Assignment 1 - Neural Models

Jihun Joo Haolun Cheng Jonathan Li Michael Ziegler Ray Sy
Joseph Shamma

October 2020

1 Contributions

- **Ray Sy:** Worked on creating Hodgkin-Huxley model for question 3.5; performed proofreading/debugging/ general code cleanup for other models
- **Michael Ziegler:** Completed Hodgkin-Huxley model for a simulated neuron for question 3.5; worked on final report/writeups
- **Joseph Shammy:** Completed responses for question 2.
- **Haolun Cheng:** Assisted in responses for question 2 and worked on completing LIF neuron simulation in question 3.1.
- **Jihun Joo:** Worked on questions 3.1; worked on completing questions 3.2-3.4 including the Izhikevich model
- **Jonathan Li:** Worked on questions 3.1; worked on completing questions 3.2-3.4 including the Izhikevich model

2 Questions

2.
   1 Since IF neuron models do not leak out the inputs over time, even a very low input current can accumulate voltage over a very long time to reach the threshold and spike. The LIF model would not spike because it integrates a leak which prevents the low input current from accumulating and reaching the threshold.

   2 When fed with larger input currents, both an IF or LIF neuron would spike when reaching threshold, then reset to its resting potential, continuing this behavior until the current ends. Depending on the input amplitude, it is possible that the IF model would spike sooner than the LIF model because there is no leak. With a large enough input current, it's also possible that the membrane potential of the neuron will reach its asymptotic value with increasing t. In the case of an LIF neuron, this means that all current will flow through the resistor.

   3 Neurons utilizing the LIF model must reset voltage after each spike, meaning they lack a capacity for adaptation. A neuron will also suffer conductance changes during and after a spike, meaning a spike recorded soon after another may yield a diminished change in the membrane potential. Finally, LIF neurons suffer from shunting inhibition, essentially meaning that an inhibitory synapse recorded from a neuron is likely to decrease the amplitude of subsequent spikes in membrane potential.

3.

LIF

October 28, 2020

[1]:
```
import numpy as np
import matplotlib.pyplot as plt

plt.style.use('ggplot')
```

0.0.1 Question 3.1

Simulate an LIF neuron with different input currents and plot the membrane potential, showing (a) potential decay over time and (b) spiking behavior.

