Xuenan Wang
NetID: xw336
Email: roderick_wang@outlook.com

CS 525 Homework 1

Questions:

1. IF model will continually integrate input voltage no matter how small the input is and as long as input is there, the total voltage will approach infinity. LIF model will probably ignore a very small input because as the input integrate, the leaking part will set the voltage back to rest.

2. For a very large input, IF model will integrate voltage incredibly fast and approach infinity. And LIF model will integrate voltage as well but at a relative slow rate because of the leaking mechanism.

3. The most important limitation of LIF model is that this model is isolated and can not memorize any situation or history from other neuron, or even previous status of itself. And this will result in problems like some neurons can not be precisely modeled. For instance, there is a phenomenon call adaptation in regular-spiking neuron and it's a slow process that builds up over several spikes, since LIF model can not memorize previous spike, this adaptation can not be captured. Also, LIF model can not show the depolarization process after one spike.

Programming:

1.

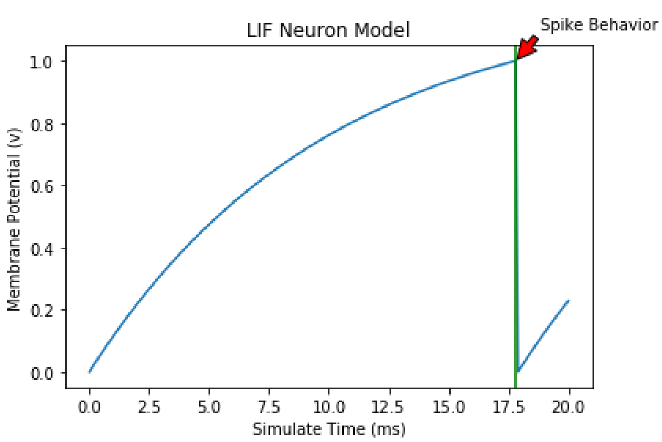

Figure 1. Spike a LIF neuron with input of 1.2 and simulate time of 20

Figure 1 shows a clear spike behavior and a potential decay over time. The green line in figure 1 indicate the time where the neuron spike.

2.

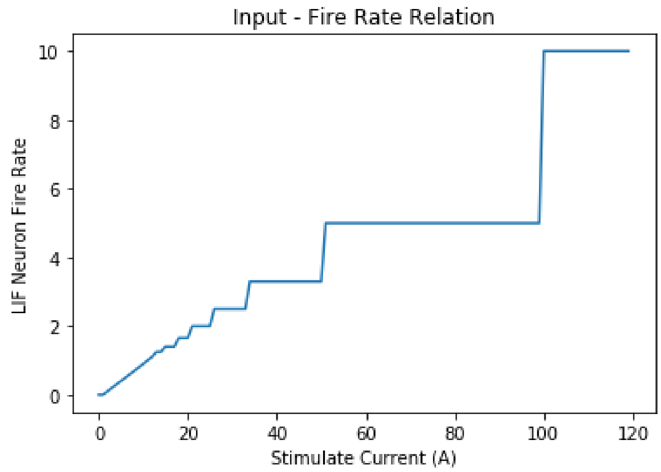

Figure 2. LIF neuron fire rate vs stimulate current

Figure 2 shows that as the stimulate current increase, the LIF neuron fire rate increase as well. However, when the stimulate current goes up to a certain value, the fire rate does not continue go up, as Figure 3 shows.

3.

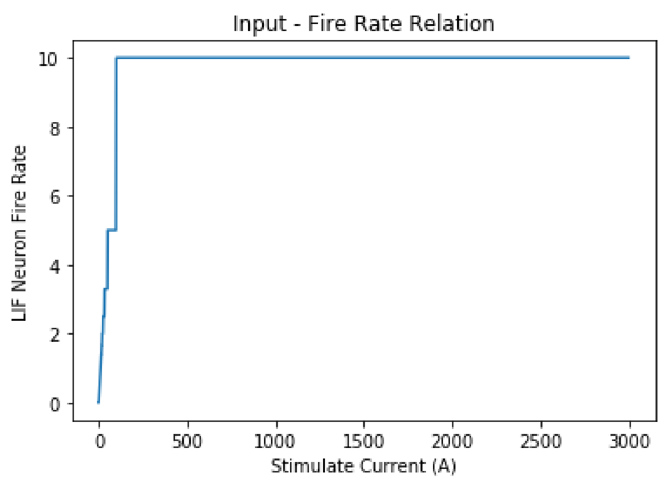

Figure 3. As stimulate current increase to a certain value, the fire rate does not continue increase

If continually increase the stimulate current, the fire rate does not go up. Because every neuron has a maximum fire speed and no matter how strong the stimulation is, neuron can not break this limit.

4.

