Question 1. Simulation of a multi-compartment Hodgkin-Huxley model.

1.1 The Hodgkin-Huxley model for an active neurite with input current $I_e(j,t)$ for a single cylindrical compartment j is given by equations 1-12.

$$V' = \frac{V}{mV} \tag{1}$$

$$I_e(t) = C_m \frac{\mathrm{d}V'_j}{\mathrm{d}t} + g_L\left(V'_j - E_L\right) + g_{\mathrm{Na},j}\left(V'_j - E_{\mathrm{Na}}\right) + g_{\mathrm{K},j}\left(V'_j - E_{\mathrm{K}}\right) + g_{\mathrm{ax}}\left(V'_j - V'_{j-1}\right) + g_{\mathrm{ax}}\left(V'_j - V'_{j+1}\right) \tag{2}$$

$$g_{\mathrm{Na},j} = \bar{g}_{\mathrm{Na}} m_j^3 h_j, \quad g_{\mathrm{K},j} = \bar{g}_{\mathrm{K}} n_j^4 \tag{3}$$

Sodium channels (not writing indices j):

$$\frac{\mathrm{d}m}{\mathrm{d}t} = \alpha_m (1-m) - \beta_m m \tag{4}$$

$$\alpha_m = \begin{cases} 0.1 \frac{V'-25}{1-\exp\left(-\frac{V'-25}{10}\right)} & V \neq 25mV \\ 1 & V = 25mV \end{cases} \tag{5}$$

$$\beta_m = 4\exp\left(-\frac{V'}{18}\right) \tag{6}$$

$$\frac{\mathrm{d}h}{\mathrm{d}t} = \alpha_h (1-h) - \beta_h h \tag{7}$$

$$\alpha_h = 0.07\exp\left(-\frac{V'}{20}\right) \tag{8}$$

$$\beta_h = \frac{1}{1+\exp\left(-\frac{V'-30}{10}\right)} \tag{9}$$

Potassium channel (not writing indices j):

$$\frac{\mathrm{d}n}{\mathrm{d}t} = \alpha_n (1-n) - \beta_n n \tag{10}$$

$$\alpha_n = \begin{cases} 0.01 \frac{V'-10}{1-\exp\left(-\frac{V'-10}{10}\right)} & V \neq 10mV \\ 0.1 & V = 10mV \end{cases} \tag{11}$$

$$\beta_n = 0.125\exp\left(-\frac{V'}{80}\right) \tag{12}$$

The electrical and geometrical properties are

- $C_m = 1\,\mu\mathrm{F}$, membrane capacitance
- $E_{Na} = 115\,\mathrm{mV}$, sodium equilibrium potential
- $E_K = -12\,\mathrm{mV}$, potassium equilibrium potential
- $E_L = 10.6\,\mathrm{mV}$, leak equilibrium potential
- $V(t=0) = 0\,\mathrm{mV}$, initial (and equilibrium) membrane potential
- $\bar{g}_{Na} = 120\,\mathrm{mS}$, maximum sodium channel conductance
- $\bar{g}_K = 36\,\mathrm{mS}$, maximum potassium channel conductance

- $g_L = 0.3\,\mathrm{mS}$, leak conductance
- $g_{ax} = 0.5\,\mathrm{mS}$, axial conductance
- $N = 100$, number of compartments

Given the Hodgkin-Huxley model described above, we approximate the DEQs using the Forward Euler method and derive an equation for $V(j,t)$ for an arbitrary compartment j. We assume that the first compartment is terminated as a 'sealed end' and the last compartment is terminated as a 'killed end', as shown in Figure 1.