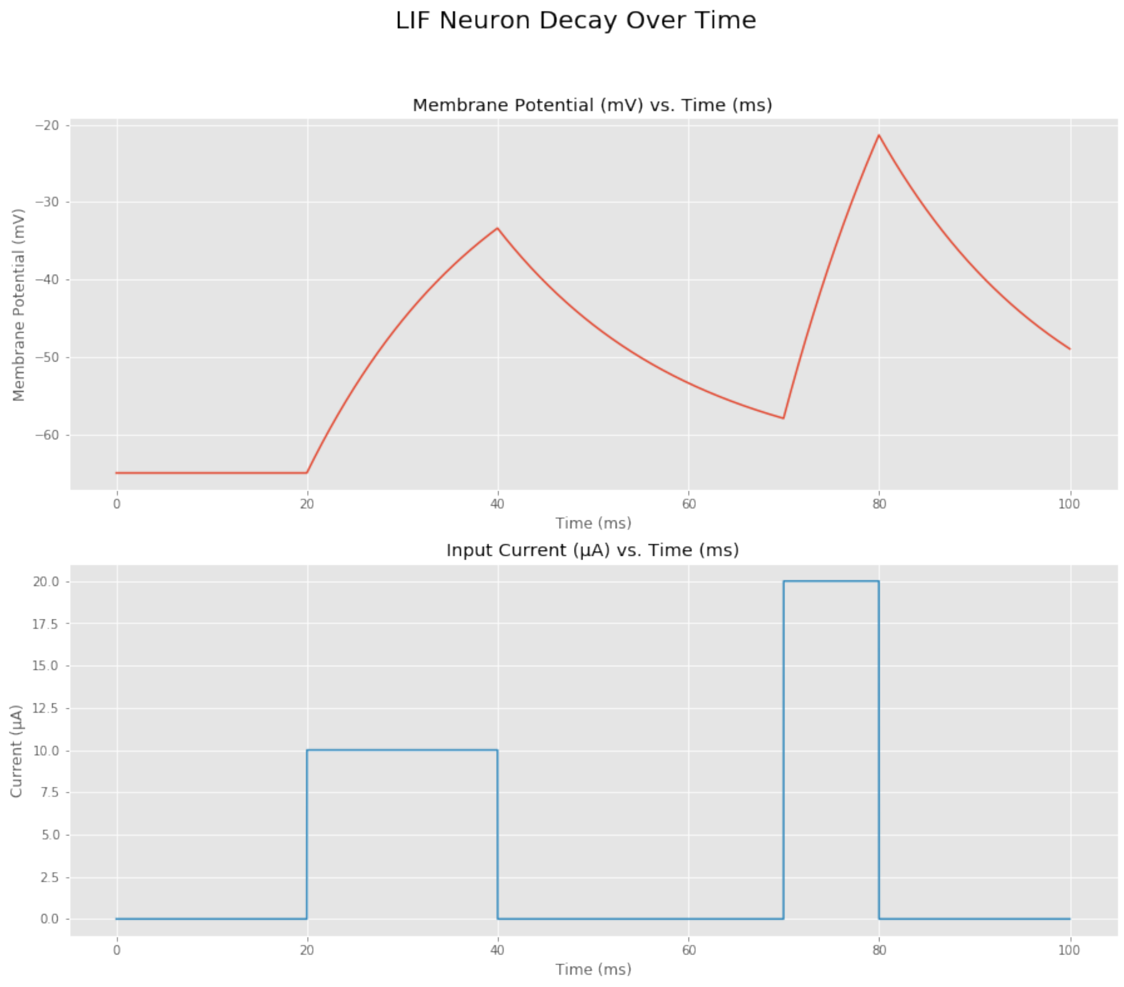
Model

[2]:
```
def LIF(I, Cm, Rm):
    """
    Runs a LIF simulation on neuron and returns outputted voltage

            Parameters:
                    I (double[]): A list of input voltages in mV
                    Cm (double): The membrane capacitance
                    Rm (double): The membrane resistance

            Returns:
                    V (double[]): A list of the output voltages in mV
    """
    V_thresh = 30
    V_rest = -65
    V_spike = 80
    dT = 0.02  # time step in ms
    total_time = (I.size) * dT

    # an array of time
    time = np.arange(0, total_time, dT)

    # default voltage list set to resting volatage of -65mV
    V = (-65) * np.ones(len(time))

    did_spike = False
    # function member variable to track spikes
    LIF.spikes = 0

    for t in range(len(time)):
        # using "I - V(t)/Rm = Cm * dV/dT"
        dV = (I[t] - (V[t - 1] - V_rest) / Rm) / Cm

        # reset membrane potential if neuron spiked last tick
        if did_spike:
            V[t] = V_rest + dV * dT
        else:
            V[t] = V[t - 1] + dV * dT

        # check if membrane voltage exceeded threshold (spike)
        if V[t] > V_thresh:
            did_spike = True
            # set the last step to spike value
            V[t] = V_spike
            LIF.spikes += 1
        else:
            did_spike = False

    return V
```

Potential decay over time

[3]:
```
total_time = 100
dT = 0.02
time = np.arange(0, total_time + dT, dT)

# input current initially at 0μA
I = np.zeros(len(time))

# 10μA from 20ms - 40ms
I[1000:2000] = 10

# 20μA from 70ms - 80ms
I[3500:4000] = 20

# output voltage from LIF model
V = LIF(I=I, Cm=4, Rm=5)

fig, (ax1, ax2) = plt.subplots(2, figsize=(15, 12))
fig.suptitle("LIF Neuron Decay Over Time", fontsize=20)

ax1.plot(time, V, label="Membrane Potential")
ax1.set_title("Membrane Potential (mV) vs. Time (ms)")
```

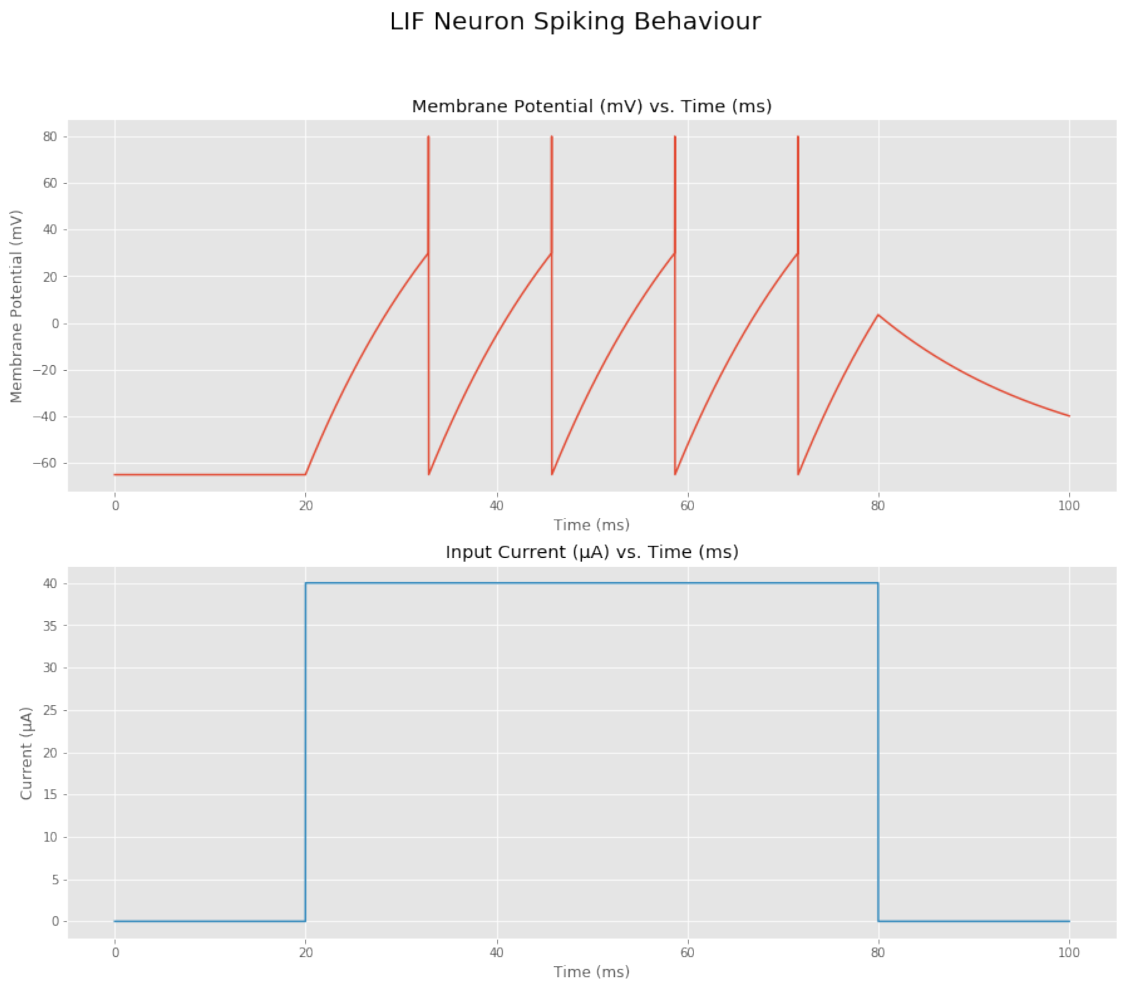
```
ax1.set_ylabel("Membrane Potential (mV)")
ax1.set_xlabel("Time (ms)")

ax2.plot(time, I, "C1", label="Input Current")
ax2.set_title("Input Current (μA) vs. Time (ms)")
ax2.set_ylabel("Current (μA)")
ax2.set_xlabel("Time (ms)")

plt.show()
```



