560510343)
```

Your answer here

3

3a

Use the Goldman equation,

$$V_m = V_T ln(\frac{P_K \cdot [K^+]_{out} + P_{Na} \cdot [Na^+]_{out} + P_{Cl} \cdot [Cl^-]_{in}}{P_K \cdot [K^+]_{in} + P_{Na} \cdot [Na^+]_{in} + P_{Cl} \cdot [Cl^-]_{out}})$$

to calculate the resting membrane potential, V_m, assuming that the ratio of membrane permeabilities

$$P_K : P_{Na} : P_{Cl} = 1.0 : 0.045 : 0.45$$

Use the ion concentrations given in question 1.2. (2 pts)

In [76]:
```
# Permeability ratios - is this how we take it??
P_K = 1.0
P_Na = 0.045
```

```
P_Cl = 0.45

# Compute numerator and denominator for the GHK equation
numerator = (P_K * outside_k) + (P_Na * outside_na) + (P_Cl * inside_cl)
denominator = (P_K * inside_k) + (P_Na * inside_na) + (P_Cl * outside_cl)

# Compute membrane potential Vm (in volts)
V_m = vt * np.log(numerator / denominator)

# Convert to mV
V_m_mV = V_m * 1000
V_m_mV
```

Out[76]: -63.15467406939219

Your answer here

3b

Calculate the membrane potential at the peak action potential, assuming a permeability ratio of 1.0 : 11 : 0.45, again using the ion concentrations given above in Question 1.2 (2 pts).

In [77]:
```
import numpy as np

# Given constants
R = 8.31  # J/(mol·K)
T = 310.15  # K (37°C)
F = 96480  # C/mol

# Compute V_T
V_T = (R * T) / F

# Ion concentrations (in mM)
inside_k, outside_k = 120, 4.5  # K+
inside_na, outside_na = 15, 145  # Na+
inside_cl, outside_cl = 12, 120  # Cl-

# Updated permeability ratios at peak action potential
P_K_peak = 1.0
P_Na_peak = 11
P_Cl_peak = 0.45

# Compute numerator and denominator for the GHK equation at peak action potential
numerator_peak = (P_K_peak * outside_k) + (P_Na_peak * outside_na) + (P_Cl_peak * inside_cl)
denominator_peak = (P_K_peak * inside_k) + (P_Na_peak * inside_na) + (P_Cl_peak * outside_cl)

# Compute membrane potential Vm at peak action potential (in volts)
V_m_peak = V_T * np.log(numerator_peak / denominator_peak)

# Convert to mV
V_m_peak_mV = V_m_peak * 1000
V_m_peak_mV
```

Out[77]: 41.53504535798821

Your answer here

4

The amplitudes of the multi-unit signals in HW0 and local field potentials (LFPs) in HW1 had magnitudes on the order of 10 to 100 microvolts. The voltage at the peak of the action potential (determined using the Goldman equation above) has a magnitude on the order of 10 millivolts. Briefly explain why we see this difference in magnitude.

Hint 1: Voltage is the difference in electric potential between two points. What are the two points for our voltage measurement in the multi-unit and LFP signals? What are the two points for our voltage measurements of the action potential?

Hint 2: The resistance of the neuronal membrane is typically much higher than the resistance of the extracellular fluid. (2 pts)

The difference in magnitude between **multi-unit signals / LFPs (10-100 µV)** and the **action potential peak (tens of mV)** arises due to the **locations of voltage measurement** and the **differences in resistance** between the intracellular and extracellular environments.

1. **Measurement Points:**
    - **Action potential voltage** is measured **across the neuronal membrane**—between the intracellular space (inside the neuron) and the extracellular space (outside the neuron). This directly captures the large voltage changes due to ion flux across the membrane.
    - **Multi-unit signals and LFPs** are recorded **extracellularly**, meaning they measure voltage differences in the **extracellular**

space between different points in the brain. These signals reflect summed activity from multiple neurons rather than direct intracellular voltage changes.

2. **Resistance Differences:**

   - The **neuronal membrane** has a **high resistance**, which leads to large voltage changes when ion channels open and close during an action potential.
   - The **extracellular space** has a **lower resistance**, allowing current to spread out more, which results in much smaller recorded voltages for multi-unit activity and LFPs.

Thus, action potentials measured intracellularly appear large, while extracellularly recorded signals (multi-unit & LFP) appear much smaller because they represent the **sum of many neurons' activity spread through a lower-resistance medium**.

In [77]:

Your answer here

2. Integrate and Fire Model (38 pts)

You may find it useful to read pg. 162-166 of Dayan and Abbott for this link textsection. The general differential equation for the integrate and fire model is

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

where $\tau_m = 10\,\text{ms}$ is the membrane time constant, describing how fast the current is leaking through the membrane, V_m in this case is constant and represents the resting membrane potential (which you have already calculated in question 1.3.a), and $V(t)$ is the actual membrane potential as a function of time. $R_m = 10^7\,\Omega$ is the constant total membrane resistance, and $I_e(t)$ is the fluctuating incoming current. Here, we do not explicitly model the action potentials (that's Hodgkin-Huxley) but instead model the neuron's behavior leading up to and after the action potential.

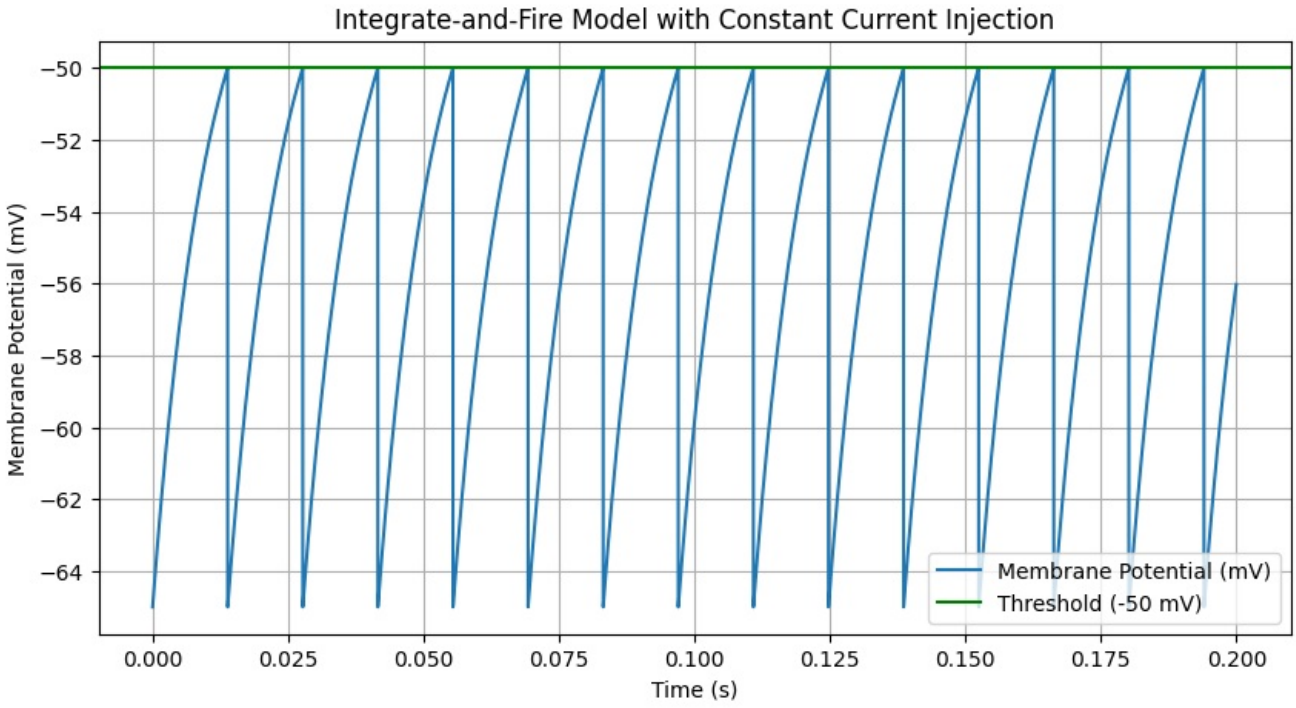
Use a $\Delta t = 10\,\mu\text{s}$ (Δt is the discrete analog of the continuous dt). Remember, one strategy for modeling differential equations like this is to start with an initial condition (here, $V(0) = V_m$), then calculate the function change (here, ΔV, the discrete analog to dV) and then add it to the function (here, $V(t)$) to get the next value at $t + \Delta t$. Once/if the membrane potential reaches a certain threshold ($V_{th} = -50\,\text{mV}$), you will say that an action potential has occurred and reset the potential back to its resting value.

1

Model the membrane potential with a constant current injection (i.e., $I_e(t) = I_e = 2\text{nA}$). Plot your membrane potential as a function of time to show at least a handful of "firings." (8 pts)

In [78]:
```
import numpy as np
import matplotlib.pyplot as plt