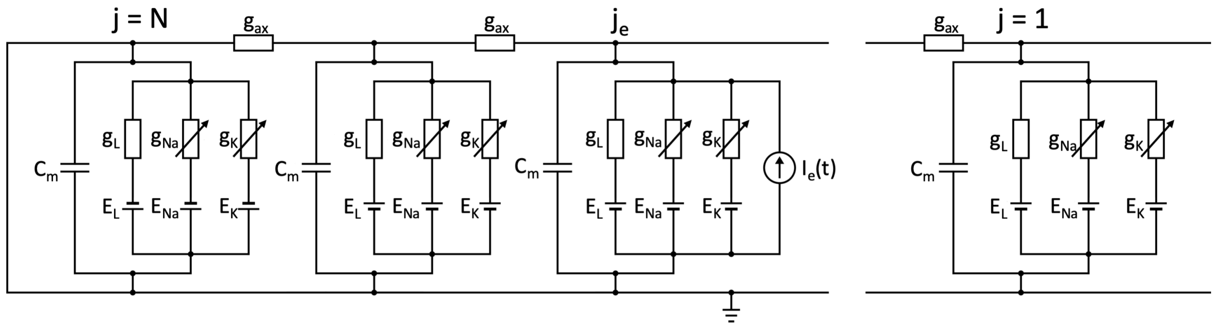

Figure 1: Circuit diagram of a multi-compartment Hodgkin-Huxley model. Compartment 1 is terminated as a 'sealed end' and compartment N is terminated as a 'killed end'. Note that the connection between compartment 1 and compartment j_e is not shown for simplicity.

The approximated DEQ for voltage response of compartment j into which current is injected looks as shown below

$$\begin{aligned}V_j'(t+\Delta t) = V_j'(t) + \frac{\Delta t}{C_m}\Big(I_e(t) &+ g_L\left(E_L - V_j'(t)\right) + \bar{g}_{Na} m_j^3(t) h_j(t)\left(E_{Na} - V_j'(t)\right)\\ &+ \bar{g}_K n_j^4(t)\left(E_K - V_j'(t)\right) + g_{ax}\left(V_{j-1}'(t) - V_j'(t)\right)\\ &+ g_{ax}\left(V_{j+1}'(t) - V_j'(t)\right)\Big)\end{aligned}$$

Following Homework 4, voltage response for compartment i where $0 < i < j$ is

$$\begin{aligned}V_i'(t+\Delta t) = V_i'(t) + \frac{\Delta t}{C_m}\Big(g_L\left(E_L - V_i'(t)\right) &+ \bar{g}_{Na} m_i^3(t) h_i(t)\left(E_{Na} - V_i'(t)\right)\\ &+ \bar{g}_K n_i^4(t)\left(E_K - V_i'(t)\right) + g_{ax}\left(V_{i-1}'(t) - V_i'(t)\right)\\ &+ g_{ax}\left(V_{i+1}'(t) - V_i'(t)\right)\Big)\end{aligned}$$

As for compartment i where $j < i < N$, we have

$$\begin{aligned}V_i'(t+\Delta t) = V_i'(t) + \frac{\Delta t}{C_m}\Big(g_L\left(E_L - V_i'(t)\right) &+ \bar{g}_{Na} m_i^3(t) h_i(t)\left(E_{Na} - V_i'(t)\right)\\ &+ \bar{g}_K n_i^4(t)\left(E_K - V_i'(t)\right) + g_{ax}\left(V_{i+1}'(t) - V_i'(t)\right)\\ &+ g_{ax}\left(V_{i-1}'(t) - V_i'(t)\right)\Big)\end{aligned}$$

As regards the termini, membrane potential for compartment $j = N$ is set to 0 and for compartment $j = 1$ we assume no axial current flowing out, and hence all membrane potential difference is due to current flowing through the membrane

$$V_1'(t + \Delta t) = V_1'(t) + \frac{\Delta t}{C_m} \Big(+ g_L \left(E_L - V_1'(t)\right) + \bar{g}_{Na} m_1^3(t) h_1(t) \left(E_{Na} - V_1'(t)\right) + \bar{g}_K n_1^4(t) \left(E_K - V_1'(t)\right) + g_{ax} \left(V_2'(t) - V_1'(t)\right)\Big)$$

For the approximations of m, h, n gating variables please refer to Homework 4.

1.2 Now, we turn on simulating the above model for an input current $I_e(j, t)$

$$I_e(j,t) = \begin{cases} 0, & (t < t_e) \vee (t_s \leqslant t) \vee (j \neq j_e) \\ I_0, & (t_e \leqslant t < t_s) \wedge (j = j_e) \end{cases}$$

with $j_e = 14$, $t_e = 60\,\text{ms}$, $t_s = 260\,\text{ms}$ and different amplitudes $I_0 = 6\,\mu\text{A}$, $I_0 = 8\,\mu\text{A}$, $I_0 = 15\,\mu\text{A}$, and $I_0 = 20\,\mu\text{A}$. The results of this simulation appear in Figures 2, 3, 4, 5.

