Shaivi Pandey

Machine Learning Assignment

Problem 1

Problem Statement

The Izhikevich neuron response model is one way to simulate neurons. While not as simple as integrate and fire, it is much simpler than the Hodgkin-Huxley model. The Izhikevich model is described by two differential equations, which are shown below.

$$\frac{dv}{dt} = 0.04v^2 + 5v + 140 - u + I(t), \frac{du}{dt} = a(bv - u)\ if\ v \geq 30, v = c, u = u + d$$

The first problem was to implement the Izhekevich neuron model in software, and to simulate the response of the neuron to increasing inputs. The simulation takes place over 500 time steps, with the input being applied after time step 50. Each simulation would use 500 steps to simulate the neuron for a desired input. The inputs ranged from 0 to 20 on .25 step intervals. Finally, after generating the neuron response curves, the *mean spike rate* R was to be generated by counting the number of spikes in the time interval 300 to 500, and dividing that number by 300. A plot of the mean spike rate R versus input, and neuron response for inputs 1, 5, 10, 15, and 20 were also desired plots.

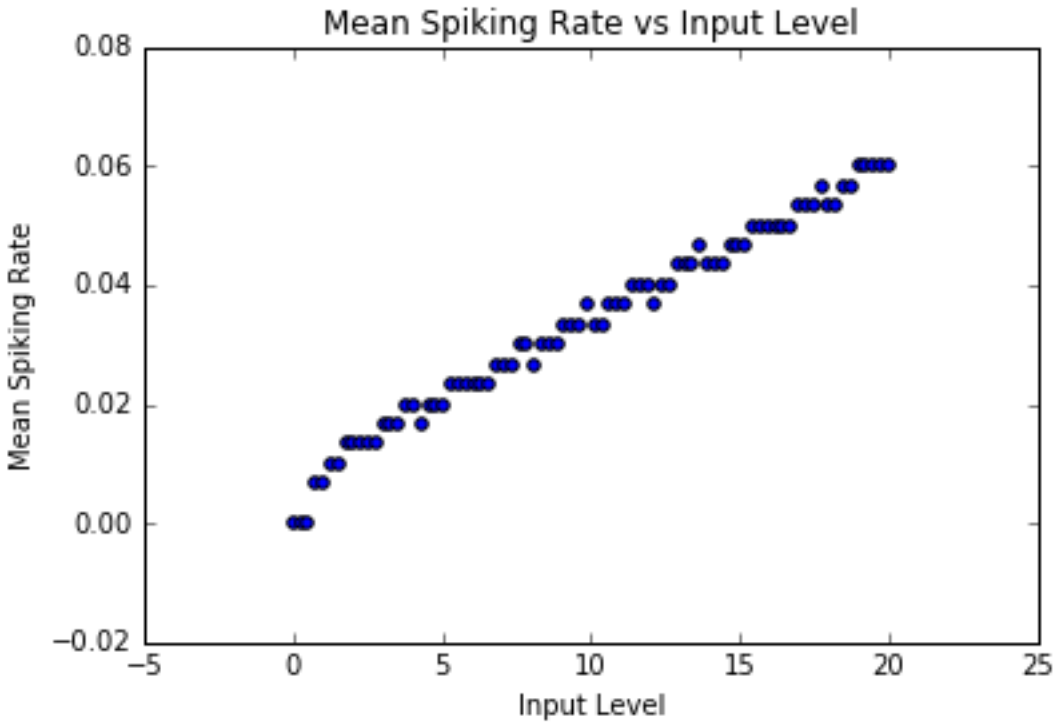
Implementation Overview

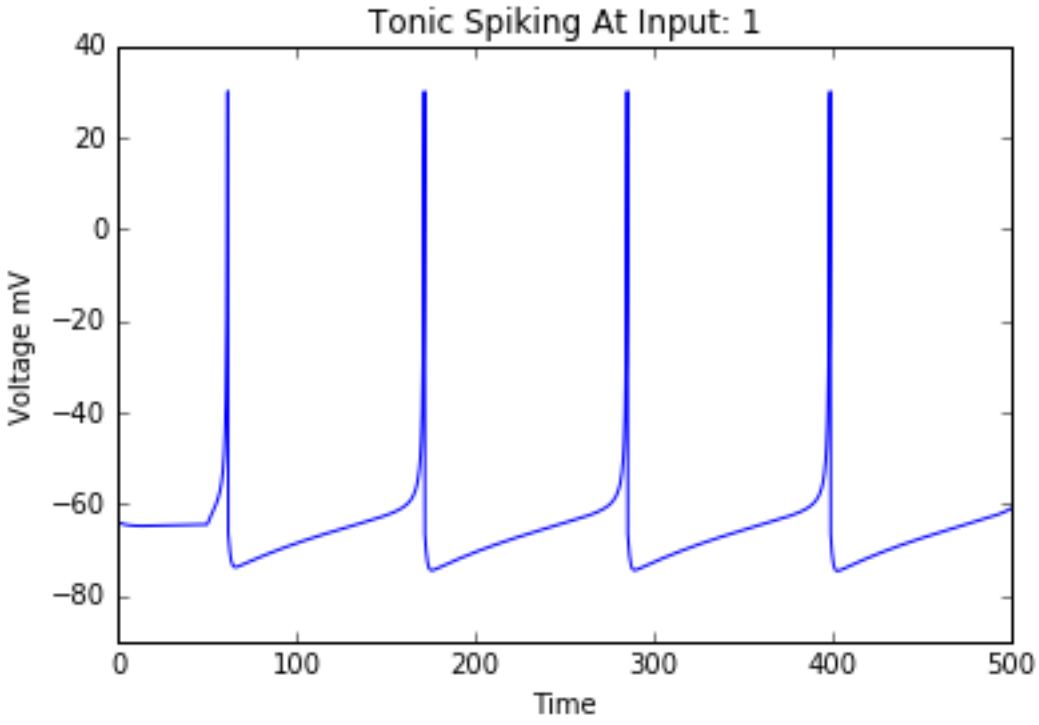
The simulations were implemented in python. First, a spiking neuron class was made. Then, a function for simulating a single neuron response (with the option to plot the response) was created. The function took an input, time set, and input start time. Finally, a function called hwPart1() was written, which looped through values 0 to 20 at .25 steps, and called the simulation function for each input.