# Given parameters
tau_m = 10e-3  # Membrane time constant (s)
V_rest = -65e-3  # Resting membrane potential (V)
V_th = -50e-3  # Threshold potential (V)
Rm = 1e7  # Membrane resistance (ohms)
Ie = 2e-9  # Input current (A)

dt = 10e-6  # Time step (s)
total_time = 0.2  # Total simulation time (s)
n_steps = int(total_time / dt)  # Number of time steps

# Initialize membrane potential array
V = np.zeros(n_steps)
V[0] = V_rest  # Initial condition

time = np.arange(0, total_time, dt)

# Simulation loop
for i in range(1, n_steps):
    dV = (V_rest - V[i-1] + Rm * Ie) * (dt / tau_m)
    V[i] = V[i-1] + dV

    # If threshold is reached, reset potential
    if V[i] >= V_th:
        V[i] = V_rest

# Plot the results
plt.figure(figsize=(10, 5))

# Plot the membrane potential (V in mV)
plt.plot(time, V * 1e3, label="Membrane Potential (mV)")  # Convert V to mV for plotting
```

```
# Plot the threshold line
plt.axhline(y=V_th * 1e3, color="g", linestyle='-', label="Threshold (-50 mV)")

# Adding labels, title, and grid
plt.xlabel("Time (s)")
plt.ylabel("Membrane Potential (mV)")
plt.title("Integrate-and-Fire Model with Constant Current Injection")
plt.grid()

