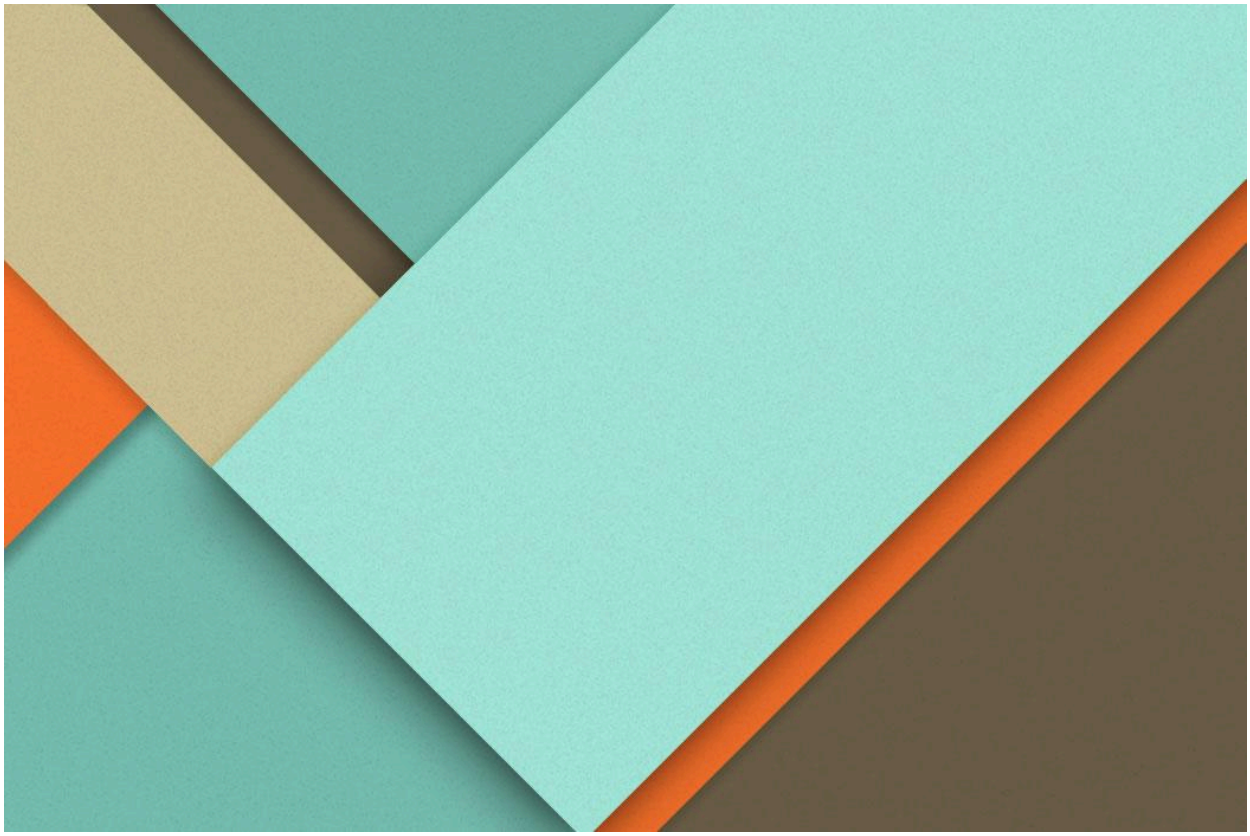


Neurons population

Katayoon Kobraei
Shahid Beheshti University of Tehran
Instructor: Dr. Kheradpisheh

Overview

This report presents the second assignment in the series, which builds upon the neuron models implemented previously by extending the simulation to neuronal populations. The objective is to model and analyze the behavior of interconnected neurons, simulating how they interact within a network. This exercise requires the creation of excitatory and inhibitory connections between neurons and involves testing various configurations of these connections to observe their effects on neural activity.

Goals

The assignment focuses on two primary tasks:

1. **Neuron-to-Neuron Connection**: In this section, the connection between two neurons is established, with different currents applied to each neuron. The experiment examines various combinations of excitatory and inhibitory interactions between the neurons, generating plots to illustrate their response to stimuli.
2. **Neuronal Population Simulation**: A larger network comprising multiple excitatory and inhibitory neurons is simulated. The behavior of the entire population is analyzed using raster plots, which visualize the spiking patterns of the neurons over time. Additionally, connections between different neuronal populations are implemented to observe their influence on one another after a certain number of spikes.

In summary

We first examine the connection between two neurons of different types: excitatory-excitatory, excitatory-inhibitory, and inhibitory-inhibitory. In the next section, we explore the connections between more than two neurons, for instance, eight neurons. Finally, in the last section, we will investigate the connection between multiple neuron populations.