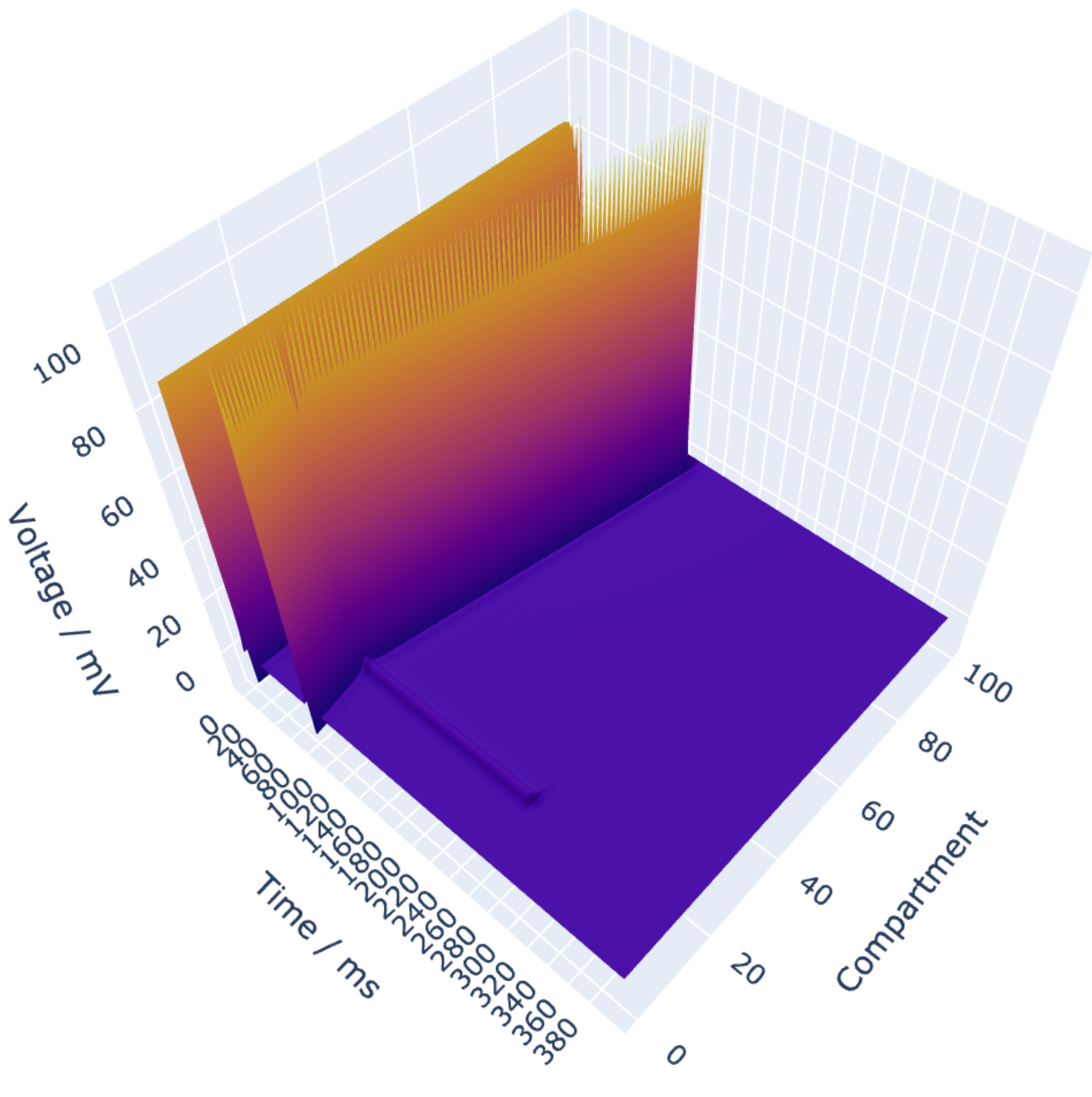

Figure 2: Voltage response to $I_e(14, t)$ of $6\,\mu$A amplitude.