# Add the legend
plt.legend()

# Show the plot
plt.show()
```



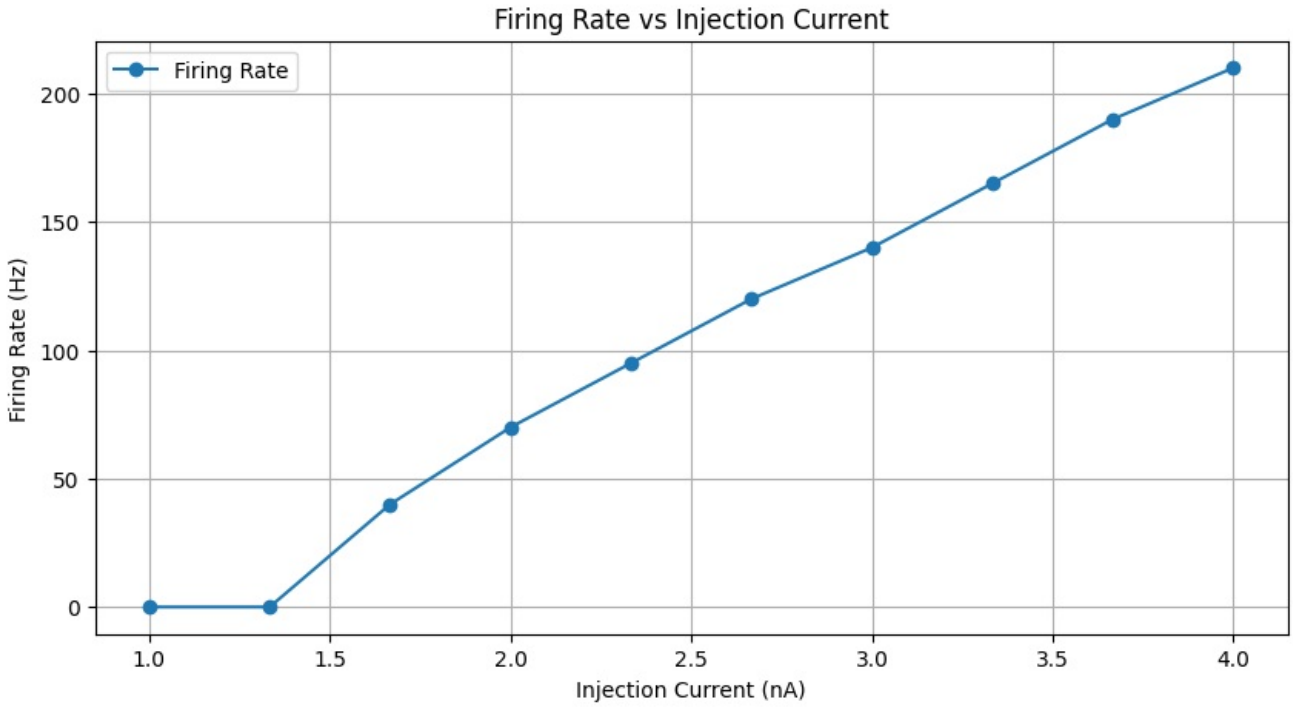
Your answer here

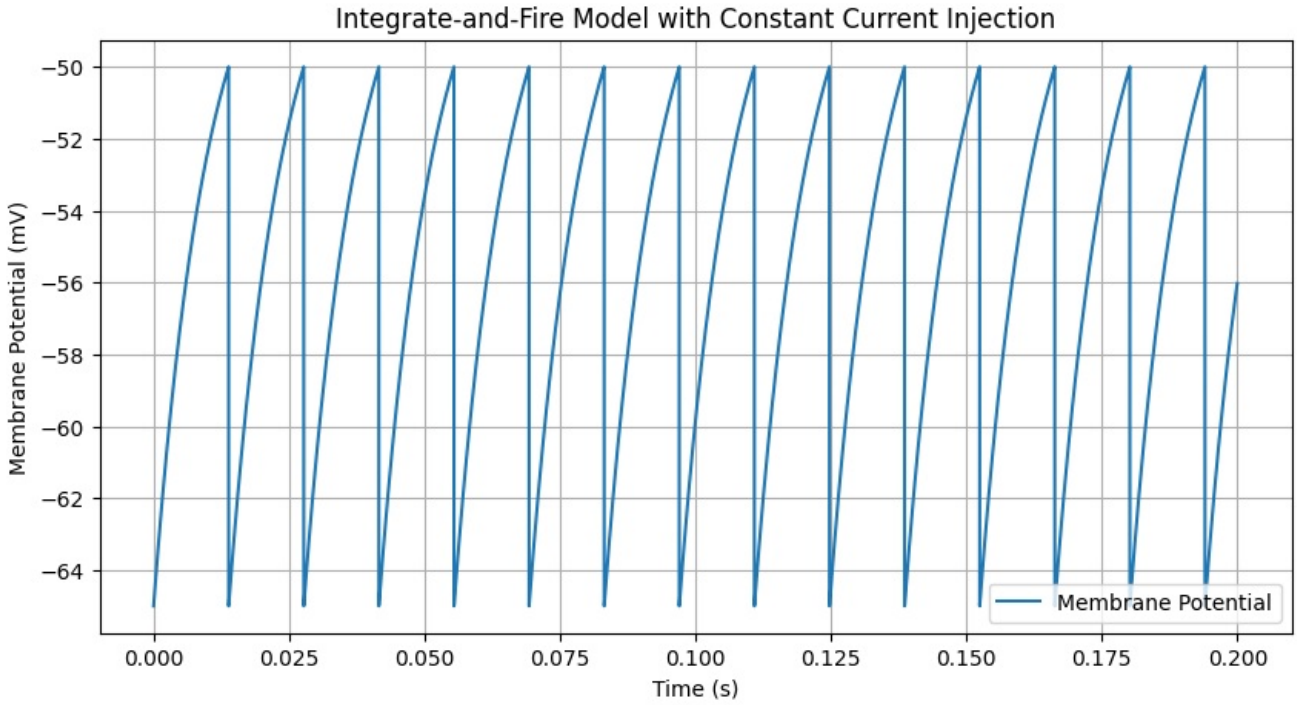
2

Produce a plot of firing rate (in Hz) versus injection current, over the range of 1-4 nA. (4 pts)

In [79]:
```
import numpy as np
import matplotlib.pyplot as plt

# Given parameters
tau_m = 10e-3  # Membrane time constant (s)
V_rest = -65e-3  # Resting membrane potential (V)
V_th = -50e-3  # Threshold potential (V)
Rm = 1e7  # Membrane resistance (ohms)

dt = 10e-6  # Time step (s)
total_time = 0.2  # Total simulation time (s)
n_steps = int(total_time / dt)  # Number of time steps

# Injection currents to test
Ie_values = np.linspace(1e-9, 4e-9, 10)  # Range from 1 to 4 nA
firing_rates = []

for Ie in Ie_values:
    V = np.zeros(n_steps)
    V[0] = V_rest  # Initial condition
    spike_count = 0

    for i in range(1, n_steps):
        dV = (V_rest - V[i-1] + Rm * Ie) * (dt / tau_m)
        V[i] = V[i-1] + dV

        # If threshold is reached, reset potential
        if V[i] >= V_th:
            V[i] = V_rest
            spike_count += 1
```

```
    firing_rate = spike_count / total_time  # Hz
    firing_rates.append(firing_rate)