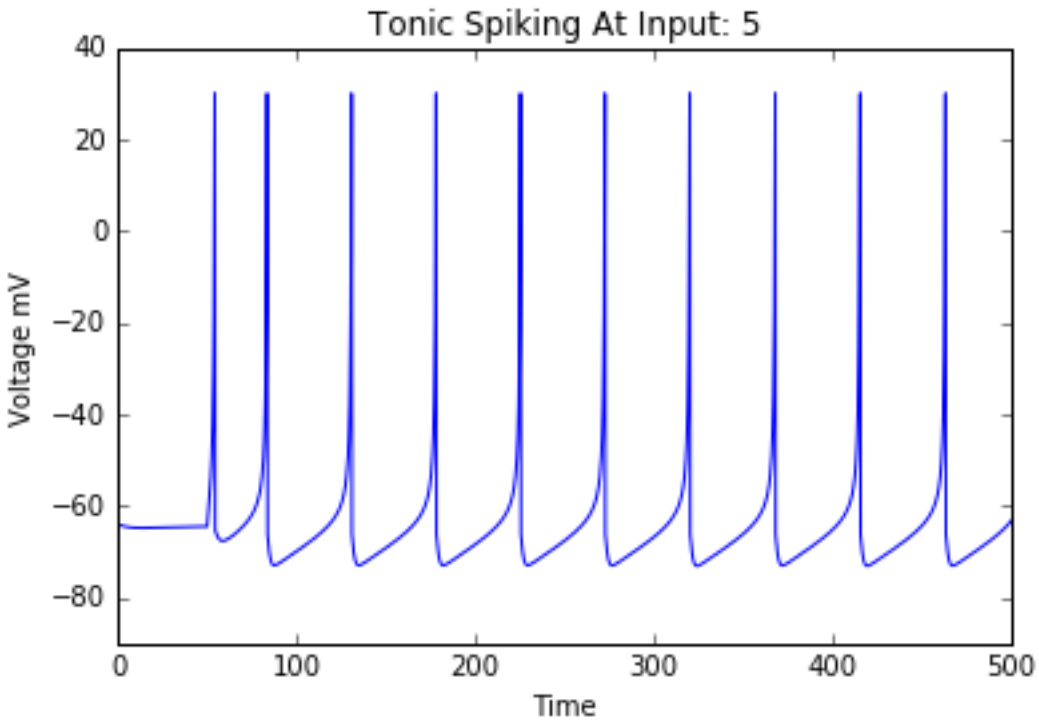
Results and Conclusions Discussion

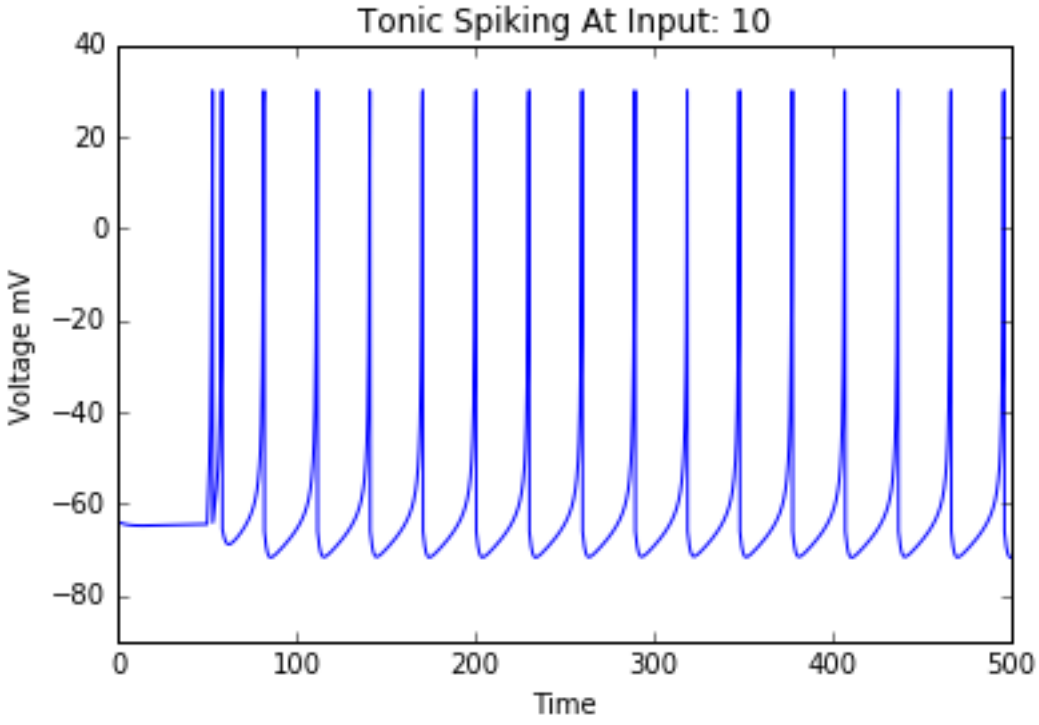
Looking at the plots, it is clearly evident that there is a linear relationship between the mean spiking rate and the input value. A line with slope 0.002666 average spikes per input increase fits the scatter plot well. What these plots show is that there is a strong correlation between spiking rates and input magnitude. We also don't see any issue with refractory periods for this neuron in the input range from 0 to 20. Due to its behavior, we can take the neuron as a function that maps from the input space to a new space where the input information is encoded in the frequency of spikes. In this way, the neuron is converting the data from a set of levels, to a set of frequencies for impulse spikes described by the Izhekevich model.