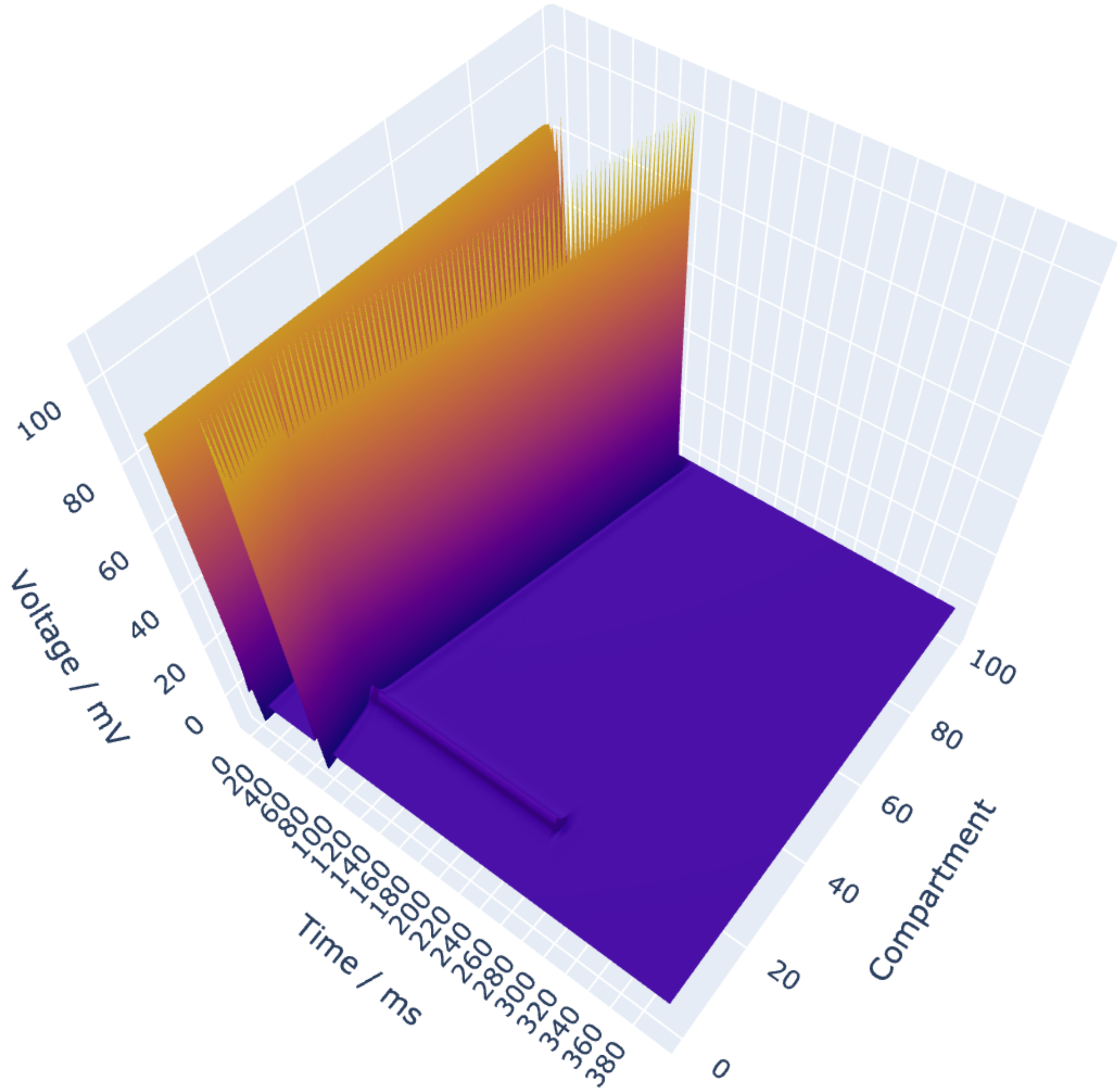

Figure 3: Voltage response to $I_e(14, t)$ of $8\,\mu$A amplitude.