# Plot firing rate vs injection current
plt.figure(figsize=(10, 5))
plt.plot(Ie_values * 1e9, firing_rates, marker='o', label="Firing Rate")  # Convert A to nA for plotting
plt.xlabel("Injection Current (nA)")
plt.ylabel("Firing Rate (Hz)")
plt.title("Firing Rate vs Injection Current")
plt.grid()

# Add the legend for firing rate plot
plt.legend()

# Show the plot
plt.show()

# Plot membrane potential for a single current injection example
Ie = 2e-9  # Input current (A)
V = np.zeros(n_steps)
V[0] = V_rest  # Initial condition
time = np.arange(0, total_time, dt)

for i in range(1, n_steps):
    dV = (V_rest - V[i-1] + Rm * Ie) * (dt / tau_m)
    V[i] = V[i-1] + dV

    # If threshold is reached, reset potential
    if V[i] >= V_th:
        V[i] = V_rest

plt.figure(figsize=(10, 5))
plt.plot(time, V * 1e3, label="Membrane Potential")  # Convert V to mV for plotting
plt.xlabel("Time (s)")
plt.ylabel("Membrane Potential (mV)")
plt.title("Integrate-and-Fire Model with Constant Current Injection")
plt.grid()

# Add the legend for membrane potential plot
plt.legend()

# Show the plot
plt.show()
```





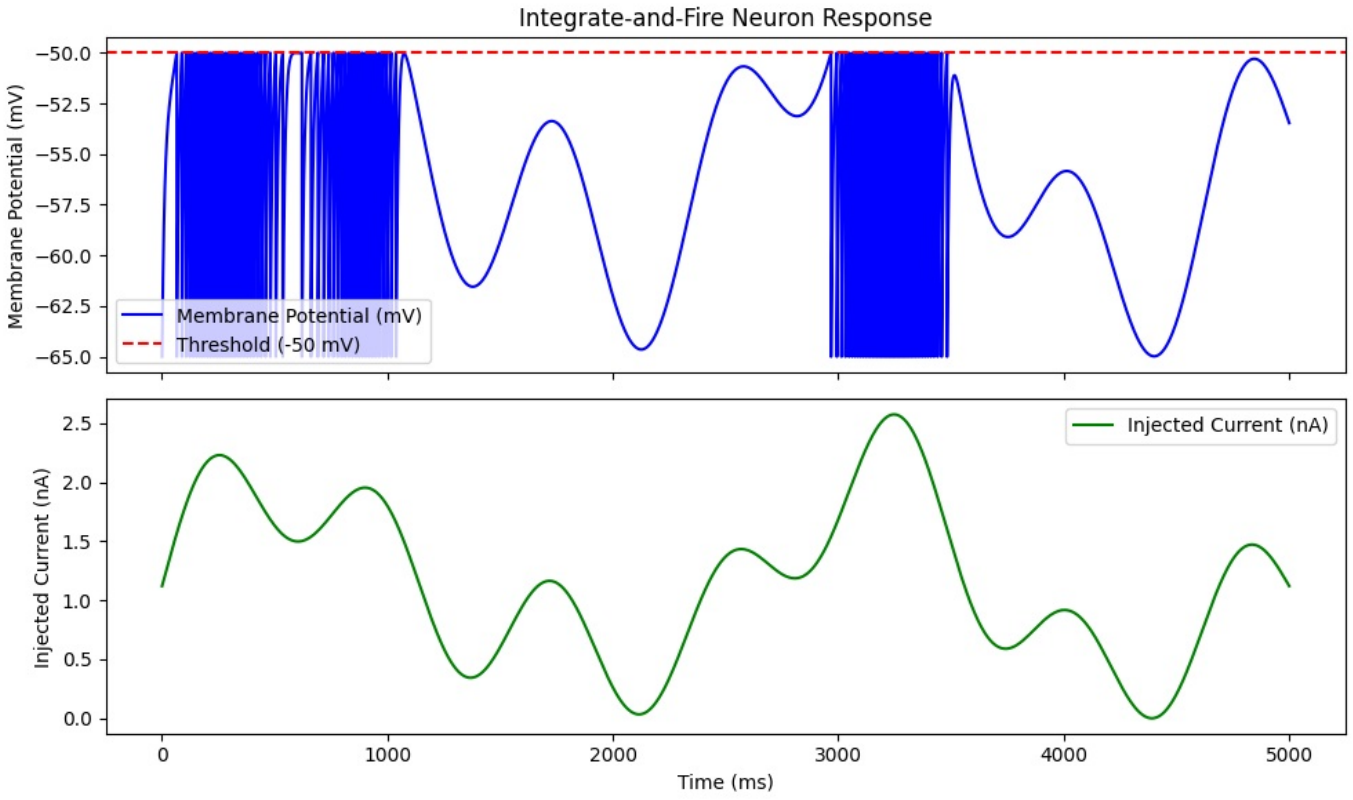
Your answer here

3

`I521_A0002_D001` contains a dynamic current injection in nA. Plot the membrane potential of your neuron in response to this variable injection current. Use `plt.subplots()` to place the plot of the membrane potential above the injection current so that they both have the same time axis. (Hint: the sampling frequency of the current injection data is different from the sampling frequency ($\frac{1}{\Delta t}$) that we used above.) (4 pts)

In [80]:
```python
import numpy as np
import matplotlib.pyplot as plt

# Given parameters
tau_m = 10e-3  # Membrane time constant (s)
V_rest = -65e-3  # Resting membrane potential (V)
V_th = -50e-3  # Threshold potential (V)
Rm = 1e7  # Membrane resistance (ohms)
dt = 10e-6  # Time step (s)
total_time = 0.2  # Total simulation time (s)
n_steps = int(total_time / dt)  # Number of time steps

# Load the dataset
with open('/content/pra_ieeglogin(9).bin', 'r') as f:
    session = Session('prasadpr', f.read())

dataset = session.open_dataset('I521_A0002_D001')