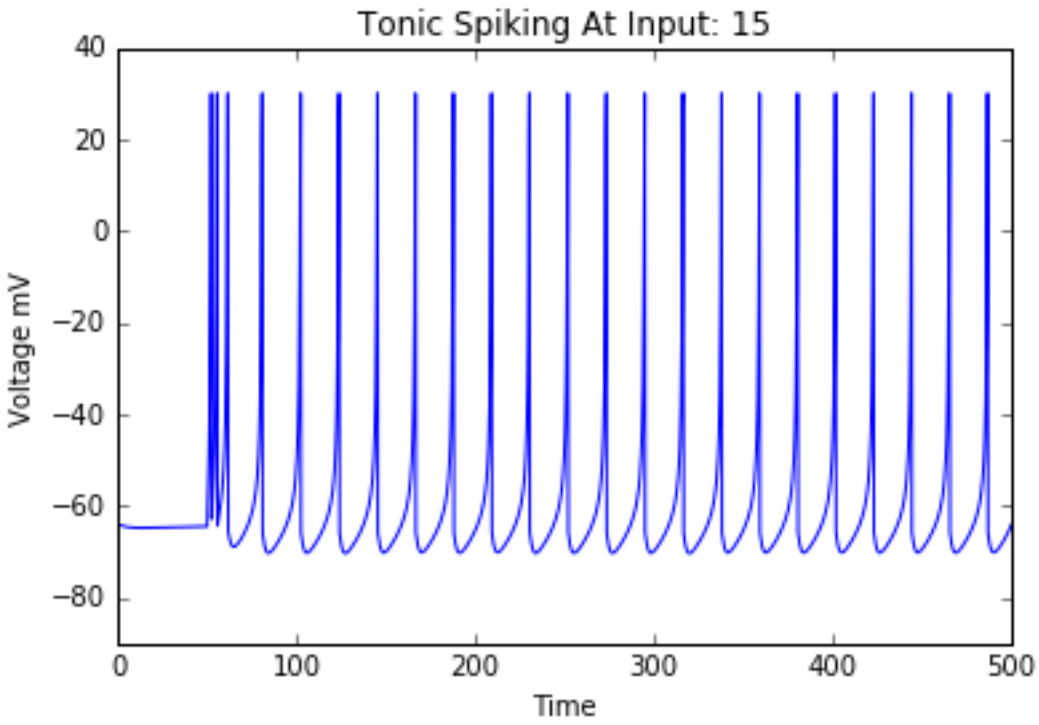
Problem Plots

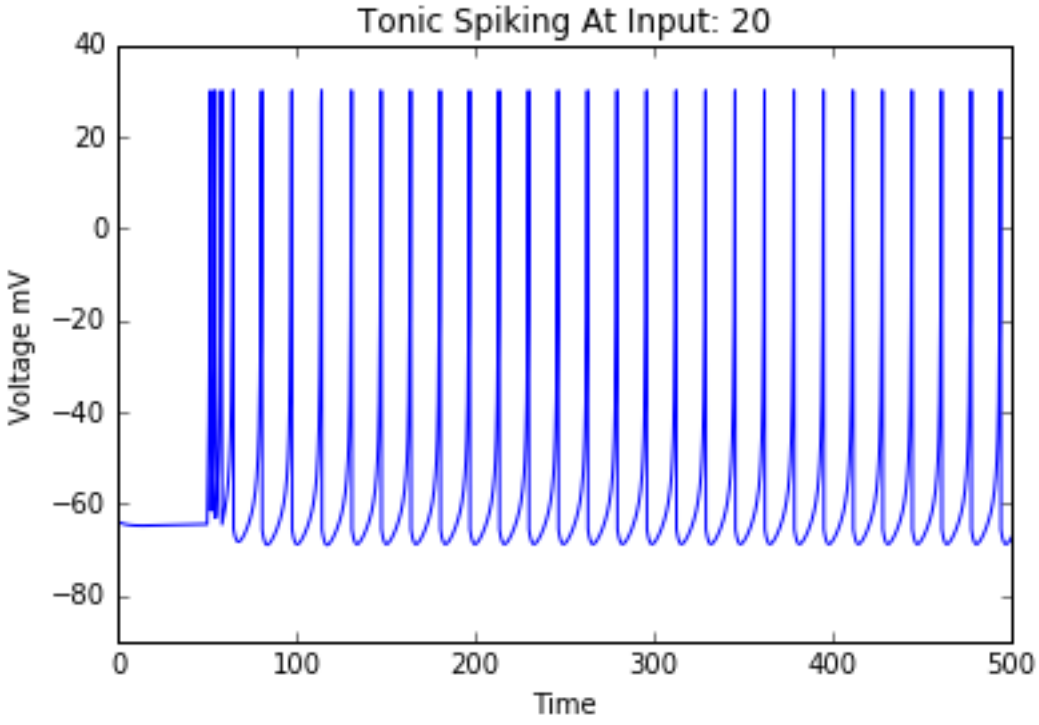





Tonic Spiking At Input: 5
Voltage mV
Time
40
20
0
−20
−40
−60
−80
0
100
200
300
400
500


Tonic Spiking At Input: 10
Voltage mV
Time
40
20
0
−20
−40
−60
−80
0
100
200
300
400
500


Tonic Spiking At Input: 15
Voltage mV
Time
40
20
0
−20
−40
−60
−80
0
100
200
300
400
500


Tonic Spiking At Input: 20
Voltage mV
Time
40
20
0
−20
−40
−60
−80
0
100
200
300
400
500

