Assignment 1

Q2An experimenter recorded a SINGLE cortical neuron's activity from the part of the brain processing visual information as the organism viewed a visual stimuli on the screen. The collected dataset is attached herewith: 'Data1_NDM.mat'. It is a Three-dimensional MATLAB array. Note the following.

● Dimension 1 = Stimulus orientations (45 degree to 202.5 degree with increments in step size of 22.5 degree visual angle).

● Dimension 2 = Time (sampling frequency of the neuronal activity = 1000 Hz); Total neuronal recording duration = 3.5 seconds; Time = 0 (stimulus not moving);Time = 500 (stimulus moving) ; Time = 2500 (stimulus off);

● Dimension 3 = Trials or stimulus repetitions

● Value 1 = action potential (spike) fired by the neuron; Value 0 = no action potential fired by the neuron

Now solve the following. Insert a figure (wherever required) and paste the MATLAB/Python/R code for the same. Any figure must provide all information necessary to interpret it including axes labels, captions/legends (see Fig.1 of the attached paper for a sample; simple figure titles as captions are not enough).

[links about the 3-dimensional array in MATLAB and importing MATLAB data arrays into Python and R

� https://in.mathworks.com/help/matlab/math/multidimensional-arrays.html

� https://in.mathworks.com/help/matlab/matlab_external/matlab-arrays-as-python-variables.html

� https://stackoverflow.com/questions/11671883/importing-an-array-from-matlab-into-r]

A. Create a Raster plot of the neuron for ALL EIGHT orientations of the stimulus and mark the onset of stimulus movement and offset of the stimulus by a vertical green and red line respectively on the same individual subplots. Mark the spikes (action potentials) with solid black circles. [5 marks]

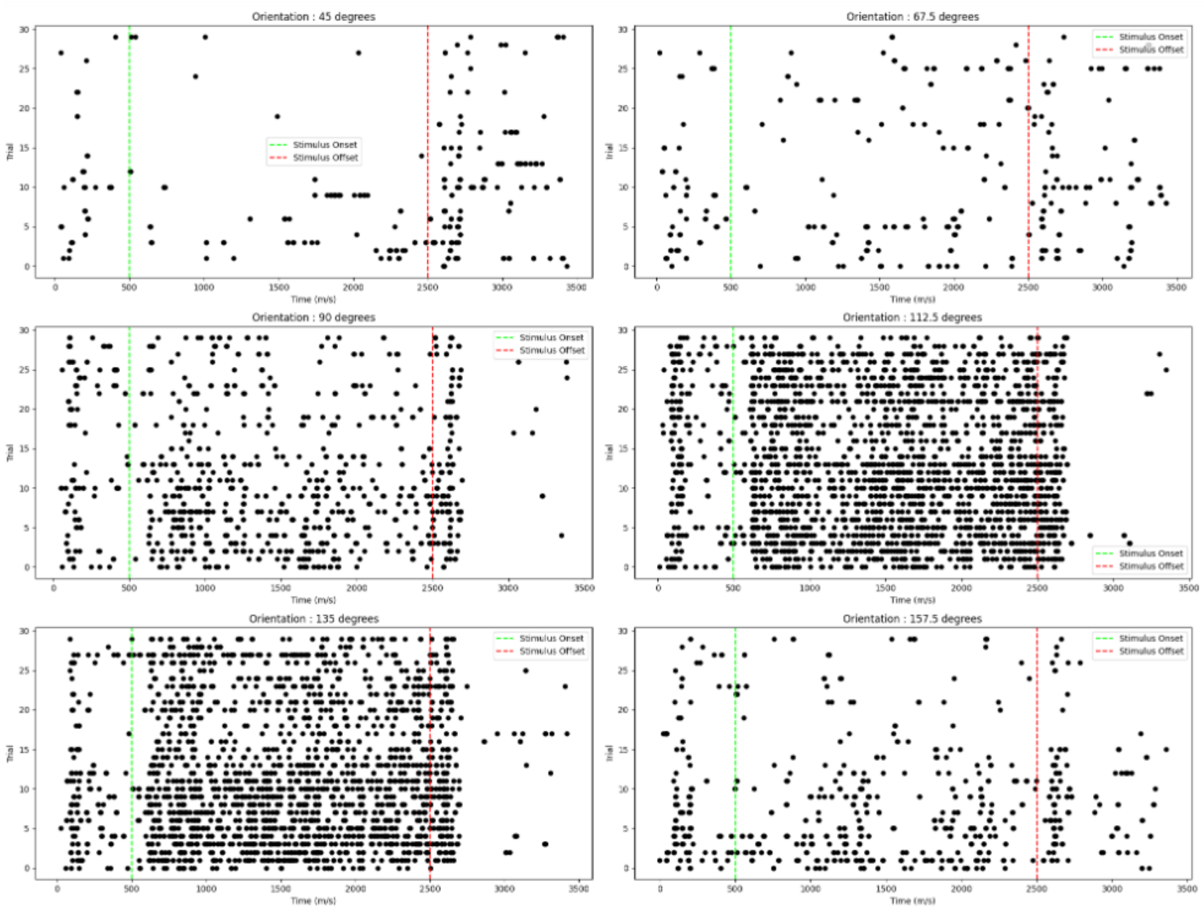
Hint: Create a larger figure with eight subplots (positioned as 4 rows x 2 columns);