Spiking behavior

[4]:
```
# input current array
total_time = 100
dT = 0.02
time = np.arange(0, total_time + dT, dT)
```

```
# input current initially at 0µA
I = np.zeros(len(time))

# 40µA from 20ms - 80ms
I[1000:4000] = 40

# output voltage from LIF model
V = LIF(I=I, Cm=4, Rm=5)

fig, (ax1, ax2) = plt.subplots(2, figsize=(15, 12))
fig.suptitle("LIF Neuron Spiking Behaviour", fontsize=20)

ax1.plot(time, V, label="Membrane Potential")
ax1.set_title("Membrane Potential (mV) vs. Time (ms)")
ax1.set_ylabel("Membrane Potential (mV)")
ax1.set_xlabel("Time (ms)")

ax2.plot(time, I, "C1", label="Input Current")
ax2.set_title("Input Current (µA) vs. Time (ms)")
ax2.set_ylabel("Current (µA)")
ax2.set_xlabel("Time (ms)")

plt.show()
```



Abstract

Given a list of input voltages in mV (I), the membrane capacitance (Cm), and the membrane resistance (Rm), our LIF model returns a list of the output voltages in mV (V). We assumed that the threshold voltage is 30 mV, resting voltage is -65 mV. We then calculate the total time we run the experiment by the size of I and the change in time, dT, which we assumed to be 0.02 ms. Using the LIF equation "I - V(t)/Rm = Cm * dV/dT", we derive the dV, which is the change in voltage. Then we consider both the change in voltage and time to continuously integrate the inputs until it crosses the threshold. If this happens, the voltage is saved as the spike voltage, which we assumed to be 80 mV, and the next addition to the voltage starts from the resting voltage. In other words, it resets after a spike. We continue doing this until we reach the time determined in the beginning.