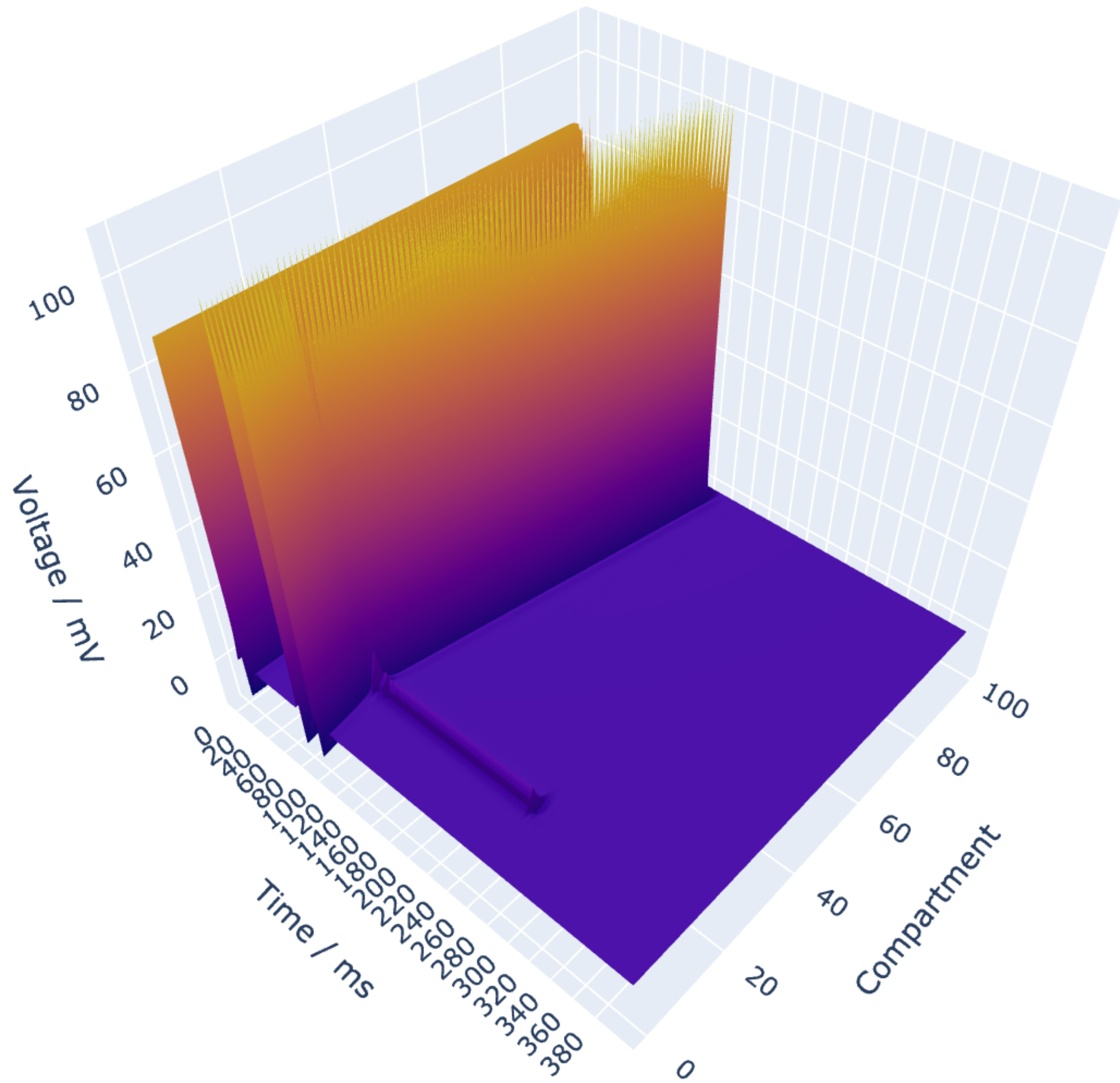

Figure 4: Voltage response to $I_e(14, t)$ of $15\,\mu$A amplitude. Note the second wave of spikes.