Code

```
import numpy as np
import matplotlib.pyplot as plt

tau = .25
steps = 500
tspan = np.linspace(0, steps, num=steps/tau, endpoint=True)
#T1 is the time interval at which the input first rises.
T1 = 50

def runSingleNeuronSpiking(inputIValue, tspan, steps=500, T1=50, plot=False):
    neuron1 = SpikingNeuron()
    inputI = 0
    voltageList = []
    for time in tspan:
        if time < T1:
            inputI = 0
        else:
            inputI = inputIValue
        #Stimulate the neuron.
        voltageList.append(neuron1.stimulate(inputI))

    #When finished, plot the function output for the time span.
    if plot:
        plt.plot(tspan, voltageList)
        plt.axis([0, steps, -90, 40])
        plt.xlabel('Time')
        plt.ylabel('Voltage mV')
        plt.title('Tonic Spiking At Input: ' + str(inputIValue))
        plt.show()

    return voltageList

class SpikingNeuron:

    def __init__(self):
        self.a = 0.02
        self.b = 0.25
        self.c = -65
        self.d = 6
        self.V = -64
        self.tau = 0.25
```

```
        self.u = self.b * self.V

    def stimulate(self, inputI):
        #Do an incremental membrane potential update using the differential equation times the
discretization.
        #Vn+1 = Vn + deltaVn
        #deltaVn = delta_t * (dV/dt)
        self.V = self.V + self.tau*( (0.04*(self.V**2)) + (5*self.V) + 140 - self.u + inputI )
        vOut = self.V
        #Do an increment for the 'u' differential equation.
        self.u = self.u + self.tau*(self.a*((self.b*self.V) - self.u))

        #Set up a peak saturation and breakdown condition.
        if(self.V >= 30 ):
            vOut = 30
            self.V = self.c
            self.u = self.u + self.d

        return vOut

def hwPart1():
    meanSpikingRate = []
    inputArray = np.linspace(0, 20, 20/.25, endpoint=True)
    for inputValue in inputArray:
        #Simulate neuron for current input.
        voltageOutputList = runSingleNeuronSpiking(inputValue, tspan, plot=False)
        #Filter out the beginning time from 0 to 200.(Aka take times 200 to 500.)
        voltageOutputList = voltageOutputList[800:]
        #Find the mean spiking rate. When V is set to 30, that means a spike has occurred.
        meanSpikingRate.append(voltageOutputList.count(30)/300)

    plt.scatter(inputArray, meanSpikingRate)
    plt.xlabel('Input Level')
    plt.ylabel('Mean Spiking Rate')
    plt.title('Mean Spiking Rate vs Input Level')
    plt.show()

    inputArray = [1, 5, 10, 15, 20]
    for inputValue in inputArray:
        #Simulate neuron for current input.
        voltageOutputList = runSingleNeuronSpiking(inputValue, tspan, plot=True)
```

Shaivi Pandey

Problem 2

Problem Statement

The second problem asked for a simulation of a two neuron model. Now however, the neurons would be arranged sequentially, with the first neuron denoted as neuron A, and the second neuron denoted as neuron B. The simulation was to occur in the same was as for the first problem, except now, and extra thresholding function would be used to filter the input to neuron B. The filter conditions were:

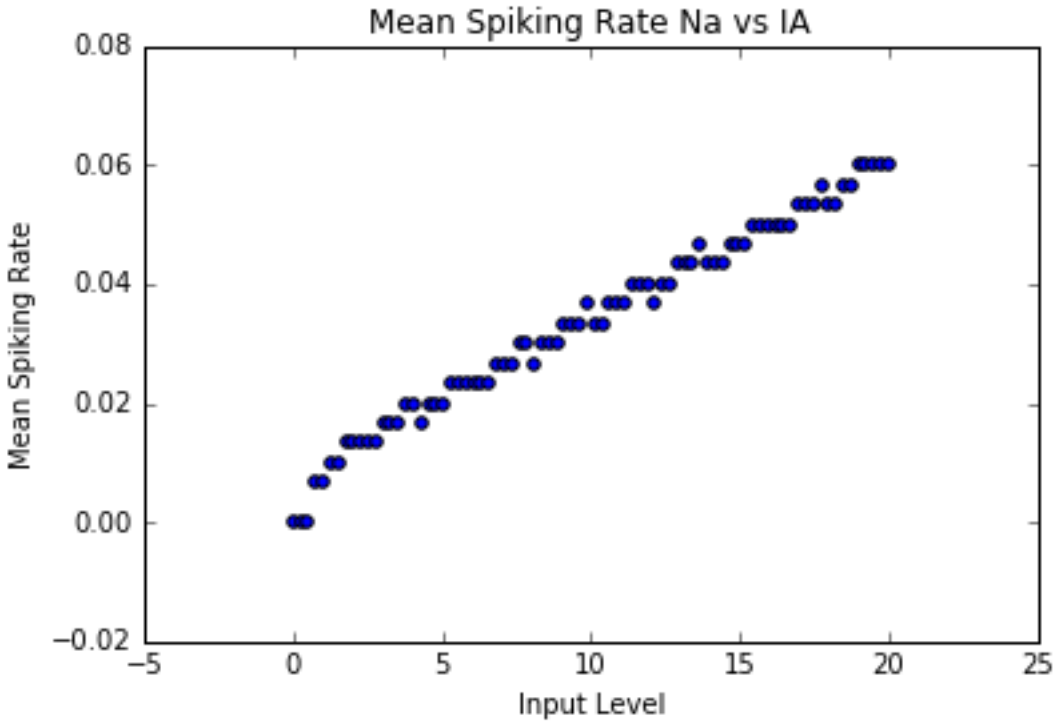
$$I_B = w * y_A, where\ w = 100, y_A(t) = 1\ if\ V_a(t) = 30, else\ y_A(t) = 0$$

