HH

November 27, 2019

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

[ ]:
```
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 simulate(self, I_t, dt, time, t_e=None, t_s=None, plot=True,␣
 ↪save=False):

        num_time_bins = int(time/dt)

        V_t  = 0
        V_course = np.zeros(num_time_bins)

        m_t  = 0
        m_t1 = np.zeros(num_time_bins)

        h_t  = 0
        h_t1 = np.zeros(num_time_bins)

        n_t  = 0
        n_t1 = np.zeros(num_time_bins)

        alpha_m = 0
        alpha_h = 0
        alpha_n = 0
```

```
        beta_m = 0
        beta_h = 0
        beta_n = 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

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

            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:
            figure = plt.figure(figsize=(9, 5))
            plt.grid(alpha=0.4)
```

```python
            xi = np.arange(0, len(V_course), 1)*dt
            plt.plot(xi, V_course)
            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)
            plt.title('Hodgkin-Huxley neurite model with %.2f ţA current␣
↪injected'%I, fontsize=17)
            if save:
                plt.savefig('Current_%dA.png'%I_t[1])

        return V_course
```

[ ]:
```python
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)
```

[ ]:
```python
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=False)
```

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

[ ]:
```python
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=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')
```

[ ]:
```

```