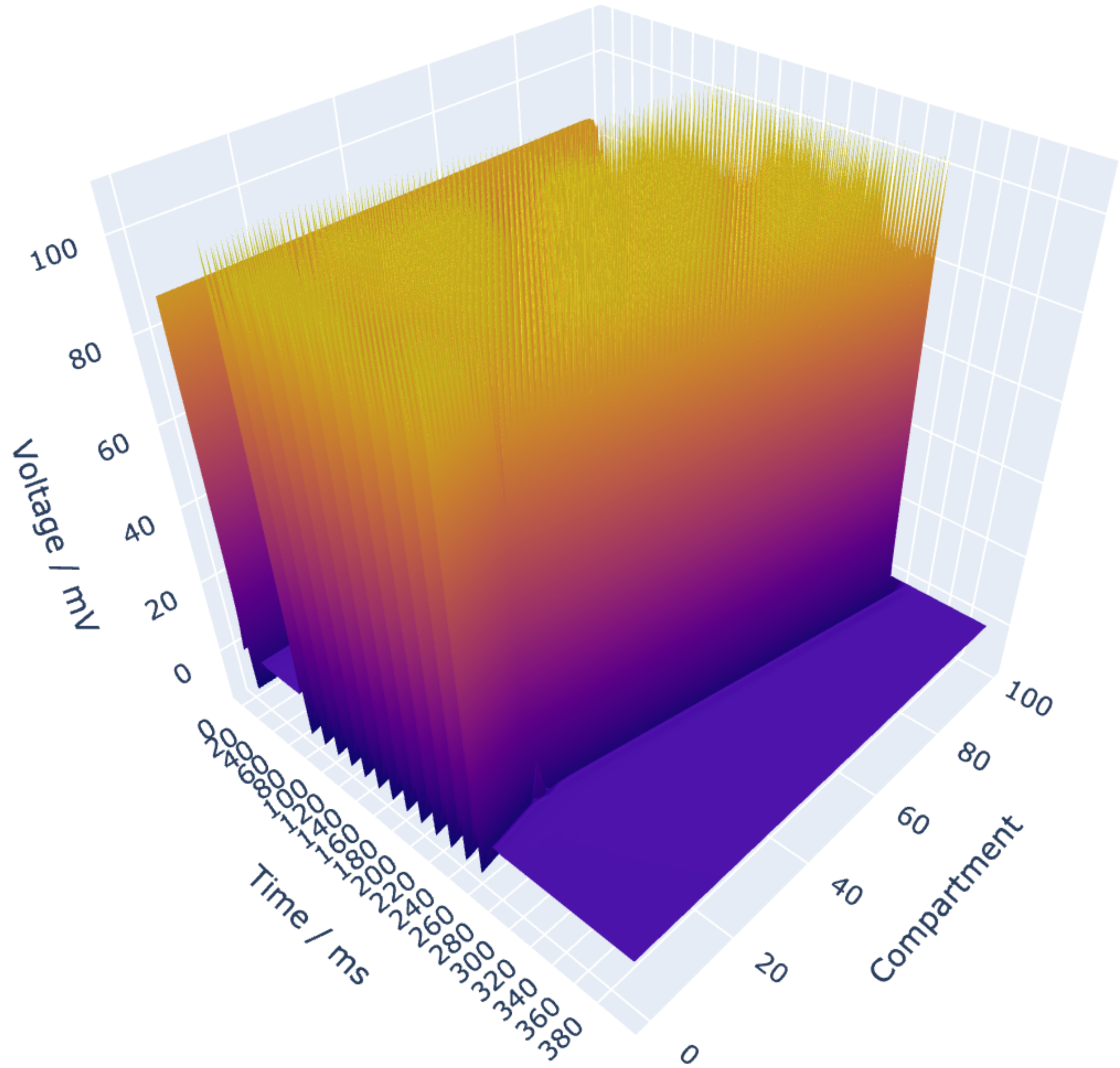

Figure 5: Voltage response to $I_e(14, t)$ of $20\,\mu$A amplitude.

HH

November 28, 2019

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import find_peaks
import plotly.graph_objects as go
%matplotlib inline
```

[ ]:
```
def threshold_detect(signal):
    peaks, _ = find_peaks(signal, prominence=1)
    return peaks
```

[ ]:
```
class Neuron:

    def __init__(self, E_m, C_m, E_na, E_k, E_l, g_na, g_k, g_l):
        self.E_m = E_m
        self.C_m = C_m
        self.E_na = E_na
        self.E_k = E_k
        self.E_l = E_l
        self.g_na = g_na
        self.g_k = g_k
        self.g_l = g_l

    def _update_gating(self, V_t, dt, m_t, h_t, n_t):
        '''Update gating variables'''

        beta_m = 4*np.exp(-V_t/18)
        beta_h = 1/(1 + np.exp(-(V_t - 30)/10))
        beta_n = 0.125*np.exp(-V_t/80)

        if V_t == 25:
            alpha_m = 1
        else:
            alpha_m = 0.1*((V_t - 25)/(1 - np.exp(-(V_t - 25)/10)))
        alpha_h = 0.07*np.exp(-V_t/20)
        if V_t == 10:
            alpha_n = 0.1
        else:
            alpha_n = 0.01*((V_t - 10)/(1 - np.exp(-(V_t - 10)/10)))
```