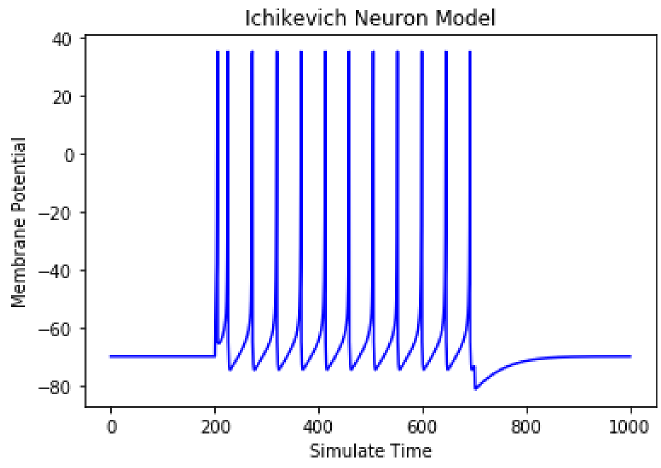

Figure 4. Simulation of Ichikevich Neuron Model

5.

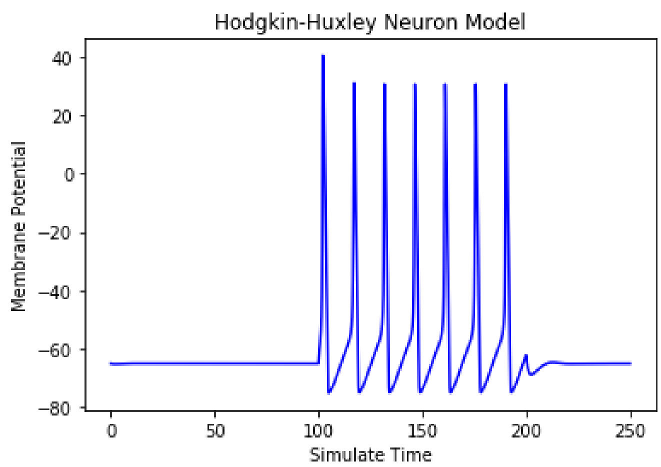

Figure 5. Simulation of Hodgkin-Huxley Neuron Model

Source Code:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import integrate
from random import *
import math

class LIF_Neuron:
        def __init__(self, simulate_time, 
input_number):
                self.simulate_time = 
simulate_time
                self.d_time = 0.1
                self.membrane_potential = 
zeros(int((self.simulate_time / self.d_time)
+ 1))
                self.threshold = 1
                self.rest_potential = 0
                self.initial_potential = 0
                self.membrane_resistance = 1
                self.membrane_capacitance = 
10
                self.time_constant = 
self.membrane_capacitance * 
self.membrane_resistance
                self.firing_rate = 10
                self.potential_difference = 0
                self.spike_number = 0
                self.not_spike_amount = 0
                self.spike_amount = 0
                self.fire_rate_array = []
                self.total_fire_rate = 0
                self.weight = []
                self.spike_time = []
                for i in range(input_number):
                        random_weight = 
math.ceil(uniform(0, 2000) - 1000) / 1000

        self.weight.append(random_weight)

        def stimulate_neuron(self, 
stimulate_current):
                self.ctr = 0.0
```

```
                self.spike_amount = 0
                self.membrane_potential =
zeros(int((self.simulate_time / self.d_time)
+ 1))
                self.potential_difference = 0
                for i in range(1,
len(self.membrane_potential)):

        self.potential_difference = (-
1*self.membrane_potential[i-
1]+self.membrane_resistance*stimulate_curren
t)

#print("membrane_potential:",self.membrane_p
otential)

        self.membrane_potential[i] =
self.membrane_potential[i-
1]+self.potential_difference/self.time_const
ant*self.d_time

        if(self.membrane_potential[i] >=
self.threshold):

        self.spike_time.append(i /
len(self.membrane_potential) *
simulate_time)

        self.membrane_potential[i] =
self.rest_potential

        self.spike_amount += 1

                return
(math.ceil((self.spike_amount /
self.simulate_time) * 1000)) / 1000

        def plot_neuron(self):
                simulate_time = arange(0,
self.simulate_time + self.d_time,
self.d_time)
                plt.plot(simulate_time,
self.membrane_potential)
                plt.xlabel('Simulate Time
(ms)')
                plt.ylabel('Membrane
Potential (v)')
                plt.title('LIF Neuron Model')
                plt.annotate('Spike
Behavior', xy = (self.spike_time[1],
self.threshold), xytext =
(self.spike_time[1] + 1, self.threshold +
1), arrowprops = dict(facecolor = 'red',
shrink = 0.05))
                plt.axvline(x =
self.spike_time[1], color = 'g')
                plt.show()
                print('Spike rate for this
input is ', self.spike_amount /
self.simulate_time)

        def calculate_fire_rate(self):
                simulate_time = arange(0,
self.simulate_time + self.d_time,
self.d_time)
                return self.spike_amount /
self.simulate_time

class Ichikevich_Neuron:
        def __init__(self):
                self.simulate_time = 1000
                self.d_time = 0.5
                self.param_a = 0.02
                self.param_b = 0.2
                self.param_c = -65
                self.param_d = 8
                self.lapp = 10
                self.tr = np.array([200,
700])//self.d_time
                self.T =
int(self.simulate_time/self.d_time)
                self.v = np.zeros(self.T)
                self.u = np.zeros(self.T)
                self.v[0] = -70
                self.u[0] = -14