# Extract current injection data
key = list(dataset.ts_details.keys())[0]
channels = dataset.get_channel_indices([key])[0]
start_time = 0
duration = 500000
I_e_raw = dataset.get_data(start_time, duration, [channels]).flatten() * 1e-9  # Convert from nA to A

# Time vector for the current injection

time = np.arange(0, len(I_e_raw)* dt, dt)
min_length = min(len(time), len(I_e_raw))
time = time[:min_length]
```

```
I_e_raw = I_e_raw[:min_length]

# Initialize membrane potential array
V = np.zeros_like(time)
V[0] = V_rest  # Initial condition

# Simulation loop (I&F model)
for i in range(1,len(time)):
    dV = (V_rest - V[i-1] + Rm * I_e_raw[i-1]) * (dt / tau_m)
    V[i] = V[i-1] + dV

    # If threshold is reached, reset potential
    if V[i] >= V_th:
        V[i] = V_rest

# Time vector for the simulation
# time_sim = np.arange(0, total_time, dt)

# Plot the results
fig, axes = plt.subplots(2, 1, figsize=(10, 6), sharex=True)

# Plot the membrane potential
axes[0].plot(time * 1000, V * 1000, label="Membrane Potential (mV)", color='b')
axes[0].axhline(y=V_th * 1000, color='r', linestyle='--', label="Threshold (-50 mV)")
axes[0].set_ylabel("Membrane Potential (mV)")
axes[0].legend()
axes[0].set_title("Integrate-and-Fire Neuron Response")

# Plot the injected current
axes[1].plot(time * 1000, I_e_raw * 1e9, label="Injected Current (nA)", color='g')
axes[1].set_xlabel("Time (ms)")
axes[1].set_ylabel("Injected Current (nA)")
axes[1].legend()

plt.tight_layout()
plt.show()
```



Your answer here

4

Real neurons have a refractory period after an action potential that prevents them from firing again right away. We can include this behavior in the model by adding a spike-rate adaptation conductance term, $g_{sra}(t)$ (modeled as a potassium conductance), to the model

$$\tau_m \frac{dV}{dt} = V_m - V(t) - r_m g_{sra}(t)(V(t) - V_K) + R_m I_e(t)$$

where

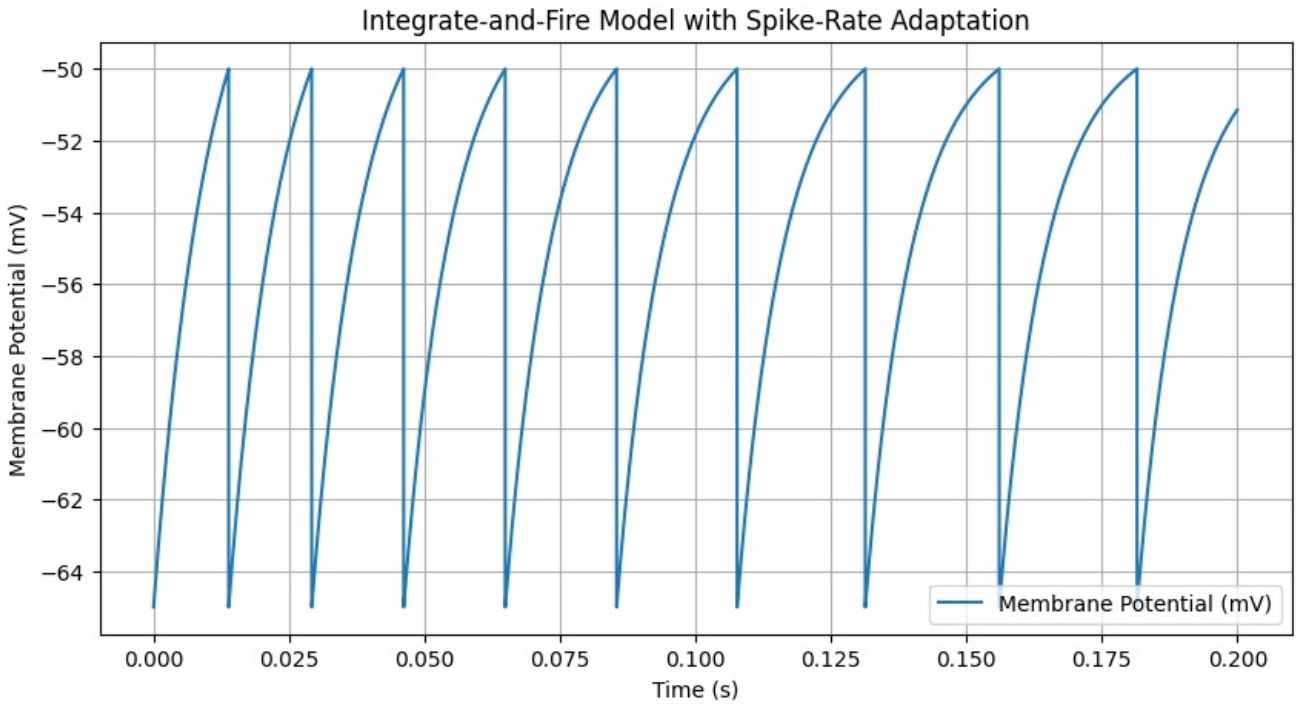
$$\tau_{sra}\frac{dg_{sra}(t)}{dt} = -g_{sra}(t).$$

Every time an action potential occurs, we increase g_{sra} by a certain constant amount, $g_{sra} = g_{sra} + \Delta g_{sra}$. Use $r_m \Delta g_{sra} = 0.06$. Use a conductance time constant of $\tau_{sra} = 100\,\text{ms}$, a potassium equilibrium potential of $V_K = -70\,\text{mV}$, and $g_{sra}(0) = 0$. (Hint: How can you use the $r_m \Delta g_{sra}$ value to update voltage and conductance separately in your simulation?)

4a

Implement this addition to the model (using the same other parameters as in question 2.1) and plot the membrane potential over 200 ms. (8 pts)

In [81]:
```
import numpy as np
import matplotlib.pyplot as plt