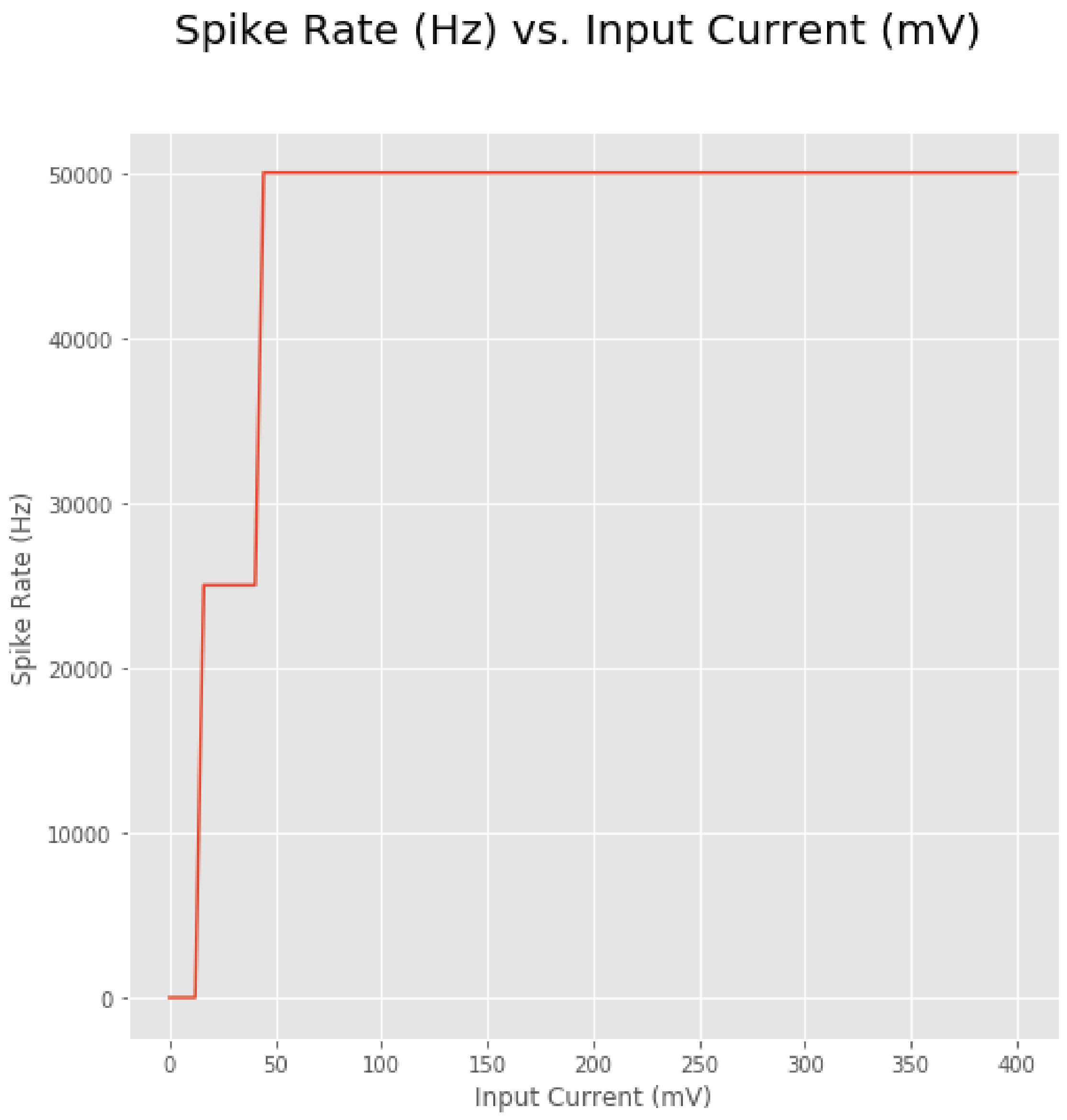
0.0.2 Question 3.2

Plot the firing rate as a function of the input current.

[5]:
```
# total time (ms)
T = 100
```

```
# step interval (ms)
dT = 0.02

I_max = 400
I_num = 100

# 20ms
T_input_start = 20 * 50
# 80ms
T_input_end = 80 * 50

# input current increases from 0mA to 400mA in 4mA intervals
input_current = np.linspace(0, I_max, I_num)

spike_rate = np.empty(input_current.size)


for i in range(input_current.size):
    """calculate spike rate for each iteration"""
    time = np.arange(0, T + dT, dT)

    # from 20ms to 80ms the input voltage spikes to input_current[i]
    I = np.zeros(len(time))
    I[T_input_start:T_input_end] = input_current[i]

    # run LIF simulation
    LIF(I=I, Cm=0.003, Rm=5)

    # calculate the spike rate during the 20ms to 80ms period
    spike_rate[i] = LIF.spikes / (time[T_input_end] - time[T_input_start]) * 1000

fig = plt.figure(figsize=(8, 8))
fig.subplots_adjust(hspace=0.4, wspace=0.4)
fig.suptitle("Spike Rate (Hz) vs. Input Current (mV)", fontsize=20)

plt.plot(input_current, spike_rate)
plt.xlabel("Input Current (mV)")
plt.ylabel("Spike Rate (Hz)")
```

```
[5]: Text(0, 0.5, 'Spike Rate (Hz)')
```



0.0.3 Question 3.3

Question: What happens to the firing rate as you continue to increase the input current? Why?

Answer: Generally speaking, input current and firing rate increase proportionally since the charge is able to reach the spiking threshold faster. At a certain point the input current will become so powerful that it fills the threshold immediately causing a spike. The cell then drops back to its resting potential after which it immediately spikes again. Therefore, the firing rate will be capped at a certain value - in our case 50000Hz, for any input greater or equal to approximately 58mV.