Section 1: Examining the Connection Between Two Neurons

STEP 1

In this section, as the first step, we define a class of a neuron of the LIF (Leaky Integrate-and-Fire) type. As we know, this neuron model must take certain parameters as input, such as the type and amount of incoming current to determine potential, resistance, and capacitance, along with time steps. Finally, we define the type of neuron, which can be either inhibitory or excitatory.

```
In [63]: class Neuron:
             def __init__(self,i_func = 0, i = 5, time_interval = 100, dt = 0.1, u_rest = 0, R = 1, C = 10, threshold = 1,
                          neuron_type = "excitatory", save_name="none"):
                 self.i = i
                 self.time_interval = time_interval
                 self.dt = dt
                 self.u_rest = u_rest
```

Note: For each input variable used in the functions, we initially assign a default value to ensure they work correctly.

Next, we begin constructing the necessary functions for the simple neuron. The first function defines a list where the potential energy of the neuron is recorded over specific time intervals and stored. This will become part of the neuron's characteristics for use in subsequent sections. The formula adds a delta of time to the initial time at each step and calculates the potential energy. If this potential energy exceeds a threshold, the neuron spikes and resets to its initial resting state.

```
def init_potencials(self):
    self.timer = np.arange(0, self.time_interval + self.dt, self.dt)
    u = [self.u_rest for i in range(len(self.timer))]
    self.i_init = [self.i_func(5, j) for j in self.timer]
    const = self.R * self.C

    for t in range(len(self.timer)):
        u[t] = u[t-1] + (((-u[t-1] + self.u_rest) + self.R * self.i_init[t]) * self.dt)/const
        if u[t] >= self.threshold or u[t] < self.u_rest:
            u[t] = self.u_rest
        self.u = u
        self.cur_time = t
```

The `Times` function defines the time intervals. Plotting functions visualize the potential of the neuron, the incoming current, and the voltage as a function of the current.

STEP 2

Next, we define a class for two connected neurons. This class includes features like the neurons themselves (which are objects of the base neuron class), their connections (represented by a list), the weights of inhibitory and excitatory neurons, the delay between them, etc.

```
In [64]: class First_neuron_group:
             def __init__(self, neurons, connections, ex_weight = 2, inh_weight = 2, delay = 1, counter = 500):
                 self.neurons = neurons
                 for i in neurons:
                     self.neroun_action.append(i.start())
                 self.conections = conections
```

One of the most important functions is the "action" function, which implements the connection between the two neurons. It checks each neuron in the list to determine if they are connected and examines the type of connection. If a spike occurs, it is added to the list of spikes for inhibitory or excitatory neurons, and the effects are accumulated.

```
    def action(self):
        length = len(self.neurons)
        self.spikes_effect = [[0] * length for i in range(self.counter)]
        for t in range(self.counter):
            for i in range(len(self.actions)):
                action_info = next(self.actions[i])
                if action_info == True :
                    for j in self.connections[i]:
                        if self.neurons[i].type == "excitatory":
                            self.ex_spikes.append(i + 1)
                            self.ex_spikes_times.append(t)
                            if t+self.delay < self.counter:
                                self.spikes_effect[t + self.delay][j] += self.ex_weight

                        if self.neurons[i].type == "inhibitory":
                            self.inhibitory_spikes.append(i + 1)
                            self.inhibitory_spikes_time.append(t)
                            if t+self.delay < self.counter:
                                self.spikes_effect[t+self.delay][j] += self.inh_weight

            for i in range(len(self.neurons)):
                self.neurons[i].u[self.neurons[i].current_time] += self.spikes_effect[t][i]
```

Note: When calculating spikes, the neuron's delay determines its impact.

Once the "action" function is defined, we can plot the potential and spike raster of the neurons.

EXAMPLES

- **Example 1: Connection Between Two Excitatory Neurons with Two Different Currents**

First, define two models of current, which could be two different constant currents. Then, create two excitatory neurons and assign each one of the currents.

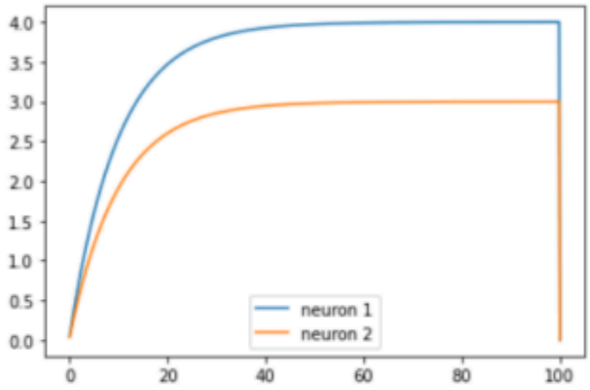
```
In [9]: i1 = lambda x: 4
        i2 = lambda x: 3
        neuron1 = Neuron(i_func=i1)
        neuron2 = Neoron(i_func=i2)
```

Next, pass these two neurons into the class for two-neuron connections, and call the action function to form the connection.