# Given parameters
tau_m = 10e-3  # Membrane time constant (s)
V_rest = -65e-3  # Resting membrane potential (V)
V_th = -50e-3  # Threshold potential (V)
Rm = 1e7  # Membrane resistance (ohms)
V_K = -70e-3  # Potassium equilibrium potential (V)
tau_sra = 100e-3  # Spike-rate adaptation time constant (s)
delta_g_sra = 0.06 / Rm  # Increase in g_sra per spike (S)

dt = 10e-6  # Time step (s)
total_time = 0.2  # Total simulation time (s)
n_steps = int(total_time / dt)  # Number of time steps

Ie = 2e-9  # Input current (A)

# Initialize membrane potential and adaptation conductance arrays
V = np.zeros(n_steps)
V[0] = V_rest  # Initial condition
g_sra = np.zeros(n_steps)  # Spike-rate adaptation conductance

time = np.arange(0, total_time, dt)

# Simulation loop
for i in range(1, n_steps):
    dV = (V_rest - V[i-1] - Rm * g_sra[i-1] * (V[i-1] - V_K) + Rm * Ie) * (dt / tau_m)
    dg_sra = (-g_sra[i-1]) * (dt / tau_sra)

    V[i] = V[i-1] + dV
    g_sra[i] = g_sra[i-1] + dg_sra

    # If threshold is reached, reset potential and increase g_sra
    if V[i] >= V_th:
        V[i] = V_rest
        g_sra[i] += delta_g_sra

# Plot the results
plt.figure(figsize=(10, 5))
plt.plot(time, V * 1e3, label="Membrane Potential (mV)")  # Convert V to mV for plotting
plt.xlabel("Time (s)")
plt.ylabel("Membrane Potential (mV)")
plt.title("Integrate-and-Fire Model with Spike-Rate Adaptation")
plt.grid()

# Add legend
plt.legend()

# Show the plot
plt.show()
```



Your answer here

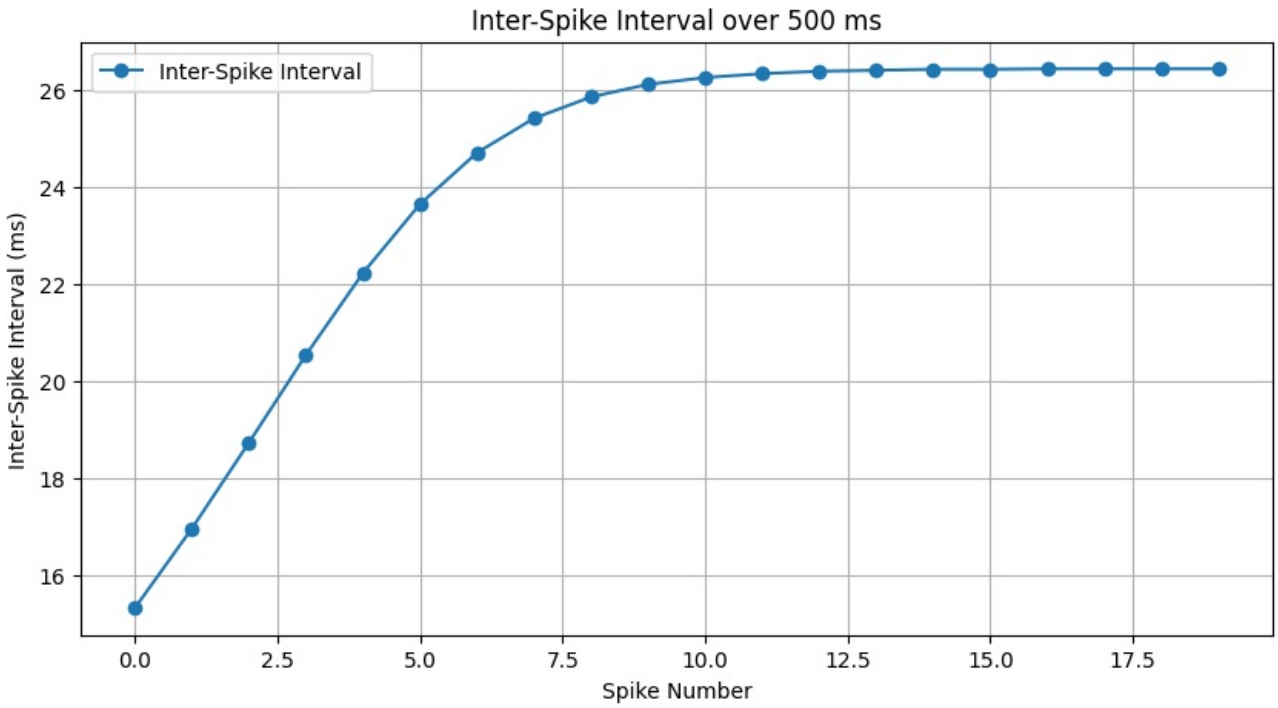
4b

Plot the inter-spike interval (the time between the spikes) of all the spikes that occur in 500 ms. (2 pts)

In [82]:

```
# Your code here

import numpy as np
import matplotlib.pyplot as plt

# Given parameters
tau_m = 10e-3  # Membrane time constant (s)
V_rest = -65e-3  # Resting membrane potential (V)
V_th = -50e-3  # Threshold potential (V)
Rm = 1e7  # Membrane resistance (ohms)
V_K = -70e-3  # Potassium equilibrium potential (V)
tau_sra = 100e-3  # Spike-rate adaptation time constant (s)
delta_g_sra = 0.06 / Rm  # Increase in g_sra per spike (S)

dt = 10e-6  # Time step (s)
total_time = 0.5  # Total simulation time (s)
n_steps = int(total_time / dt)  # Number of time steps

Ie = 2e-9  # Input current (A)