        def stimulate_neuron(self):
                for i in np.arange(self.T-1):
                        if i>self.tr[0] and
i<self.tr[1]:
                                l = self.lapp
                        else:
                                l = 0
                        if self.v[i]<35:
                                dv =
(0.04*self.v[i]+5)*self.v[i]+140-self.u[i]
                                self.v[i+1] =
self.v[i]+(dv+l)*self.d_time
                                du =
self.param_a*(self.param_b*self.v[i]-
self.u[i])
                                self.u[i+1] =
self.u[i]+self.d_time*du
                        else:
                                self.v[i] = 35
                                self.v[i+1] =
self.param_c
                                self.u[i+1] =
self.u[i] + self.param_d

        def plot_neuron(self):
                tvec = np.arange(0,
self.simulate_time, self.d_time)
                plt.plot(tvec, self.v, 'b',
label = 'Voltage Trace')
                plt.xlabel('Simulate Time')
                plt.ylabel('Membrane
Potential')
                plt.title('Ichikevich Neuron
Model')
                plt.show()

class HodgkinHuxley_Neuron():
        def __init__(self):
                self.C_m = 1.0
                self.g_Na = 120.0
                self.g_K = 36.0
                self.g_L = 0.3
                self.E_Na = 50.0
                self.E_K = -77.0
                self.E_L = -54.387
                self.time = np.arange(0.0,
450.0, 0.01)

        def alpha_m(self, V):
            return 0.1*(V+40.0)/(1.0 -
math.exp(-(V+40.0) / 10.0))

        def beta_m(self, V):
            return 4.0*math.exp(-(V+65.0) /
18.0)
```

```
        def alpha_h(self, V):
            return 0.07*math.exp(-(V+65.0) /
20.0)

        def beta_h(self, V):
            return 1.0/(1.0 + math.exp(-
(V+35.0) / 10.0))

        def alpha_n(self, V):
            return 0.01*(V+55.0)/(1.0 -
math.exp(-(V+55.0) / 10.0))

        def beta_n(self, V):
            return 0.125*math.exp(-(V+65) /
80.0)

        def I_Na(self, V, m, h):
            return self.g_Na * m**3 * h * (V
- self.E_Na)

        def I_K(self, V, n):
            return self.g_K  * n**4 * (V -
self.E_K)

        def I_L(self, V):
            return self.g_L * (V - self.E_L)

        def I_inj(self, t):
            return 10*(t>100) - 10*(t>200) +
35*(t>300) - 35*(t>400)

        @staticmethod
        def dALLdt(X, t, self):
            V, m, h, n = X

            dVdt = (self.I_inj(t) -
self.I_Na(V, m, h) - self.I_K(V, n) -
self.I_L(V)) / self.C_m
            dmdt = self.alpha_m(V)*(1.0-m) -
self.beta_m(V)*m
            dhdt = self.alpha_h(V)*(1.0-h) -
self.beta_h(V)*h
            dndt = self.alpha_n(V)*(1.0-n) -
self.beta_n(V)*n
            return dVdt, dmdt, dhdt, dndt

        def stimulate_neuron(self):
            X = odeint(self.dALLdt, [-65,
0.05, 0.6, 0.32], self.time, args=(self,))
            V = X[:,0]
            m = X[:,1]
            h = X[:,2]
            n = X[:,3]
            ina = self.I_Na(V, m, h)
            ik = self.I_K(V, n)
            il = self.I_L(V)

            plt.title('Hodgkin-Huxley Neuron
Model')
            plt.plot(self.time, V, 'k')
            plt.ylabel('Membrane Potential')
            plt.xlabel('Simulate Time')
            plt.show()

if __name__ == "__main__":
        #LIF_Neuron Model
        simulate_time = 20
        stimulation_number = 5
        stimulate_current = 1.2
        lif_neuron =
LIF_Neuron(simulate_time,
stimulation_number)
        lif_neuron.stimulate_neuron(stimulate
_current)
        lif_neuron.plot_neuron()
        '''
        fire_rate = []
        for i in range(0, 1500):
            lif_neuron_new =
LIF_Neuron(simulate_time,
stimulation_number)

lif_neuron_new.stimulate_neuron(i)

fire_rate.append(lif_neuron_new.calculate_fi
re_rate())

        plt.plot(range(0, 1500), fire_rate)
        plt.xlabel('Stimulate Current (A)')
        plt.ylabel('LIF Neuron Fire Rate')
        plt.title('Input - Fire Rate
Relation')
        plt.show()
    '''
        #Ichikevich Neuron Model
        ichikevich_neuron =
Ichikevich_Neuron()
        ichikevich_neuron.stimulate_neuron()
        ichikevich_neuron.plot_neuron()

        #Hodgkin-Huxley Neuron Model
    hodgkin_huxley = HodgkinHuxley_Neuron()
    hodgkin_huxley.stimulate_neuron()
```