```
        m_t1 = m_t + dt*(alpha_m*(1 - m_t) - beta_m*m_t)
        h_t1 = h_t + dt*(alpha_h*(1 - h_t) - beta_h*h_t)
        n_t1 = n_t + dt*(alpha_n*(1 - n_t) - beta_n*n_t)

        return m_t1, h_t1, n_t1

    def _plot_response(self, V, t_e, t_s, dt, save_fig=False):
        '''Plot 2D responses'''

        figure = plt.figure(figsize=(9, 5))
        plt.grid(alpha=0.4)
        xi = np.arange(0, len(V), 1)*dt
        plt.plot(xi, V)
        plt.axhline(y=0, color='k', linestyle='--', alpha=0.4)
        plt.axvline(x=t_e, color='r', linestyle='--', alpha=0.4)
        plt.axvline(x=t_s, color='r', linestyle='--', alpha=0.4)
        plt.xlabel('Time / ms', fontsize=15)
        plt.ylabel('Voltage / mV', fontsize=15)
        plt.xticks(fontsize = 10)
        plt.yticks(fontsize = 10)

        if save_fig:
            plt.savefig('Figure.png')
        return

    def _plot_response_3D(self, V, dt, save_fig=False):
        '''Plot 3D responses'''

        vals = np.arange(0, V.shape[1], 2000)
        ticks = vals*dt

        layout = go.Layout(scene = dict(
                            xaxis = dict(
                                title = 'Time / ms',
                                tickmode = 'array',
                                tickvals = vals,
                                ticktext = ticks),
                            yaxis_title='Compartment',
                            zaxis_title='Voltage / mV'),
                            width=700,
                            margin=dict(r=20, b=10, l=10, t=10))

        fig = go.Figure(data=go.Surface(z=V), layout=layout)
        fig.show()

        if save_fig:
```

```
            fig.write_image("fig1.png")
        return

    def simulate(self, I_t, dt, time, t_e=None, t_s=None, plot=True,␣
↪save_fig=False):

        num_time_bins = int(time/dt)

        V_t  = 0
        V_course = np.zeros(num_time_bins)

        m_t  = 0
        h_t  = 0
        n_t  = 0

        for i in np.arange(0, int(time/dt), 1):

            if t_e:
                if (i*dt) < t_e:
                    I = I_t[0]
                elif t_e <= (i*dt) and (i*dt) < t_s:
                    I = I_t[1]
                else:
                    I = I_t[2]
            else:
                I = I_t

            m_t, h_t, n_t = self._update_gating(V_t, m_t, h_t, n_t)

            leak = self.g_l*(V_t - self.E_l)
            sod  = self.g_na*(m_t**3)*h_t*(V_t - self.E_na)
            pot  = self.g_k*(n_t**4)*(V_t - self.E_k)
            V_course[i] = V_t + (dt/self.C_m)*(I - leak - sod - pot)
            V_t = V_course[i]

        if plot:
            self._plot_response(V_course, t_e, t_s, dt, save_fig=save_fig)

        return V_course

    def simulate_multi_compartment(self, N, j, g_ax, I_t,
                                   dt, time, t_e=None, t_s=None, plot=True,␣
↪save_fig=False):

        num_time_bins = int(time/dt)

        V_t = np.zeros(N)
```

```python
        V_course = np.zeros((N, num_time_bins))

        m_t  = np.zeros(N)
        h_t  = np.zeros(N)
        n_t  = np.zeros(N)

        right = np.array([i for i in range(1, j)][::-1])
        left = np.array([i for i in range((j+1), (N-1))])

        for i in np.arange(0, int(time/dt), 1):

            if t_e:
                if (i*dt) < t_e:
                    I = I_t[0]
                elif t_e <= (i*dt) and (i*dt) < t_s:
                    I = I_t[1]
                else:
                    I = I_t[2]
            else:
                I = I_t

            m_t[j], h_t[j], n_t[j] = self._update_gating(V_t[j], dt, m_t[j], h_t[j], n_t[j])
            V_course[j, i] = V_t[j] + (dt/self.C_m)*(I + self.g_l*(self.E_l - V_t[j])
                                                    + self.g_na*np.power(m_t[j], 3)*h_t[j]*(self.E_na - V_t[j])
                                                    + self.g_k*np.power(n_t[j], 4)*(self.E_k - V_t[j])
                                                    + g_ax*(V_t[j-1] - V_t[j])
                                                    + g_ax*(V_t[j+1] - V_t[j]))
            V_t[j] = V_course[j, i]