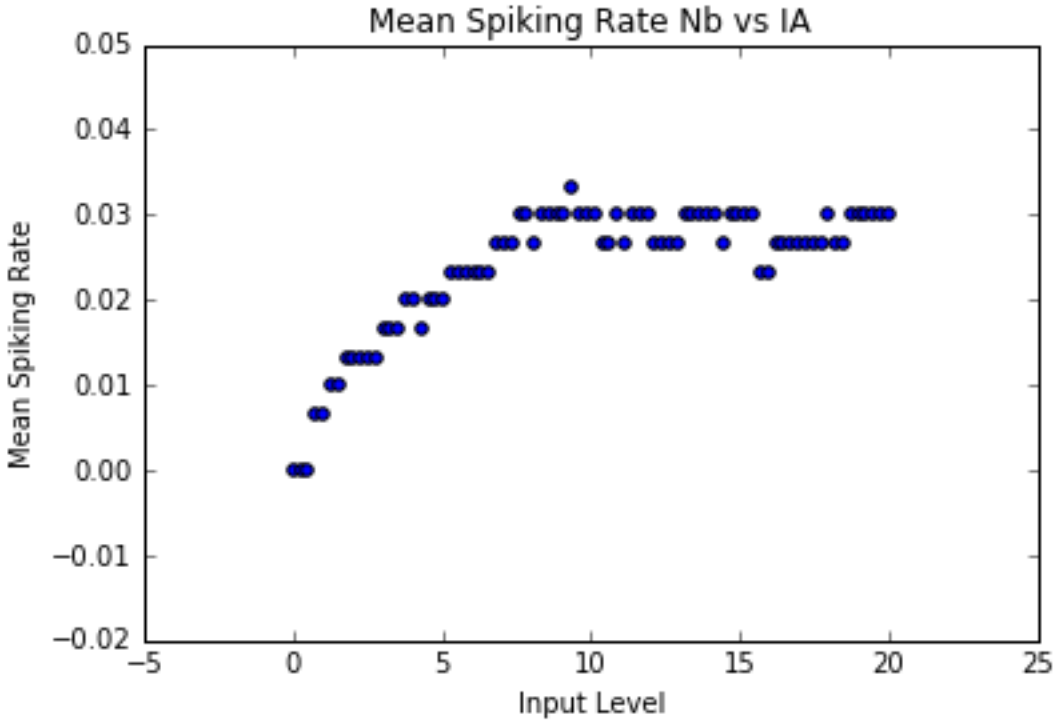
With the conditions stated above, neuron B will only receive an input when the output of neuron A is at its peak. Also, the input to neuron B will be at a level of 100 for a single time step, but will be 0 at every other time. In addition to this simulation, 3 plots were asked for. They are: mean spike rate of neuron A vs the input to A, mean spike rate of neuron B vs the input to A, and the mean spike rate of neuron A vs mean spike rate of neuron B for each simulation point.