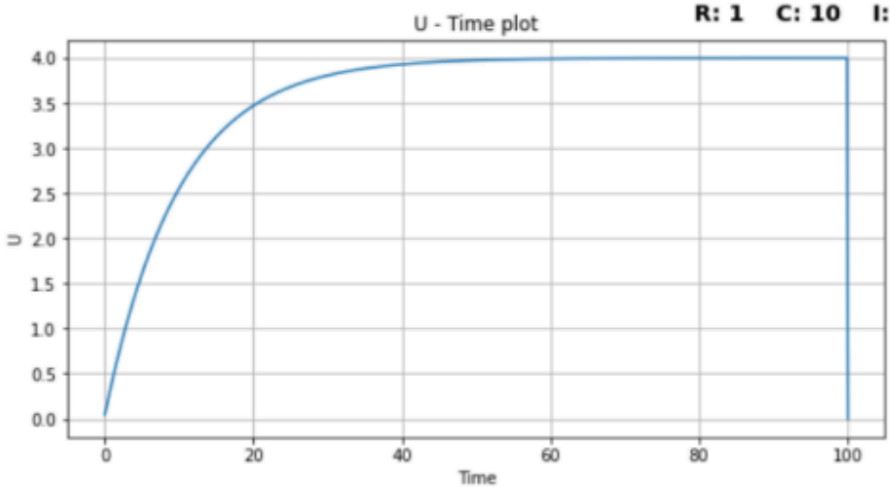
```
neurons = [neuron1, neuron2]
connections = [[1], [0]]
first_neuron_group = First_neuron_group(neurons, connections)
neurons_group.action()
```

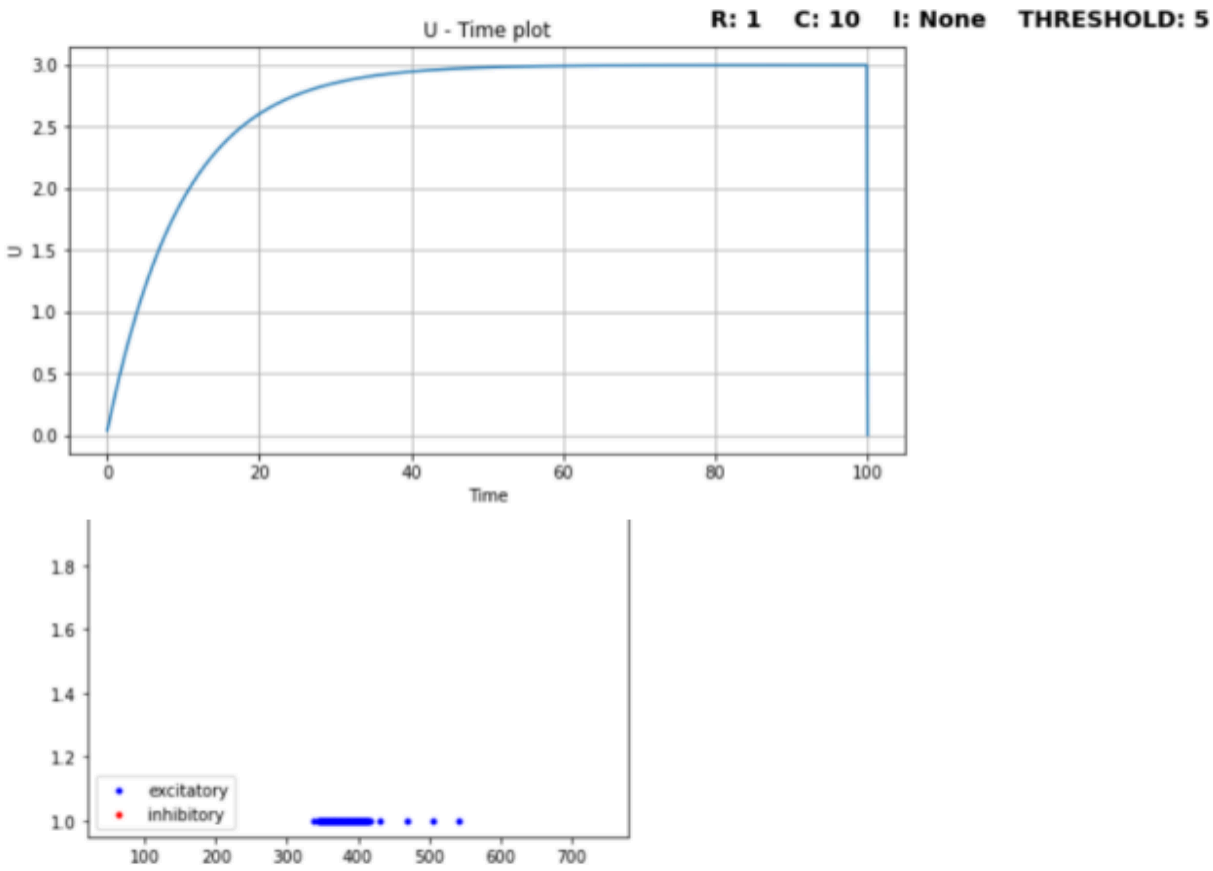
Finally, use the plotting functions to visualize the relevant graphs.

```
In [10]: first_neuron_group.neurons_u_plot()
         first_neuron_group.raster_plot()
         neuron1.plot_U_t()
         neuron2.plot_U_t()
```