            for r in right:
                m_t[r], h_t[r], n_t[r] = self._update_gating(V_t[r], dt, m_t[r], h_t[r], n_t[r])
                V_course[r, i] = V_t[r] + (dt/self.C_m)*(self.g_l*(self.E_l - V_t[r])
                                                    + self.g_na*np.power(m_t[r], 3)*h_t[r]*(self.E_na - V_t[r])
                                                    + self.g_k*np.power(n_t[r], 4)*(self.E_k - V_t[r])
                                                    + g_ax*(V_t[r-1] - V_t[r])
```

```
                                                                + g_ax*(V_t[r+1] -␣
↪V_t[r]))
                V_t[r] = V_course[r, i]

            for l in left:
                m_t[l], h_t[l], n_t[l] = self._update_gating(V_t[l], dt,␣
↪m_t[l], h_t[l], n_t[l])
                V_course[l, i] = V_t[l] + (dt/self.C_m)*(self.g_l*(self.E_l -␣
↪V_t[l])
                                                                + self.g_na*np.
↪power(m_t[l], 3)*h_t[l]*(self.E_na - V_t[l])
                                                                + self.g_k*np.
↪power(n_t[l], 4)*(self.E_k - V_t[l])
                                                                + g_ax*(V_t[l+1] -␣
↪V_t[l])
                                                                + g_ax*(V_t[l-1] -␣
↪V_t[l]))
                V_t[l] = V_course[l, i]

            c = 0
            m_t[c], h_t[c], n_t[c] = self._update_gating(V_t[c], dt, m_t[c],␣
↪h_t[c], n_t[c])
            V_course[c, i] = V_t[c] + (dt/self.C_m)*(self.g_l*(self.E_l -␣
↪V_t[c])
                                                            + self.g_na*np.power(m_t[c],␣
↪3)*h_t[c]*(self.E_na - V_t[c])
                                                            + self.g_k*np.power(n_t[c],␣
↪4)*(self.E_k - V_t[c])
                                                            + g_ax*(V_t[c+1] - V_t[c]))
            V_t[c] = V_course[c, i]

            c = (N-1)
#              m_t[c], h_t[c], n_t[c] = self._update_gating(V_t[c], dt, m_t[c],␣
↪h_t[c], n_t[c])
            V_course[c, i] = 0
            V_t[c] = V_course[c, i]

        if plot:
            self._plot_response_3D(V_course, dt, save_fig=save_fig)

        return V_course
```

[ ]:
```
E_m = 0
C_m = 1
E_na = 115
E_k = -12
```

```
E_l = 10.6
g_na = 120
g_k = 36
g_l = 0.3

a = Neuron(E_m, C_m, E_na, E_k, E_l, g_na, g_k, g_l)
```

[ ]:
```
I = [0, 8, 0]
t_e = 50
t_s = 300
dt = 0.01 #ms
time = t_s+100 #ms

v = a.simulate(I, dt, time, t_e, t_s, save_fig=False)
```

[ ]:
```
I = np.arange(0, 20, 0.25)
dt = 0.01 #ms
time = 800 #ms

v = np.zeros((len(I), int(time/dt)))

for c in range(len(I)):
    v[c, :] = a.simulate(I[c], dt, time, plot=False, save_fig=False)
```

[ ]:
```
r = []
for i in range(v.shape[0]):
    trace = v[i, 20000:]
    x = threshold_detect(trace)
    r.append(np.float(len(x))/np.float(0.6))
```

[ ]:
```
figure = plt.figure(figsize=(9, 5))
plt.grid(alpha=0.4)
plt.xlabel('Applied current / ţA', fontsize=15)
plt.ylabel('Firing rate / s^-1', fontsize=15)
plt.plot(I, r)
# plt.scatter(I, r, marker='x', alpha=0.6)
# plt.savefig('Firing_rate.png')
```

[ ]:
```
N = 100
j = 14
g_ax = 0.5
I = [0, 20, 0]
t_e = 60
t_s = 260
dt = 0.01 #ms
time = 400 #ms
a.simulate_multi_compartment(N, j, g_ax, I, dt, time, t_e, t_s)
```

[ ]:
```

```