# Initialize membrane potential and adaptation conductance arrays
V = np.zeros(n_steps)
V[0] = V_rest  # Initial condition
g_sra = np.zeros(n_steps)  # Spike-rate adaptation conductance

time = np.arange(0, total_time, dt)
spike_times = []

# Simulation loop
for i in range(1, n_steps):
    dV = (V_rest - V[i-1] - Rm * g_sra[i-1] * (V[i-1] - V_K) + Rm * Ie) * (dt / tau_m)
    dg_sra = (-g_sra[i-1]) * (dt / tau_sra)

    V[i] = V[i-1] + dV
    g_sra[i] = g_sra[i-1] + dg_sra

    # If threshold is reached, reset potential and increase g_sra
    if V[i] >= V_th:
        V[i] = V_rest
        g_sra[i] += delta_g_sra
        spike_times.append(time[i])

# Compute inter-spike intervals
isi = np.diff(spike_times)  # Time differences between consecutive spikes

# Plot the inter-spike interval
plt.figure(figsize=(10, 5))
plt.plot(range(len(isi)), isi * 1e3, marker='o', linestyle='-', label="Inter-Spike Interval")  # Convert s to m
```

```
plt.xlabel("Spike Number")
plt.ylabel("Inter-Spike Interval (ms)")
plt.title("Inter-Spike Interval over 500 ms")
plt.grid()
plt.legend()
plt.show()
```



Your answer here

4c

Explain how the spike-rate adaptation term we introduced above might be contributing to the behavior you observe in 2.4.b. (2 pts)

The spike-rate adaptation (SRA) term introduces a dynamic conductance that increases every time an action potential occurs, effectively acting as a temporary inhibitory force. This added conductance counteracts the depolarizing input current, making it harder for the neuron to reach the threshold for subsequent spikes.

As a result, **the inter-spike intervals (ISIs) gradually increase over time**, meaning the neuron fires rapidly at first but slows down as the adaptation accumulates. This mimics real neuronal behavior, where neurons often display a reduction in firing rate during sustained input due to adaptation mechanisms such as potassium channel activation.

In summary, the SRA term is responsible for **spike frequency adaptation**, causing the neuron to fire less frequently over time when exposed to a constant input current. This is evident in the ISI plot, where the time between spikes progressively increases.

Your answer here

5

Pursue an extension of this basic integrate and fire model. A few ideas are: implement the Integrate-and-Fire-or-Burst Model of Smith et al. 2000 (included); implement the Hodgkin-Huxley model (see Dayan and Abbot, pg. 173); provide some sort of interesting model of a population of neurons; or perhaps model what an electrode sampling at 200 Hz would record from the signal you produce in question 2.3. Feel free to be creative. We reserve the right to give extra credit to particularly interesting extensions and will in general be more generous with points for more difficult extensions (like the first two ideas), though it is possible to get full credit for any well-done extension.

5a

Briefly describe what your extension is and how you will execute it in code. (6 pts)

The Hodgkin-Huxley (HH) model is a detailed biophysical model of neuronal action potentials. Unlike the basic integrate-and-fire model, the HH model explicitly accounts for the dynamics of ion channels, specifically sodium (Na^+), potassium (K^+), and a leak current, which drive the neuron's voltage changes. This model is based on differential equations governing the membrane potential and the gating variables that control ion conductance. Execution in Code

```
Define the Membrane Potential Equation
    The change in membrane potential VV is governed by:
    CmdVdt=I−(gKn4(V−EK)+gNam3h(V−ENa)+gL(V−EL))
```

```
        CmdtdV=I−(gKn4(V−EK)+gNam3h(V−ENa)+gL(V−EL))
        Where:
            CmCm is the membrane capacitance.
            gK,gNa,gLgK,gNa,gL are the conductances for potassium, sodium, and leak currents.
            EK,ENa,ELEK,ENa,EL are the reversal potentials for each ion.

    Model the Gating Variables (m, h, n)
        Each gating variable follows a differential equation:
        dxdt=αx(1−x)−βxx
        dtdx=αx(1−x)−βxx
        Where xx represents m, h, or n, and αα, ββ are voltage-dependent rate functions.

    Numerical Integration (Euler Method or Runge-Kutta)
        Implement a time-stepping loop to solve the system using Euler's method.
        Use a small time step (e.g., 0.01 ms) for accuracy.