Izhikevich

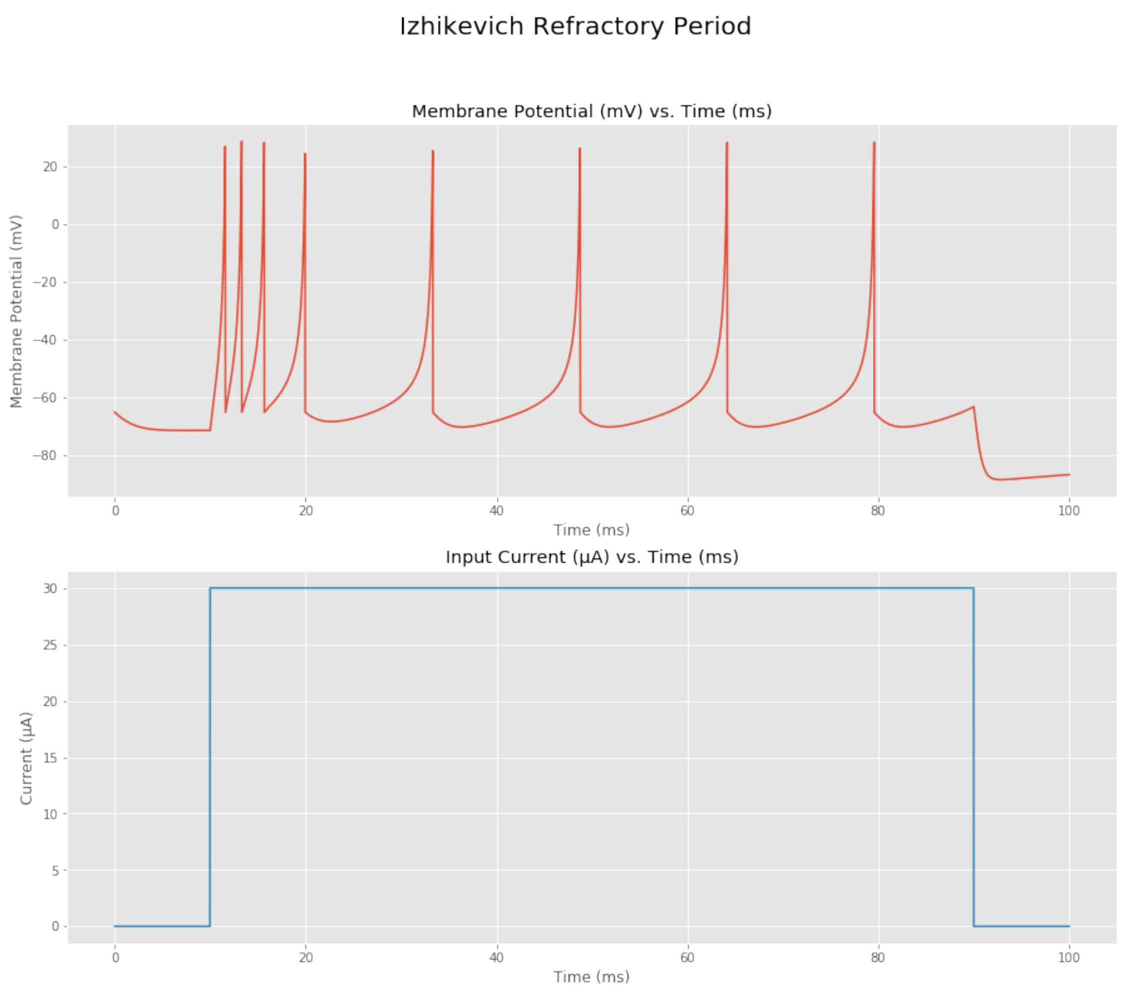
October 28, 2020

[1]:
```python
import numpy as np
import matplotlib.pyplot as plt

plt.style.use('ggplot')
```