R: 1 C: 10 I: None THRESHOLD: 5





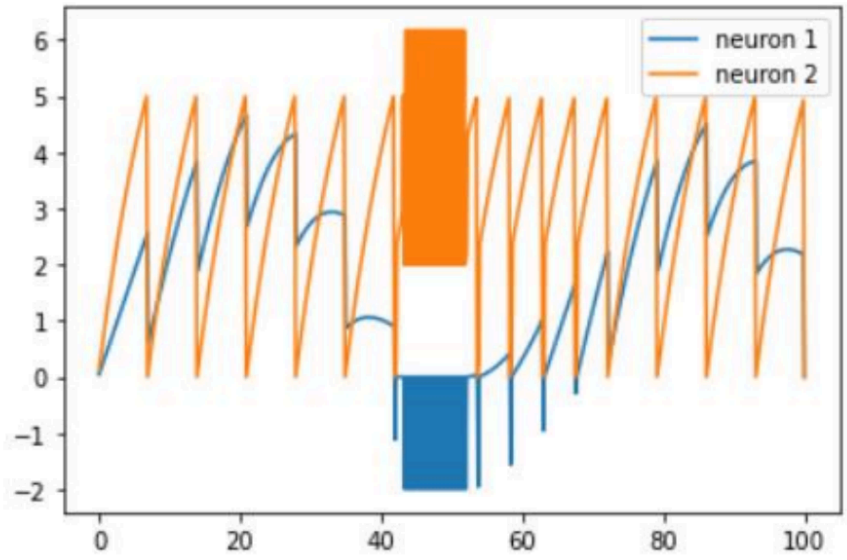
Example 2: Connection Between an Excitatory and an Inhibitory Neuron

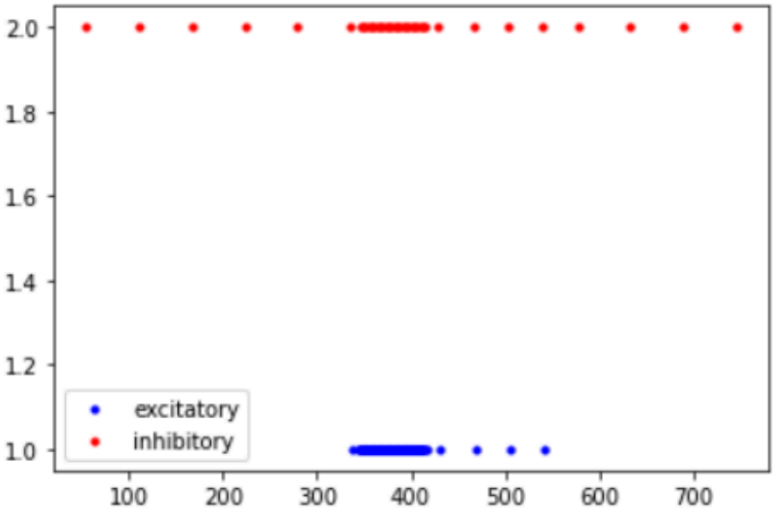
As in the previous example, define two neurons: one inhibitory and one excitatory with different currents.

```
i1 = lambda x: 4 * (math.sin(x/5) + 0.3)
i2 = lambda x: 10
neuron1 = neuron(i_func=i1)
neuron2 = neuron(i_func=i2 , neuron_type='inhibitory')
```

Then, pass them into the connection class, call the action function, and plot the relevant graphs.

```
neurons = [neuron1, neuron2]
connections = [[1], [0]]
first_neuron_group = First_neuron_group(neurons, connections)
first_neuron_group.action()
```





Example 3: Connection Between Two Inhibitory Neurons

First, define the two inhibitory neurons and their respective currents.