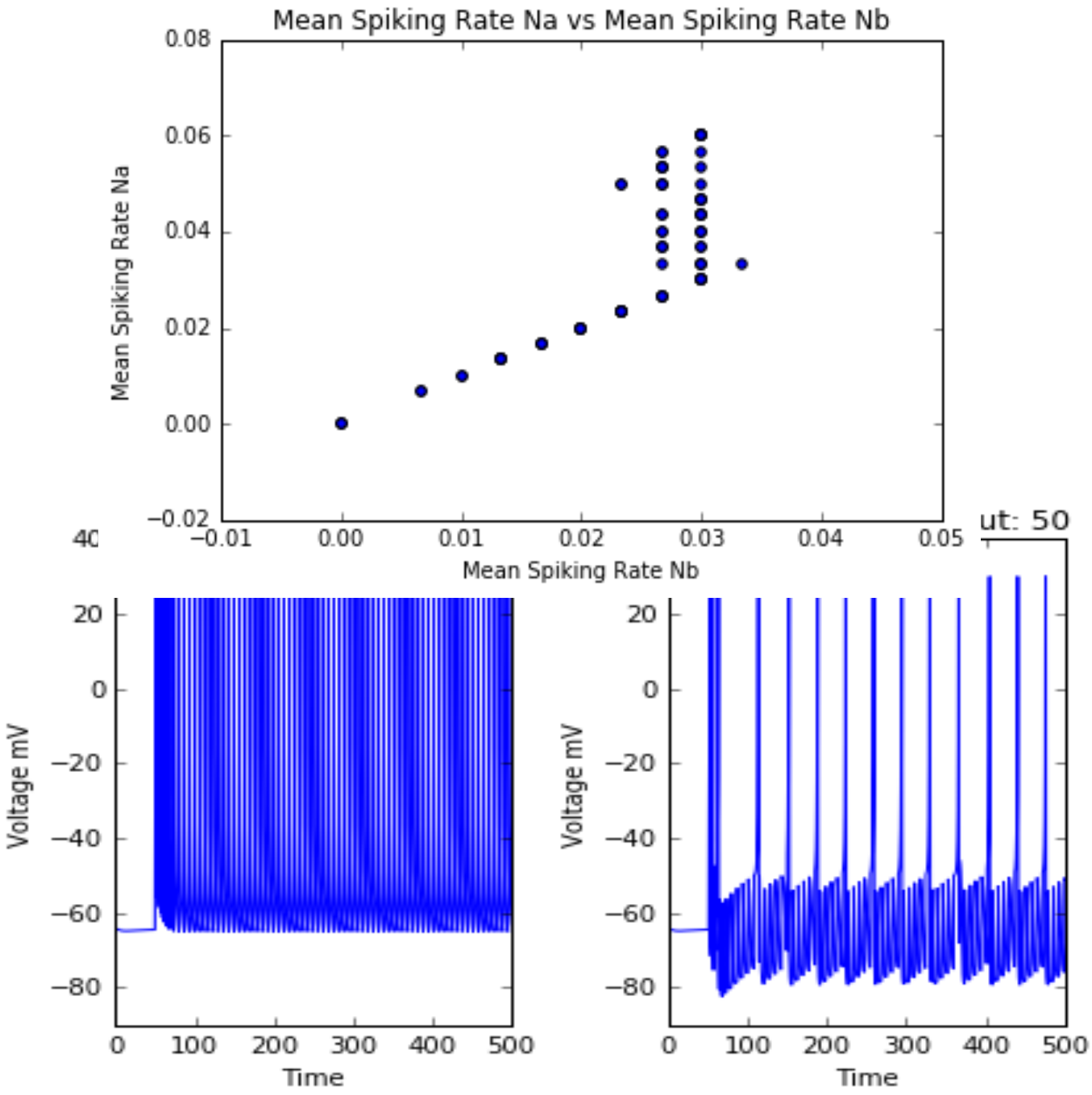
Results and Conclusions Discussion

Initially, both neurons have a relatively similar response to the inputs. However, at around input value 10, the mean firing rate of neuron B hits a wall. The mean spiking rate with which neuron B reaches its full peak value approaches 0.03. Looking at an exaggerated individual response plot for neuron A and neuron B at an input level of 50 shows that neuron B is responding, but it has an oscillating response up to the point that it actually fully fires. Looking back at the characteristic differential equations, this phenomenon can be explained by the interaction of U and V on each other's differential equations, along with the periodic input of 100 to neuron B. When the input level to neuron A is high, neuron A fires at a high mean spike rate. This causes the input to neuron B to flip on and off very quickly. Every time neuron B gets a spike, its output voltage V is pulled up. However, due to the value of U and the equation dU/dt being positively dependent on the value of V, the voltage output of neuron B is regulated. Only when V becomes high enough so that the value of constant b times V (b*V) is greater than U will neuron B increase its output until it reaches maximum peak. Under the condition that b*V – U > 0, the value of dU/dt will be positive and U will begin increasing (decreasing its negative value). This causes a chain reaction which ultimately leads to the output V reaching its maximum peak. At a higher level, what's going on is that the input from neuron A is coming when neuron B is still in a refractory period. Once the voltage for neuron B has reached a threshold sufficiently out of the refractory period, an input from neuron A is then able to trigger a peak response from neuron B.

Problem Plots







Extra plot of exaggerated input level 50. (Figure D.)

Code

```
def runSequentialNeuronSpikingModel(inputIValue, tspan, steps=500, T1=50, plot=False):
    neuron1 = SpikingNeuron()
    neuron2 = SpikingNeuron()
    input1I = 0
    weight = 100
    voltageN1List = []
    voltageN2List = []
    for time in tspan:
        if time < T1:
            input1I = 0
        else:
            input1I = inputIValue

        #Stimulate neuron 1.
        neuron1Output = neuron1.stimulate(input1I)
        voltageN1List.append(neuron1Output)
        #Stimulate neuron 2.
        if neuron1Output == 30:
            voltageN2List.append(neuron2.stimulate(weight*1))
        else:
            voltageN2List.append(neuron2.stimulate(0))