0.0.1 Question 3.4

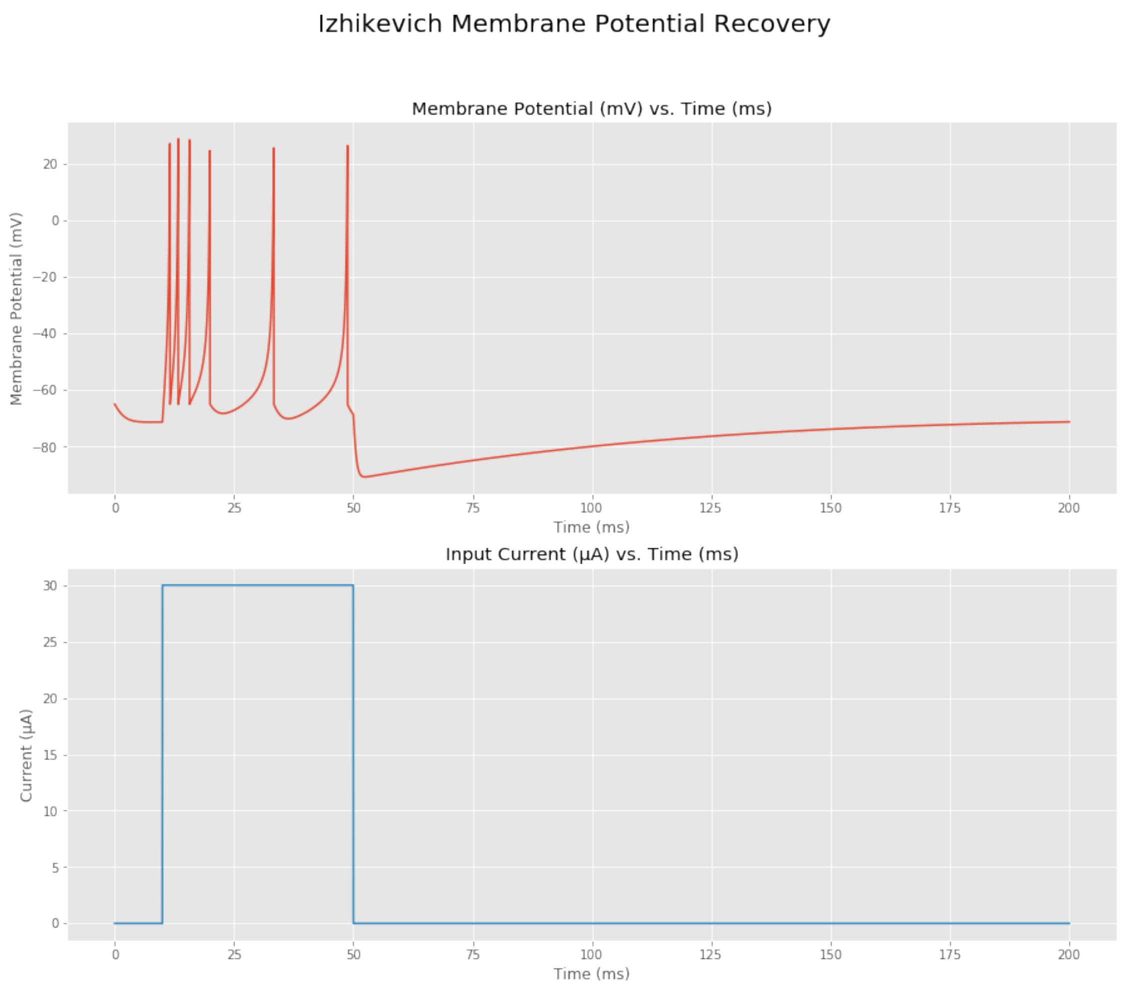
[2]:
```python
def Izhikevich(I, a, b, c, d):
    """
    Runs a Izhikevich simulation on neuron and returns outputted voltage

            Parameters:
                    a (double): The time scale of the recovery recovery variable␣
 ↪U
                    b (double): The sensitivity of the recovery variable U
                    c (double): The after spike reset value of the membrane␣
 ↪potential V caused by fast high-threshold K+ conductances
                    d (double): The after spike reset value of the recovery␣
 ↪variable U caused by slow high-threshold Na+ and K+ conductances

            Returns:
                    V (double[]): A list of the output voltages in mV
    """

    V_thresh = 30
    V_rest = -65

    dT = 0.02
    T = (I.size) * dT
    time = np.arange(0, T, dT)

    # cell voltage
    V = (-65) * np.ones(len(time))
    # membrane recovery
    U = np.zeros(len(time))
    U[0] = b * V[0]
    for i in range(0, len(time) - 1):
        # U[current] = slope of U[previous] * time(current-previous)
        U[i + 1] = U[i] + dT * (a * (b * V[i] - U[i]))
        # V[current] = slope of V[previous] * time(current-previous)
        V[i + 1] = V[i] + dT * (0.04 * V[i] ** 2 + 5 * V[i] + 140 - U[i] + I[i])

        # when v reaches 30 mV, the cell fires, and then v is reset to c, and u␣
↪increases by d
        if V[i + 1] > V_thresh:
            V[i + 1] = c
            U[i + 1] = U[i + 1] + d
    return V
```

[3]:
```
T = 100
dT = 0.02
time = np.arange(0, T + dT, dT)
I = np.zeros(len(time))
I[500:4500] = 30

V = Izhikevich(I=I, a=0.02, b=0.2, c=-65, d=8)

fig, (ax1, ax2) = plt.subplots(2, figsize=(15, 12))
fig.suptitle("Izhikevich Refractory Period", fontsize=20)

ax1.plot(time, V, label="Membrane Potential")
ax1.set_title("Membrane Potential (mV) vs. Time (ms)")
ax1.set_ylabel("Membrane Potential (mV)")
ax1.set_xlabel("Time (ms)")

ax2.plot(time, I, "C1", label="Input Current")
ax2.set_title("Input Current (µA) vs. Time (ms)")
ax2.set_ylabel("Current (µA)")
ax2.set_xlabel("Time (ms)")

plt.show()
```



[4]:
```
T = 200
dT = 0.02
time = np.arange(0, T + dT, dT)
I = np.zeros(len(time))
I[500:2500] = 30

V = Izhikevich(I=I, a=0.02, b=0.2, c=-65, d=8)

fig, (ax1, ax2) = plt.subplots(2, figsize=(15, 12))
fig.suptitle("Izhikevich Membrane Potential Recovery", fontsize=20)

ax1.plot(time, V, label="Membrane Potential")
ax1.set_title("Membrane Potential (mV) vs. Time (ms)")
ax1.set_ylabel("Membrane Potential (mV)")
ax1.set_xlabel("Time (ms)")

ax2.plot(time, I, "C1", label="Input Current")
```

```
ax2.set_title("Input Current (µA) vs. Time (ms)")
ax2.set_ylabel("Current (µA)")
ax2.set_xlabel("Time (ms)")

plt.show()
```