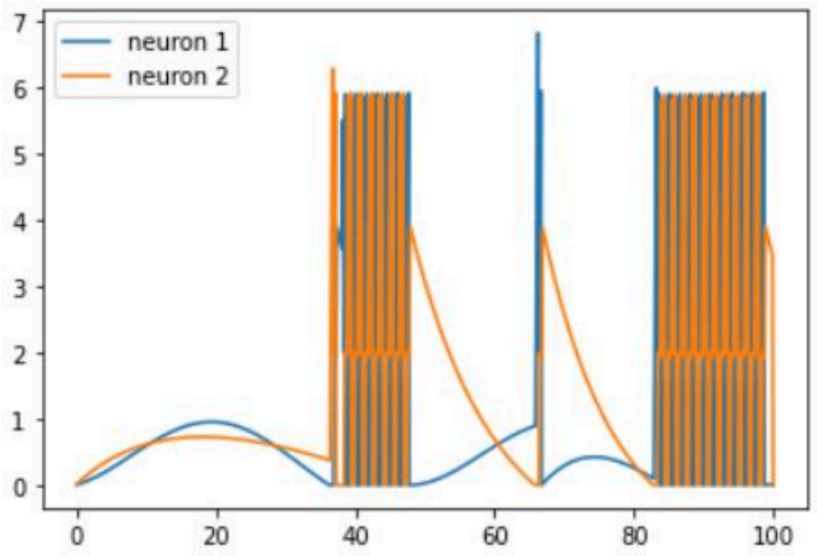
```
i1 = lambda x: (math.sin(x/8) + 0.3)
i2 = lambda x: (math.cos(x/24))
neuron1 = neuron(i_func=i1, neuron_type='inhibitory')
neuron2 = neuron(i_func=i2, neuron_type='inhibitory')
```

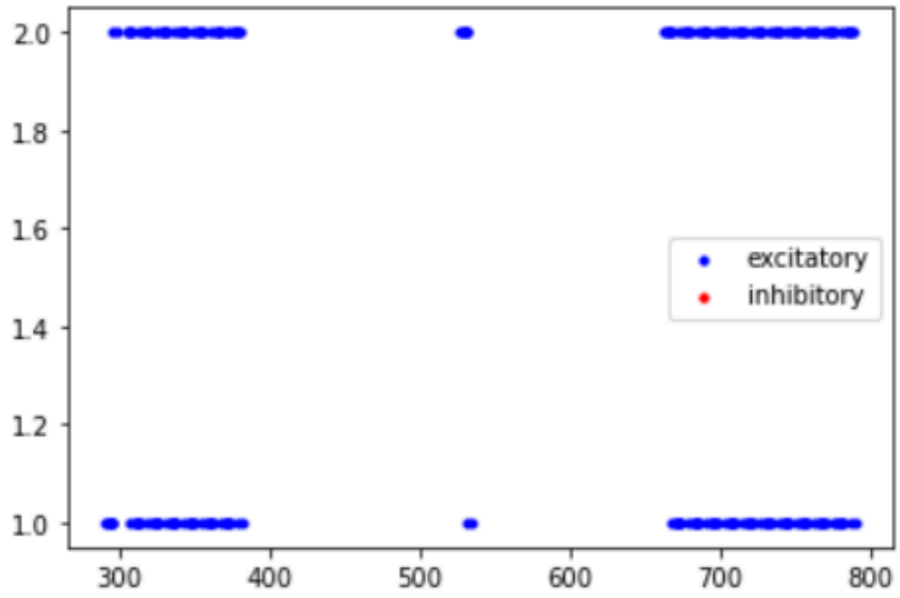
Then, pass them into the action function from the two-neuron connection class to form the connection.

```
neurons = [neuron1, neuron2]
connections = [[1], [0]]
first_neuron_group = First_neuron_group(neurons, connections)
first_neuron_group.action()
```

Finally, plot the relevant graphs.

```
first_neuron_group.u_plot()
first_neuron_group.raster_plot()
```





As observed, when the excitatory neuron spikes, the potential of the neurons connected to it increases. The exact opposite happens for the inhibitory neuron, where the potential decreases upon spiking.

Section 2: Examining the Connection Between Multiple Neurons

In this section, we first take as input the variables relevant to this exercise, including the input neurons, their connections (represented by a list), the number of neurons, and the probability of connections between them.

```
In [76]: class second_neuron_groups:
             def __init__(neurons, neurons_count, ex_count, inh_count, ex_prob, inh_prob, i = 0):
                 self.neurons = neurons
                 self.neurons_count = neurons_count
                 self.inh_count = in_count
```

Then, we define the main function of this class, which is the action function for connecting multiple neurons.

```
def action(self):
    counter = 0
    ex_conn_count = math.ceil(neurons_count * ex_prob)
    inh_conn_count = math.ceil(neurons_count * inh_prob)
    for i in range(self.inh_count):
        self.population.append(Neuron(neuron_type = "inhibitory"))
        counter += 1
    for j in range(counter):
        neuron = neuron(neuron_type="inhibitory")
        neurons.append(neuron)
        connections.append(random.sample(range(neurons_count), inh_conn_count))
    for i in range(self.ex_count):
        self.population.append(Neuron(neuron_type = "excitatory"))
        counter += 1
    for k in range(counter):
        neuron = neuron(neuron_type="excitatory")
        neurons.append(neuron)
        connections.append(random.sample(range(neurons_count), ex_conn_count))
    neurons_group = First_neuron_group(neurons, connections)
    return neurons_group
```

Note: We build on the previous class for two-neuron connections in this section.