    #When finished, plot the function output for the time span.
    if plot:
        plt.plot(tspan, voltageN2List)
        plt.axis([0, steps, -90, 40])
        plt.xlabel('Time')
        plt.ylabel('Voltage mV')
        plt.title('Tonic Spiking At Input: ' + str(inputIValue))
        plt.show()

    return voltageN1List, voltageN2List

class SpikingNeuron:

    def __init__(self):
        self.a = 0.02
        self.b = 0.25
        self.c = -65
        self.d = 6
        self.V = -64
        self.tau = 0.25
        self.u = self.b * self.V
```

```
    def stimulate(self, inputI):
        #Do an incremental membrane potential update using the differential equation times the
discretization.
        #Vn+1 = Vn + deltaVn
        #deltaVn = delta_t * (dV/dt)
        self.V = self.V + self.tau*( (0.04*(self.V**2)) + (5*self.V) + 140 - self.u + inputI )
        vOut = self.V
        #Do an increment for the 'u' differential equation.
        self.u = self.u + self.tau*(self.a*((self.b*self.V) - self.u))

        #Set up a peak saturation and breakdown condition.
        if(self.V >= 30 ):
            vOut = 30
            self.V = self.c
            self.u = self.u + self.d

        return vOut


def hwPart2():
    meanSpikingRateN1 = []
    meanSpikingRateN2 = []
    inputArray = np.linspace(0, 20, 20/.25, endpoint=True)
    #inputArray = [0, 1, 5, 10, 15, 20]
    for inputValue in inputArray:
        #Simulate neuron for current input.
        voltageN1List, voltageN2List = runSequentialNeuronSpikingModel(inputValue, tspan,
plot=False)
        #Filter out the beginning time from 0 to 200.(Aka take times 200 to 500.)
        voltageN1List = voltageN1List[800:]
        voltageN2List = voltageN2List[800:]
        #Find the mean spiking rate. When V is set to 30, that means a spike has occurred.
        meanSpikingRateN1.append(voltageN1List.count(30)/300)
        meanSpikingRateN2.append(voltageN2List.count(30)/300)

    plt.scatter(inputArray, meanSpikingRateN1)
    plt.xlabel('Input Level')
    plt.ylabel('Mean Spiking Rate')
    plt.title('Mean Spiking Rate Na vs IA')
    plt.show()
    plt.clf()
    plt.scatter(inputArray, meanSpikingRateN2)
    plt.xlabel('Input Level')
```

```
plt.ylabel('Mean Spiking Rate')
plt.title('Mean Spiking Rate Nb vs IA')
plt.show()
plt.clf()
plt.scatter(meanSpikingRateN2, meanSpikingRateN1)
plt.xlabel('Mean Spiking Rate Nb')
plt.ylabel('Mean Spiking Rate Na')
plt.title('Mean Spiking Rate Na vs Mean Spiking Rate Nb')
plt.show()
```