Abstract

Our Izhikevich neuron model was written with the threshold being 30mV and resting potential -65mV, as stated in Eugene M Izhikevich's "Simple model of spiking neurons". The membrane potential is graphed with step method as it computes the membrane potential of time i by adding the potential of time i-1 with the slope of time i-1 multiplied by the small interval of time, in our case 0.02 mS. The two equations that the model follows are: $dv/dt = 0.04v^2 + 5v + 140 - u + Idu/dt = a(bv - u)$. The first one being the neuron voltage, the second being the neuron recovery model. The model also has parameters a,b,c,d to adjust the model's membrane recovery rate. In the code, a,b,c, and d are selected according to the "typical values" of the previously mentioned article. A representing the time scale of recovery, B as the sensitivity of recovery, C as the after-spike reset value of membrane potential, and D as the after-spike reset value of recovery.

Hodgkin-Huxley

October 28, 2020

0.0.1 Question 3.5

[1]:
```
import scipy as sp
import pylab as plt
from scipy.integrate import odeint

plt.style.use('ggplot')
```

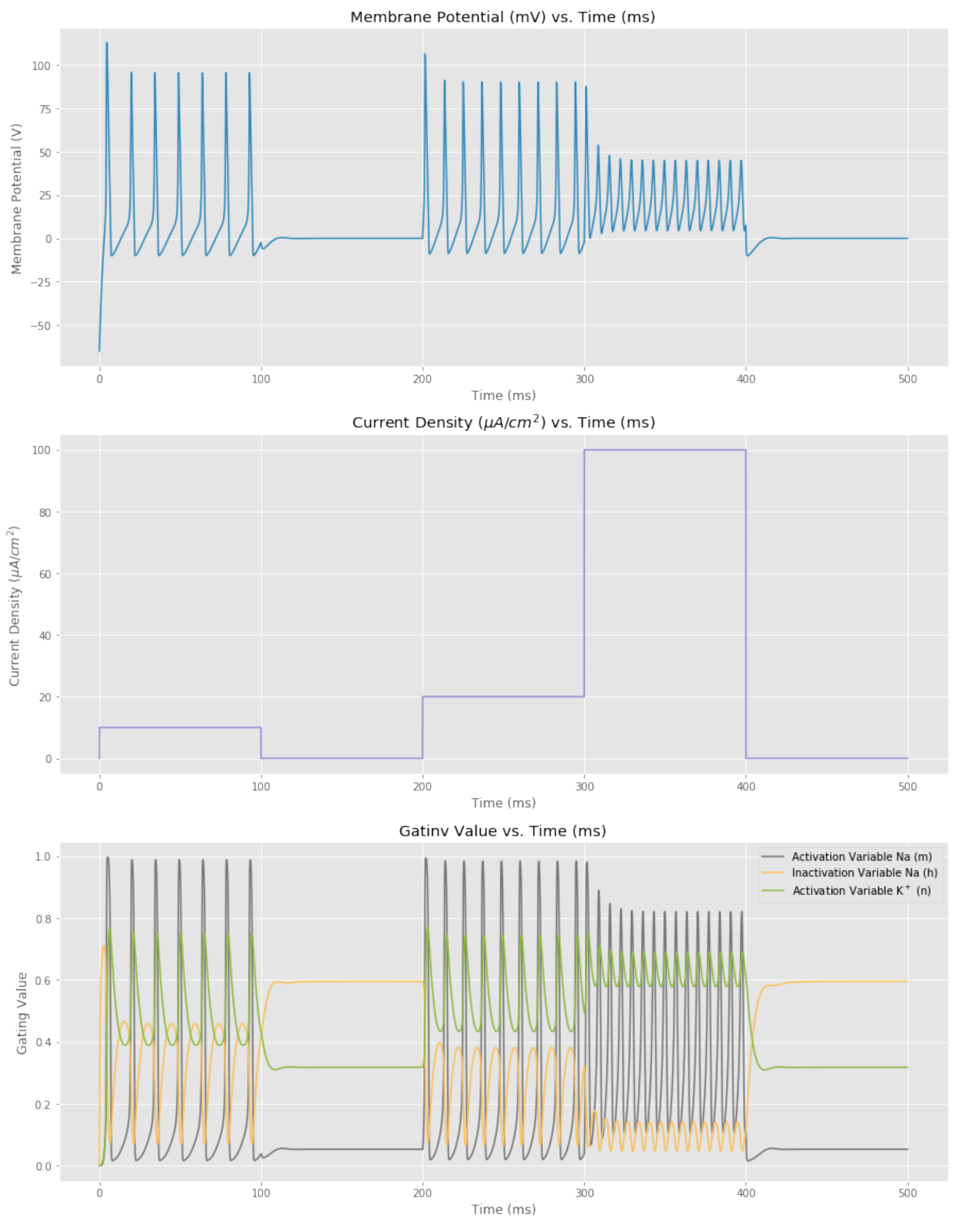
[2]:
```
# full Hodgkin-Huxley model
# membrane capacitance (uF/cm^2)
C_m = 1.0

# constants for Na gate
g_Na = 120.0
V_Na = 115.0

# constants for K gate
g_K = 36.0
V_K = -12.0

# constants for leak gate
g_L = 0.3
V_L = 10.613

# time we'll be integrating over. We'll be operating over 500 ms with a step␣
 ↪size of 0.01.
t = sp.arange(0.0, 500.0, 0.01)

# functions for computing current through membranes
def i_Na(V, m, h):
    return g_Na * m ** 3 * h * (V - V_Na)


def i_K(V, n):
    return g_K * n ** 4 * (V - V_K)
```

```python
def i_L(V):
    return g_L * (V - V_L)


# functions for rate constants of ion channels
def alpha_n(V):
    return 0.01 * (10.0 - V) / (sp.exp((10.0 - V) / 10.0) - 1.0)


def beta_n(V):
    return 0.125 * sp.exp(-V / 80.0)


def alpha_m(V):
    return 0.1 * (25.0 - V) / (sp.exp((25.0 - V) / 10.0) - 1.0)


def beta_m(V):
    return 4.0 * sp.exp(-V / 18.0)


def alpha_h(V):
    return 0.07 * sp.exp(-V / 20.0)


def beta_h(V):
    return 1.0 / (1.0 + sp.exp((30 - V) / 10.0))


# input injection function
def inject_I(t):
    # tuples dictating fluctuations in external current upon injection in given␣
 ↪time frames (basially where spikes will occur) =>
    step1 = {"t": 0, "V": 10}  # step up 10 uA/cm^2 @ t=0
    step2 = {"t": 100, "V": -10}  # step down 10 uA/cm^2 @ t=100
    step3 = {"t": 200, "V": 20}  # step up 20 uA/cm^2 @t=200
    step4 = {"t": 300, "V": 80}  # step up 80 uA/cm^2 @ t=300
    step5 = {"t": 400, "V": -100}  # step down 100 uA/cm^2 @ t=400

    return (
        step1["V"] * (t > step1["t"])
        + step2["V"] * (t > step2["t"])
        + step3["V"] * (t > step3["t"])
        + step4["V"] * (t > step4["t"])
        + step5["V"] * (t > step5["t"])
    )
```

```python
# computes derivatives of activation variables
def compute_derivs(X, t):
    V, m, h, n = X
    dVdt = (inject_I(t) - i_Na(V, m, h) - i_K(V, n) - i_L(V)) / C_m
    dmdt = alpha_m(V) * (1.0 - m) - beta_m(V) * m
    dhdt = alpha_h(V) * (1.0 - h) - beta_h(V) * h
    dndt = alpha_n(V) * (1.0 - n) - beta_n(V) * n
    return dVdt, dmdt, dhdt, dndt
```

