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)))
```

```python
        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)
```

```
        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)
```

[ ]:
```

```