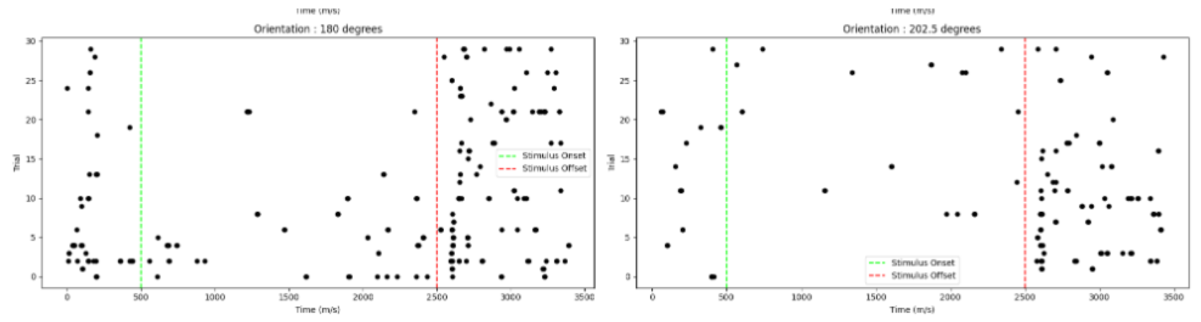
Indicate the stimulus orientation on top of each subplot as subplot title

Code is:

```
#!/usr/bin/env python
# coding: utf-8

# In[2]:


import scipy.io
import matplotlib.pyplot as plt
import numpy as np


# In[3]:


data = scipy.io.loadmat("Data1_NDM.mat")
data


# In[7]:


spikes = data['Data1_NDM']

# taking orientations
Orientations = spikes.shape[0]
OrentationDegree = [45,67.5,90,112.5,135,157.5,180,202.5]
# taking timepoints
TimePoints = spikes.shape[1]

# Taking trials
Trial = spikes.shape[2]

fig, axis = plt.subplots(4, 2, figsize=(20, 20))

for i in range(Orientations):

    dataS = spikes[i, :, :]
    xDimension = i //2
    yDimension = i%2
    # Now here we will plot raster plot
    iTrial = 0  # Initialize the trial index

    while iTrial < Trial:
        spike = [g for g in range(TimePoints) if dataS[g, iTrial] == 1]
        axis[xDimension, yDimension].plot(spike, [iTrial] * len(spike), 'ko',
markersize=5)

        iTrial += 1

    # Mark the onset of stimulus movement
    axis[xDimension, yDimension].axvline(x=500, color='#00FF00', linestyle='--
', label='Stimulus Onset')

    # Mark the offset of stimulus movement
    axis[xDimension, yDimension].axvline(x=2500, color='#FF0000', linestyle='-
-', label='Stimulus Offset')

    # #00FF00 - Green
    # #FF0000- Red

    # Setting  the title
    axis[xDimension, yDimension].set_title(f'Orientation :
{OrentationDegree[i]} degrees')

    # Set the x-axis as  Time (m/s)
    # and y-axis as Trails labels
    axis[xDimension, yDimension].set_xlabel('Time (m/s)')
    axis[xDimension, yDimension].set_ylabel('Trial')


    axis[xDimension, yDimension].legend()

# Adjust subplot layout
plt.tight_layout()

# Show the plot
plt.show()


# In[ ]:
```

OutPut is:





B. Create a Peri Stimulus Time Plot of the neuron for ALL EIGHT orientations of the stimulus and mark the onset of stimulus movement and offset of the stimulus by a vertical green and red line respectively on the same individual subplots. Before computing the histogram, smooth the data for each subplot over a time window of 61 ms. [5 marks]