[26]:
```python
# plotting Hodgkin-Huxley model simulation
X = odeint(compute_derivs, [-65.0, 0.0, 0.0, 0.0], t)
V = X[:, 0]
m = X[:,1]
h = X[:,2]
n = X[:,3]

plt.figure(figsize=(15, 20))
plt.subplot(3, 1, 1)
plt.title("Membrane Potential (mV) vs. Time (ms)")
plt.plot(t, V, "C1", label="Membrane Potential")
plt.ylabel("Membrane Potential (V)")
plt.xlabel("Time (ms)")

plt.subplot(3, 1, 2)
plt.title("Current Density ($µA/cm^2$) vs. Time (ms)")
plt.plot(t, [inject_I(x) for x in t], "C2", label="Input Current")
plt.ylabel("Current Density ($µA/cm^2$)")
plt.xlabel("Time (ms)")

plt.subplot(3, 1, 3)
plt.title("Gatinv Value vs. Time (ms)")
plt.plot(t, m, "C3", label="Activation Variable Na (m)")
plt.plot(t, h, "C4", label="Inactivation Variable Na (h)")
plt.plot(t, n, "C5", label="Activation Variable K$^+$ (n)")
plt.legend(loc="upper right")
plt.ylabel("Gating Value")
plt.xlabel("Time (ms)")
```

[26]:
```
Text(0.5, 0, 'Time (ms)')
```



```
[ ]:
```

Abstract

Our Hodgkin-Huxley model is intended to simulate a neuron's spiking behavior upon in-

jection of variable amounts of input current density. Using the set of differential equations provided by Hodgkin and Huxley in their 1952 paper, our model takes given input values for voltage V and activation variables m, n, and h, and continuously computes their derivatives for each step in our time frame, with the ultimate goal of computing dV/dt and integrating to find values for voltage. Using a hardcoded input current injection function, we coordinated the following fluctuations in current density with the goal of observing subsequent effects in membrane potential:

- Increase current density by 10 A/cm^2 at t=0 ms
- Decrease current density by 10 A/cm^2 at t=100 ms
- Increase current density by 20 A/cm^2 at t=200 ms
- Increase current density by 80 A/cm^2 at t=300 ms
- Decrease current density by 100 A/cm^2 at t=400 ms

The provided plots exhibit the expected spiking behavior for this neuron upon being exposed to the given amounts of input current in the specified time frames. The neuron produces spikes in membrane potential upon continuous exposure to the current, with a higher frequency of spikes with higher amounts of current.