EXAMPLES

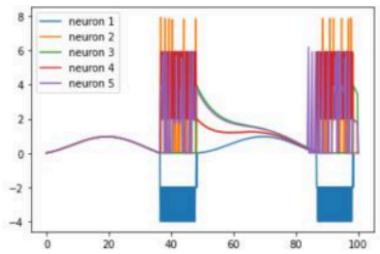
- **Example 1**

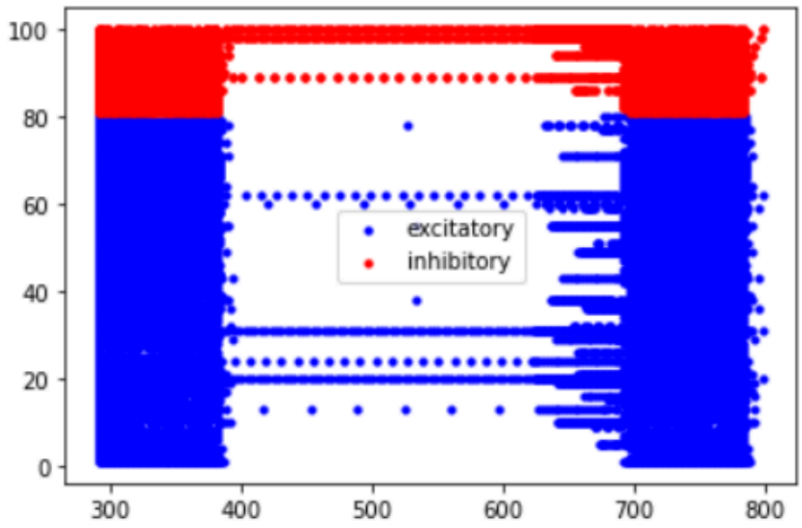
  Define eight excitatory neurons and two inhibitory neurons so that they all affect each other. Also, create a list of various currents to assign to the neurons.

```
In [40]: Is = []
         for i in range(100):
           I = lambda x: math.sin(x/8) + 0.3
           Is.append(I)
```

  After defining the neurons and passing them to the class, call the action function, then plot the relevant graphs for the model.

```
second_neuron_groups = Second_neuron_groups(100, 80, 20, 0.01, 0.02, Is)
second_neuron_groups.action()
```


neuron 1
neuron 2
neuron 3
neuron 4
neuron 5


excitatory
inhibitory

Section 3: Examining the Connection Between Multiple Neuron Populations

In this section, we create a class that includes the input neurons, their connections, the spike weight of each neuron (inhibitory and excitatory), the spike delay, and the threshold potential of the neurons.

```
class Third_neuron_groups:
    def __init__(self, neurons, connections, ex_weight = 2, inh_weight = 2, delay = 1, threshold = 20, effect = 2,
                 counter = 800):
            self.neurons = neurons
            for i in neurons:
                self.neroun_action.append(i.start())
            self.conections = conections
            self.ex_weight = ex_weight
            self.inh_weight = inh_weight
            self.delay = delay
            self.counter = counter
            self.actions = []
            self.spikes = []
            self.ex_spikes_times = []
            self.ex_spikes = []
            self.inh_spikes_times = []
            self.inh_spikes = []
            self.spikes_effect = []
            self.connected_neuron_groups = []
            self.threshold = threshold
            self.effect = effect
```

In the next step, we define the action function for connecting multiple neuron populations.

```
    def action(self):
            length = len(self.neurons)
            self.effect = [[0] * length for i in range(self.counter)]
            for t in range(self.counter):
                  for i in range(len(self.actions)):
                        action_info = next(self.actions[i])
                        if action_info == True :
                              for j in self.connections[i]:
                                    if self.neurons[i].neuron_type == "excitatory":
                                          self.ex_spikes.append(i + 1)
                                          self.ex_spikes_time.append(t)
                                          if t+self.delay < self.counter:
                                                self.effect[t + self.delay][j] += self.ex_weight

                                    if self.neurons[i].neuron_type == "inhibitory":
                                          self.inh_spikes.append(i + 1)
                                          self.inh_spikes_time.append(t)
                                          if t+self.delay < self.counter:
                                                self.effect[t+self.delay][j] += self.inh_weight
                  for i in range(len(self.neurons)):
                        self.neurons[i].u[self.neurons[i].current_time] += self.effect[t][i]
```

Additionally, we use a function called `Connection`, which, when called, connects a group of input neurons to other neuron populations.

```
    def connect(self, neuron_group):
                self.connected_neuron_groups.append(neuron_group)
```

Finally, we plot the potential and spike raster graphs for this class.

```
    def neurons_u_plot(self, neurons_count=5):
            for i in range(min(neurons_count, len(self.neurons))):
                plt.plot(list(map(lambda j: j * self.neurons[i].dt, range(len(self.neurons[i].u)))), self.neurons[i].u)


    def raster_plot(self):
            plt.scatter(self.excitatory_spikes_time, self.excitatory_spikes, color = "red", s = 10)
            plt.scatter(self.inhibitory_spikes_time, self.inhibitory_spikes, color = "blue", s = 10)
            plt.legend(["excitatory", "inhibitory"])
```

EXAMPLES

- **Example 1**

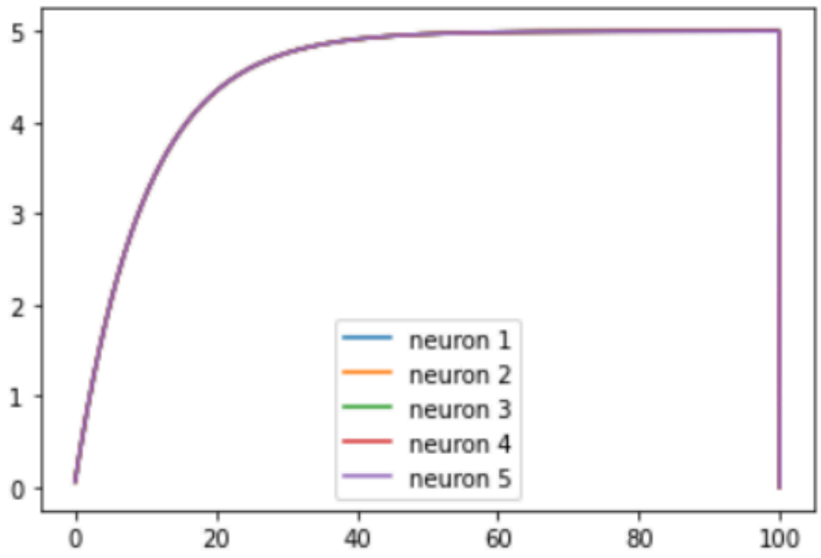
  Define three neuron populations, each consisting of 10 neurons (one inhibitory and two excitatory). Create a list of input currents and assign them to the relevant neurons.

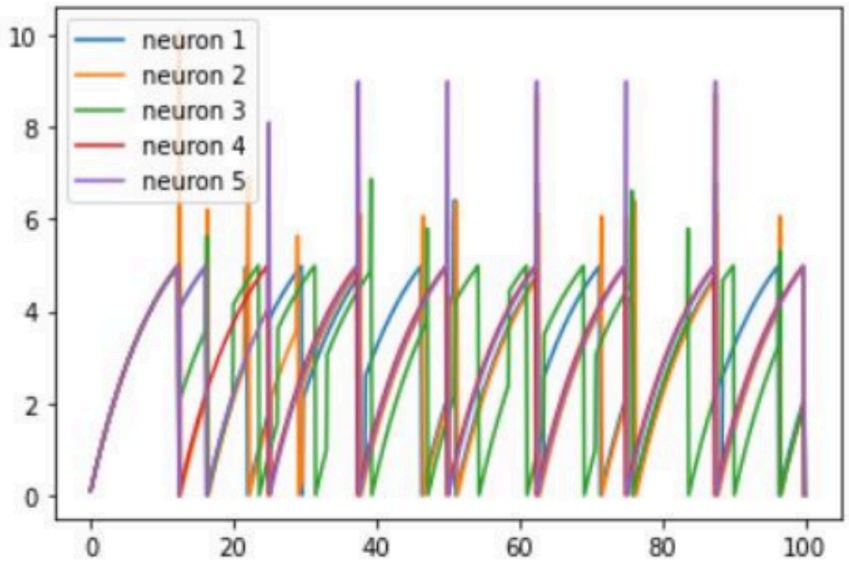
```
ex1_Is = []
for i in range(100):
  I = lambda x: 3*i
  ex1_Is.append(I)
ex2_Is = []
for i in range(100):
  I = lambda x: random.random() * 5.5
  ex2_Is.append(I)
inh_Is = []
for i in range(100):
  I = lambda x: 3
  inh_Is.append(I)
```

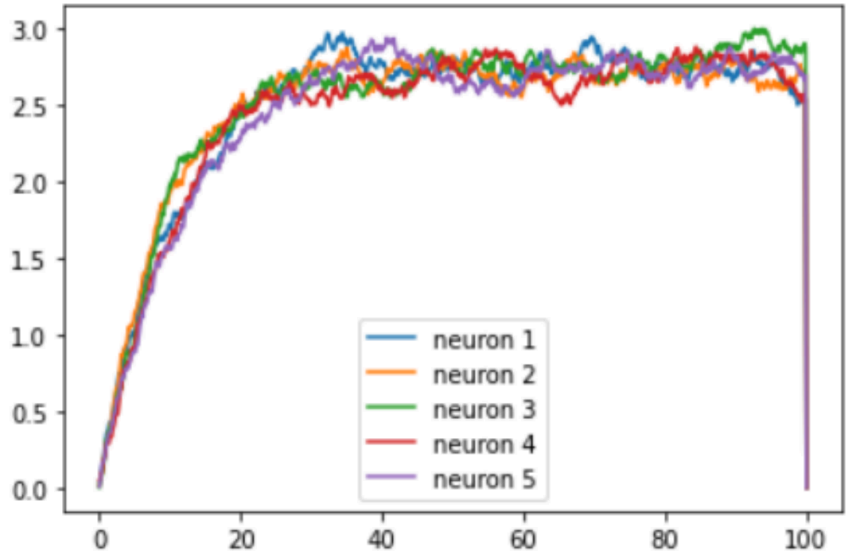
Then, implement the neuron populations, call the action function to create the connections between them, and plot the relevant graphs.

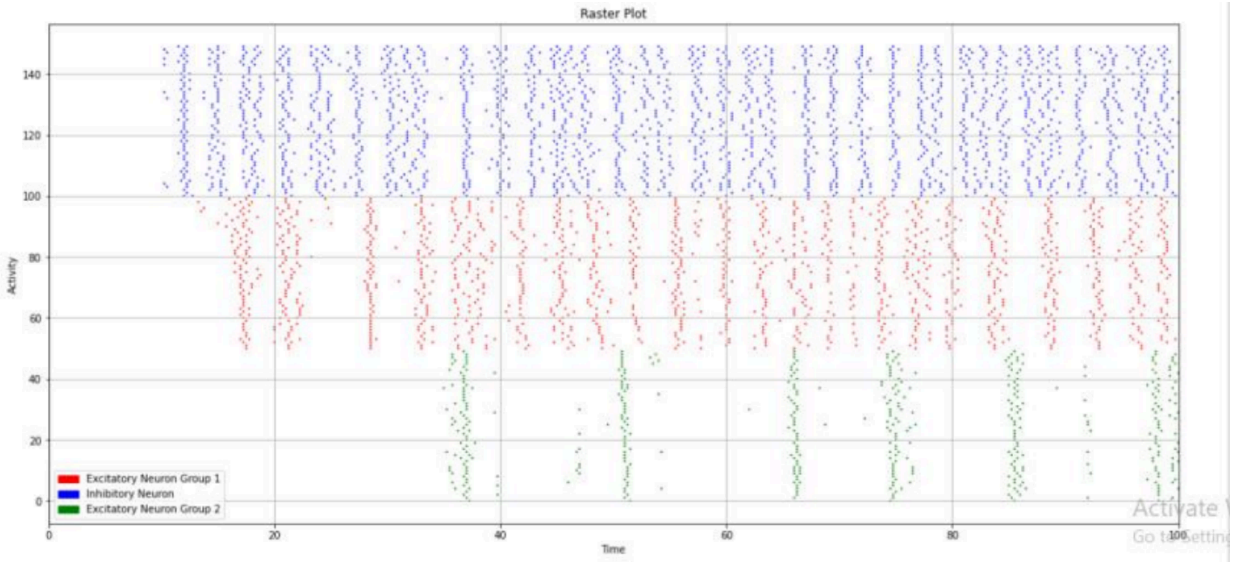
```
neuron_group1 = Third_neuron_groups(100, 100, 0, 0.01, 0, ex1_Is)
neuron_group2 = Third_neuron_groups(100, 100, 0, 0.01, 0, exc_I_2)
neuron_group3 = Third_neuron_groups(100, 0, 100, 0, 0.1, inh_Is)
neuron_group1.connect(neuron_group3)
neuron_group2.connect(neuron_group3)
neuron_group1.action()
neuron_group2.action()
neuron_group3.action()
```

Then, at this stage, we include the functions to plot the corresponding graphs which the resulting graphs will be as follows:




neuron 1
neuron 2
neuron 3
neuron 4
neuron 5
10
8
6
4
2
0
0
20
40
60
80
100


neuron 1
neuron 2
neuron 3
neuron 4
neuron 5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100


Raster Plot
Activity
Time
Excitatory Neuron Group 1
Inhibitory Neuron
Excitatory Neuron Group 2
0
20
40
60
80
100
120
140