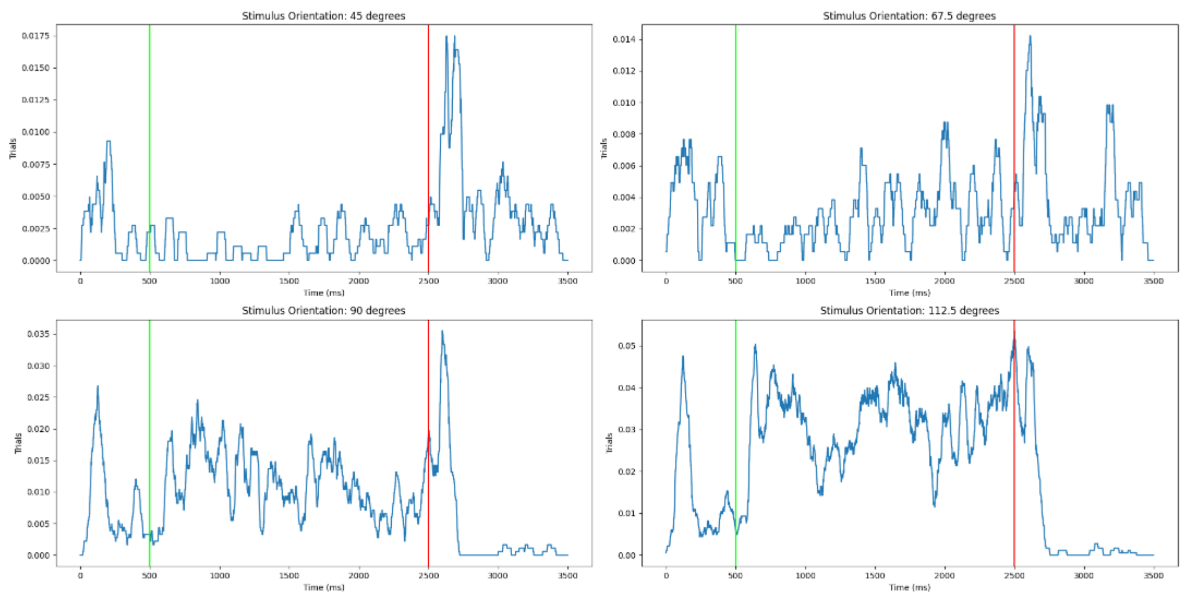
Hint: Create a larger figure with eight subplots (positioned as 4 rows x 2 columns); Smooth the data by moving average method; A line plot would suffice and depict the trend; Indicate the stimulus orientation on top of each subplot as subplot title

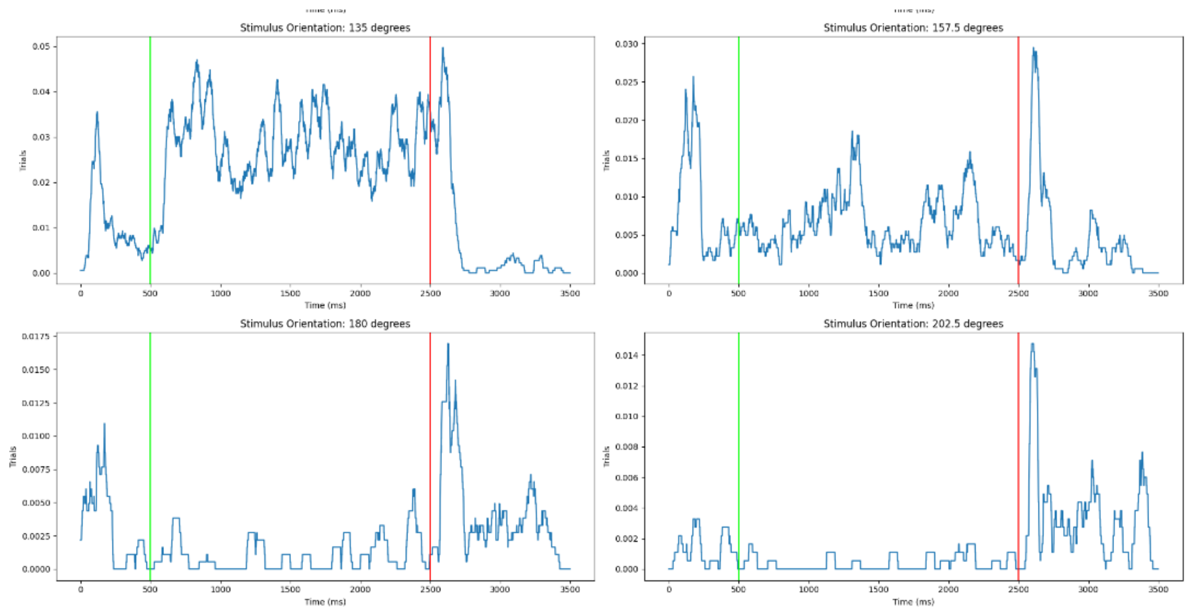
Code is:

```python
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import scipy.io
import numpy as np
import matplotlib.pyplot as plt


# In[2]:


data = scipy.io.loadmat("Data1_NDM.mat")


# In[3]:


OrentationDegree = [45,67.5,90,112.5,135,157.5,180,202.5]
TimeWindow = 61

fig, axis = plt.subplots(4, 2, figsize=(20, 20))

for i, axis in enumerate(axis.flat):

    orientation_data = data['Data1_NDM'][i][:][:]

    SmoothedData = np.convolve(orientation_data.mean(axis=1),
np.ones(TimeWindow)/TimeWindow, mode='same')

    # Plot the smoothed data
    axis.plot(SmoothedData)

    axis.axvline(x=500, color='#00FF00')  # Onset of stimulus movement
    axis.axvline(x=2500, color='#FF0000')  # Offset of stimulus

    # #00FF00 - Green
    # #FF0000- Red
    axis.set_xlabel('Time (ms)')
    axis.set_ylabel('Trials')
    axis.set_title(f'Stimulus Orientation: {OrentationDegree[i]} degrees')

# Display the figure
plt.tight_layout()
plt.show()
```

```
# In[ ]:
```

Output is:





C. Create a figure representing the Tuning Curve of the neuron from the average firing rate of the neuron (between 600 – 2500 ms and all trials) for each orientation.

Computationally calculate the 'preferred orientation' of the neuron. Report the same on

the title of the plot and mark it on the plot. [5+5 marks]

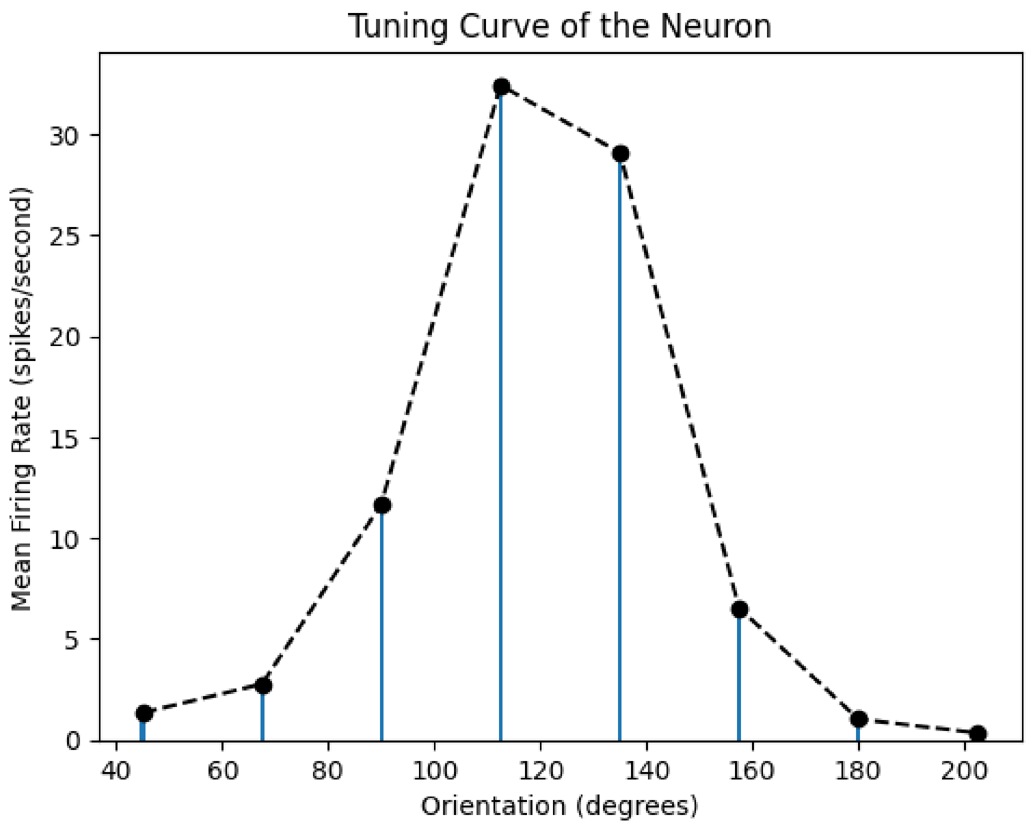
A.

Code :

```
data = sio.loadmat('Data1_NDM.mat')['Data1_NDM']
data


# In[6]:


#Q2c-1st Part

#  As given in the question we define the time window for analysis Start Time
is 600 ms and EndTime is  2500 ms
StartTime = 600
EndTime = 2500

Orientations = data.shape[0]
NumTimePoints = data.shape[1]

# Stimulus orientations
OrentationDegree = [45,67.5,90,112.5,135,157.5,180,202.5]

NumTrials = data.shape[2]

FiringRates = np.zeros((Orientations, NumTrials))

i = 0
while i < NumTrials:
    orien = 0
    while orien < Orientations:

        spike = data[orien, :, i]

        # Calculate firing rate by counting spikes within the time window
        FiringRate = np.sum(spike[StartTime:EndTime]) / ((EndTime - StartTime)
/ 1000)

        # Storing firing rate
        FiringRates[orien, i] = FiringRate

        orien += 1
    i += 1

# Now calculating the mean firing rate across all trials for each orientation
```

```
mean_firing_rates = np.mean(FiringRates, axis=1)

plt.figure()
plt.bar(OrentationDegree, mean_firing_rates)
plt.plot(OrentationDegree, mean_firing_rates,'ko--')
plt.xlabel('Orientation (degrees)')
plt.ylabel('Mean Firing Rate (spikes/second)')
plt.title('Tuning Curve of the Neuron')
plt.show()


# In[13]:
```

Output :

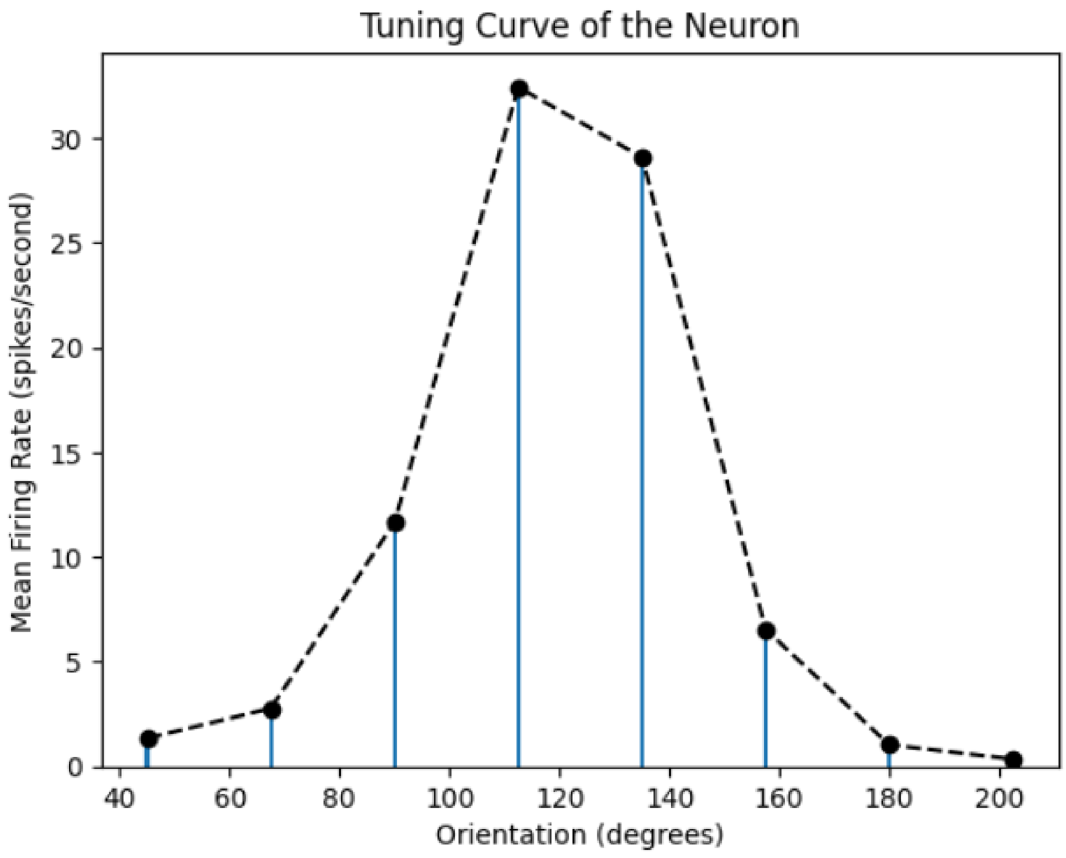


B.

Code:

```
#Q2c- 2nd Part


# Stimulus orientations
OrentationDegree = [45,67.5,90,112.5,135,157.5,180,202.5]
```

```python
TimeWindow = range(600, 2500)

#  To store the average firing rates
AverageFiringRates = []

i = 0

while i < len(OrentationDegree):
    OrientationData = data[i, TimeWindow, :]

    # Now to calculate average firing rate for this orientation
    average_firing_rate = OrientationData.mean()
    AverageFiringRates.append(average_firing_rate)

    i += 1

# Now we calcute the preferred orientation
preferred_orientation = OrentationDegree[np.argmax(AverageFiringRates)]

plt.figure(figsize=(10, 6))

# Ploting the tuning curve
plt.plot(OrentationDegree, AverageFiringRates, marker='o')

plt.axvline(x=preferred_orientation, color='#FF0000', linestyle='--')
# #FF0000- Red

plt.xlabel('Stimulus Orientation (degrees)')
plt.ylabel('Average Firing Rate(Spikes/s)')
plt.title(f'Tuning Curve of Neuron of the Preferred Orientation:
{preferred_orientation} degrees)')

plt.show()


# In[ ]:




# In[ ]:
```



Whole Code:

```
#!/usr/bin/env python
# coding: utf-8

# In[12]:


import numpy as np
import scipy.io as sio
import matplotlib.pyplot as plt


# In[13]:


data = sio.loadmat('Data1_NDM.mat')['Data1_NDM']
data


# In[10]:

```

```
#Q2c-1st Part

#  As given in the question we define the time window for analysis Start Time
is 600 ms and EndTime is  2500 ms
StartTime = 600
EndTime = 2500

Orientations = data.shape[0]
NumTimePoints = data.shape[1]

# Stimulus orientations
OrentationDegree = [45,67.5,90,112.5,135,157.5,180,202.5]

NumTrials = data.shape[2]

FiringRates = np.zeros((Orientations, NumTrials))

i = 0
while i < NumTrials:
    orien = 0
    while orien < Orientations:

        spike = data[orien, :, i]

        # Calculate firing rate by counting spikes within the time window
        FiringRate = np.sum(spike[StartTime:EndTime]) / ((EndTime - StartTime)
/ 1000)

        # Storing firing rate
        FiringRates[orien, i] = FiringRate

        orien += 1
    i += 1

# Now calculating the mean firing rate across all trials for each orientation
mean_firing_rates = np.mean(FiringRates, axis=1)

plt.figure()
plt.bar(OrentationDegree, mean_firing_rates)
plt.plot(OrentationDegree, mean_firing_rates,'ko--')
plt.xlabel('Orientation (degrees)')
plt.ylabel('Mean Firing Rate (spikes/second)')
plt.title('Tuning Curve of the Neuron')
plt.show()


# In[16]:
```

```python
#Q2c- 2nd Part


# Stimulus orientations
OrentationDegree = [45,67.5,90,112.5,135,157.5,180,202.5]
TimeWindow = range(600, 2500)

#  To store the average firing rates
AverageFiringRates = []

i = 0

while i < len(OrentationDegree):
    OrientationData = data[i, TimeWindow, :]

    # Now to calculate average firing rate for this orientation
    average_firing_rate = OrientationData.mean()
    AverageFiringRates.append(average_firing_rate)

    i += 1

# Now we calcute the preferred orientation
preferred_orientation = OrentationDegree[np.argmax(AverageFiringRates)]

plt.figure(figsize=(10, 6))

# Ploting the tuning curve
plt.plot(OrentationDegree, AverageFiringRates, marker='o')

plt.axvline(x=preferred_orientation, color='#FF0000', linestyle='--')
# #FF0000- Red

plt.xlabel('Stimulus Orientation (degrees)')
plt.ylabel('Average Firing Rate(Spikes/s)')
plt.title(f'Tuning Curve of Neuron of the Preferred Orientation:
{preferred_orientation} degrees)')

plt.show()


# In[ ]:




# In[ ]:
```

[redacted]