    Simulation and Visualization
        Apply an external current II to the neuron.
        Simulate the evolution of membrane potential over time.
        Plot the voltage trace to observe action potentials.
```

Your answer here

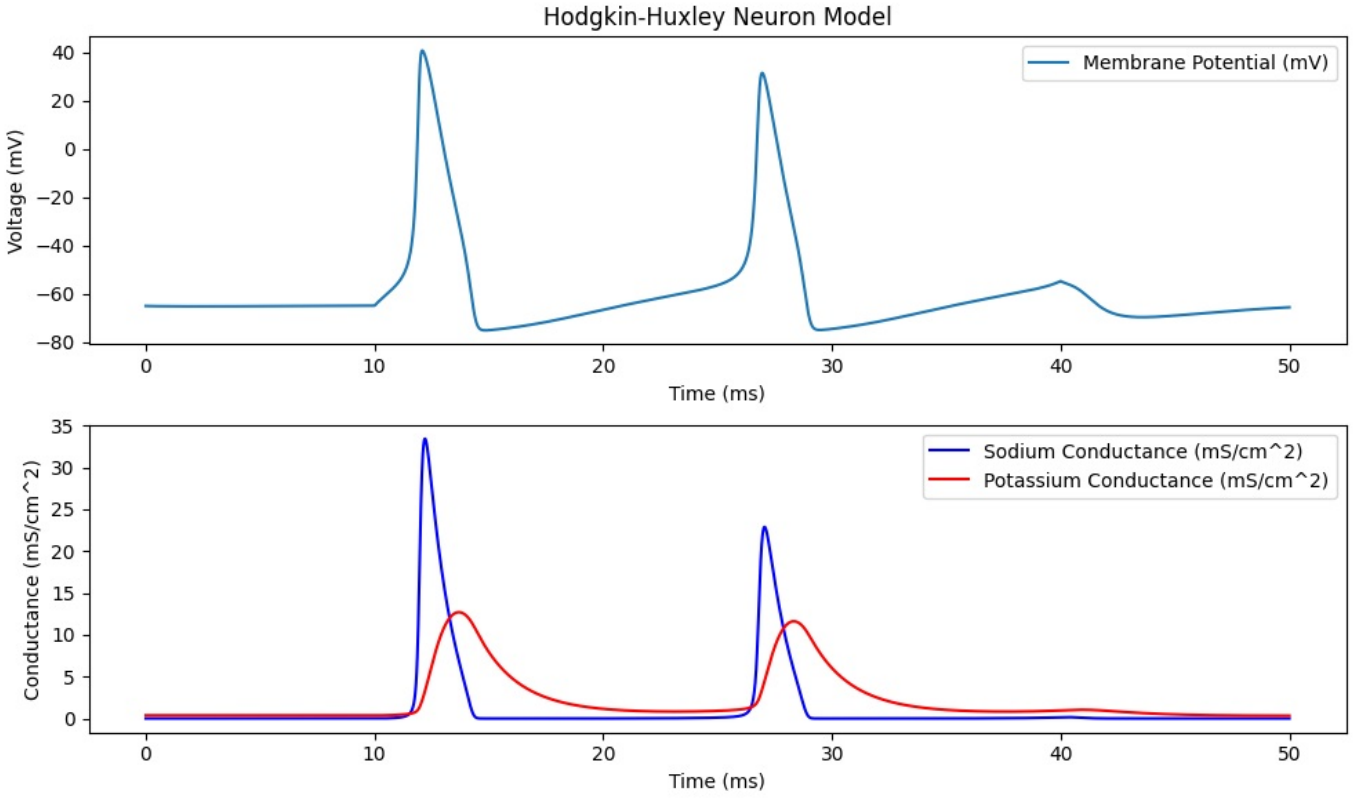
5b

Provide an interesting figure along with an explanation illustrating the extension. (4 pts)

In [83]:

```python
import numpy as np
import matplotlib.pyplot as plt

# Constants
Cm = 1.0  # Membrane capacitance, uF/cm^2
EK = -77   # Potassium reversal potential, mV
ENa = 50   # Sodium reversal potential, mV
EL = -54.4 # Leak reversal potential, mV
gK = 36    # Maximum conductance for K, mS/cm^2
gNa = 120  # Maximum conductance for Na, mS/cm^2
gL = 0.3   # Leak conductance, mS/cm^2

# Time parameters
dt = 0.01   # Time step, ms
tmax = 50   # Total time, ms
t = np.arange(0, tmax, dt)

# External current
I = np.zeros_like(t)
I[1000:4000] = 10  # Applied current from 10 ms to 40 ms

# Initial conditions
V = -65
m, h, n = 0.05, 0.6, 0.32  # Initial gating variables

# Functions for alpha and beta rates
def alpha_m(V): return 0.1 * (V + 40) / (1 - np.exp(-(V + 40) / 10))
def beta_m(V): return 4.0 * np.exp(-0.0556 * (V + 65))
def alpha_h(V): return 0.07 * np.exp(-0.05 * (V + 65))
def beta_h(V): return 1 / (1 + np.exp(-(V + 35) / 10))
def alpha_n(V): return 0.01 * (V + 55) / (1 - np.exp(-(V + 55) / 10))
def beta_n(V): return 0.125 * np.exp(-(V + 65) / 80)

# Initialize variables
V_trace = []
gNa_trace = []
gK_trace = []

# Simulation loop
for i in range(len(t)):
    # Compute conductances
    gNa_t = gNa * (m**3) * h
    gK_t = gK * (n**4)
    gL_t = gL

    # Compute currents
    INa = gNa_t * (V - ENa)
    IK = gK_t * (V - EK)
    IL = gL_t * (V - EL)

    # Update membrane potential
    dVdt = (I[i] - (INa + IK + IL)) / Cm
    V += dVdt * dt
```

```
    # Update gating variables
    m += (alpha_m(V) * (1 - m) - beta_m(V) * m) * dt
    h += (alpha_h(V) * (1 - h) - beta_h(V) * h) * dt
    n += (alpha_n(V) * (1 - n) - beta_n(V) * n) * dt

    # Store results
    V_trace.append(V)
    gNa_trace.append(gNa_t)
    gK_trace.append(gK_t)

# Plot results
plt.figure(figsize=(10, 6))
plt.subplot(2, 1, 1)
plt.plot(t, V_trace, label='Membrane Potential (mV)')
plt.xlabel('Time (ms)')
plt.ylabel('Voltage (mV)')
plt.title('Hodgkin-Huxley Neuron Model')
plt.legend()

plt.subplot(2, 1, 2)
plt.plot(t, gNa_trace, label='Sodium Conductance (mS/cm^2)', color='b')
plt.plot(t, gK_trace, label='Potassium Conductance (mS/cm^2)', color='r')
plt.xlabel('Time (ms)')
plt.ylabel('Conductance (mS/cm^2)')
plt.legend()
plt.tight_layout()
plt.show()
```



This extension of the Hodgkin-Huxley model enhances the basic implementation by explicitly tracking and visualizing the sodium ((g_{Na})) and potassium ((g_K)) ion conductances over time. The model simulates the neuron's membrane potential by solving differential equations that govern ion channel dynamics, using gating variables ((m, h, n)) to control the opening and closing of sodium and potassium channels. The inclusion of ion conductance tracking provides insight into how these channels influence action potential generation and propagation. The simulation applies an external current, leading to characteristic action potentials, and the plotted conductances reveal the dynamic shifts in sodium and potassium permeability that underlie each spike. This addition helps in better understanding the biophysical properties of neuronal activity beyond just voltage changes.

Your answer here

In [83]: