I. INTRODUCTION

This project examines neuronal activity in the cerebellum during a classical conditioning trial. The question I will be investigating is how the rates of learning an association (acquisition) and rates of extinguishing the association (extinction) differ in cerebellar cells. I will be using datasets generated from a computational simulation developed by the lab of Dr. Michael Mauk. I chose this topic because I am interested in learning how a computational simulation can model the activities of neurons. Moreover, I'm interested to see how different cerebellar neurons act while it undergoes learning.

The simulation models the behavior of different neurons in the cerebellum during an eyelid conditioning trial, which is one of the widely used classical conditioning methods. The data I gathered from the simulation are raster files of spike activities from 2 different type of neurons (Purkinje cell, Deep Nucleus cell) over time, as the assocation is learned and extinguished. Each trial that the neurons undergo is either strengthening the learned association or weakening it. In my experiment, the first 500 trials were designed to strengthen the association (acquisition) and the following 500 trials were designed to weaken the association (extinction). In summary, my dataset includes activity of 2 different types of cerebellar neurons over 1000 trials. Within each trial, spike activity of all neurons are recorded for a certain period of time whose units will be in milliseconds.

With this data, I will be able to analyze the neuron activity and examine the rates of acquisition and extinction in the cerebellum. This project will allow me to gain a deeper understanding of computational simulation of neurons and answer the question of whether the pattern of acquisition and extinction shows a significant difference.

II. DATA ORGANIZATION

This section is where necessary libraries and datasets are imported. The initial datasets are in a binary format, and are later reshaped according to how the information is organized. I will be using 10 different data files in total. They are as follows:

- Purkinje cell activity file while the interstimulus interval (ISI) is 250 ms, 500 ms, 750 ms, 1000 ms, and 1250 ms
- Deep Nucleus cell activity file while the interstimulus interval (ISI) is 250 ms, 500 ms, 750 ms, 1000 ms, and 1250 ms

Background Information:

- Interstimulus interval (ISI) is an important concept in classical conditioning, which is why I gathered cell activity data from various ISIs and will be comparing the result accordingly. ISI is the temporal interval between the two stimulus that the neuron has to learn to associate with one another. For instance, let's say a trial has the ISI of 250 ms. One stimulus will be presented after the initial 400 ms has passed. (This onset timing will be the same across all trials.) The first stimulus will last for the length of the ISI (250 ms). At the end of this 250 ms, the second stimulus will be presented to the brain. In the case where the ISI is 500 ms, the temporal interval between the two stimuls will be 500 ms.

- In eyelid conditioning, the first stimulus presented (conditioned stimulus) is an auditory stimulus, a simple tone. The following stimulus (unconditioned stimulus) is a physical puff to the eye that makes the eyelid blink. As the two stimuli are consistently presented after one another, neurons will learn the association and start responding when the conditioned stimulus is presented (conditioned response). When only the first stimulus is consistently presented without the second stimulus following, the learned association will gradually be extinguished.

In [1]:

```
# -------- Import Libraries -------- #

import numpy as np                                          #
this library is used for creating and manipulating arrays
import matplotlib.pyplot as plt                             #
this library is used for plotting graphs
import scipy.ndimage                                        #
this library is used when defining a function to smooth the
data
from sklearn.linear_model import LinearRegression          #
this library is used for statistical testing with Linear
Regression
import os                                                   #
this library is used for creating a gif
import imageio                                              #
this library is used for creating a gif
from PIL import Image                                       #
this library is used for creating a gif
from IPython.display import Image                           #
this library is used for displaying the gif I created
from scipy import stats                                     #
this library is used for paired t-test



# -------- Loading Temporary Data Files -------- #

# this is the list of different ISIs (i.e. interstimulus
intervals) from which neuron activity files will be gathered
and anaylzed
# this list is defined so that I can make a reference to this
list when importing files later in this block of code
filelist = [250, 500, 750, 1000, 1250]
```

```
# load Purkinje cell activity data as a temporary file from
each isi
tempfiles_pc = {}                          # create a dictionary
to store temporary data files of Purkinje cells
for isi in filelist:                       # loop through the
different ISIs that I prepared in the 'filelist' list above
    # code below loads and stores each file as a value in the
dictionary 'tempfiles_pc'
    # the key for each file will be the ISI number loaded
from the list 'filelist'
    tempfiles_pc[isi] = np.fromfile("data/PCraster"+str(isi),
dtype=np.ubyte)

# load Deep Nucleus cell activity data as a temporary file
from each isi
tempfiles_nc = {}                          # create a dictionary
to store temporary data files of Deep Nucleus cells
for isi in filelist:                       # loop through the
different ISIs that I prepared in the 'filelist' list above
    # code below loads and stores each file as a value in the
dictionary 'tempfiles_nc'
    # the key for each file will be the ISI number loaded
from the list 'filelist'
    tempfiles_nc[isi] = np.fromfile("data/NCraster"+str(isi),
dtype=np.ubyte)
```

The data I loaded are stacked as a one-dimensional array, while the information are stored in a three-dimensional structure. Therefore, I'm going to convert the temporary data files into true raster data files by reshaping.

In [4]:

```
# -------- Converting Temporary Data Files to Raster Files --
------ #

NUM_PC_CELLS = 32   # number of Purkinje cells in the
simulation (Purkinje cell = PC)
NUM_NC_CELLS = 8    # number of Deep Nucleus cells in the
simulation (Deep Nucleus cell = NC)
```

```
# reshape each Purkinje cell data in 'tempfiles_pc' into raster data
rasterfiles_pc = {}             # create a new dictionary to store reshaped Purkinje cell data
for isi in filelist:            # loop through the different ISIs that I prepraed in the 'filelist' list above
    # code below reshapes each file in 'tempfiles_pc' and stores it as a new file in 'rasterfiles_pc'
    # the key for each file will be the ISI number loaded from the lsit 'filelist'
    rasterfiles_pc[isi] = tempfiles_pc[isi].reshape(NUM_PC_CELLS, 1000, 400+isi+400)
    # in each of the new reshaped file, the spike data is organized as 32 (= number of Purkinje cells) sets of 1000 (= number of trials) rows and 400*2+j (= trial length in ms) columns
    # there are 1000 trials because the first 500 will be acquisition trials and the following 500 will be extinction trials
    # trial length is calculated as 400+isi+400 because:
        # onset timing (400 ms) + interstimulus interval ('isi' ms) + 400 ms extra after both stimulus is presented

# reshape each Deep Nucleus cell data in 'tempfiles_nc' into raster data
rasterfiles_nc = {}             # create a new dictionary to store reshaped Deep Nucleus cell data
for isi in filelist:            # loop through the different ISIs that I prepraed in the 'filelist' list above
    # code below reshapes each file in 'tempfiles_nc' and stores it as a new file in 'rasterfiles_nc'
    # the key for each file will be the ISI number loaded from the lsit 'filelist'
    rasterfiles_nc[isi] = tempfiles_nc[isi].reshape(NUM_NC_CELLS, 1000, 400+isi+400)
    # in each of the new reshaped file, the spike data is organized as 32 (= number of Purkinje cells) sets of 1000 (= number of trials) rows and 400*2+j (= trial length in ms)
```

```
columns
    # there are 1000 trials because the first 500 will be
acquisition trials and the following 500 will be extinction
trials
    # trial length is calculated as 400+isi+400 because:
        # onset timing (400 ms) + interstimulus interval
('isi' ms) + 400 ms extra after both stimulus is presented
```

Then, I combine the raster data from every cell during a given trial number and time point so that I can plot a Peristimulus Time Histogram (PSTH) of neuron activity.

In [5]:

```
# -------- Defining Smoothing Functions -------- #

# codes below define half-guassian smoothing function
# this is used for smoothing the psth graph
# psth graph has to be smoothed because it has a lot of noise
to start with
# details of this function is not necessary for my project
def halfgaussian_kernel1d(sigma, radius):
    sigma2 = sigma * sigma
    x = np.arange(0, radius+1)
    phi_x = np.exp(-0.5 / sigma2 * x ** 2)
    phi_x = phi_x / phi_x.sum()
    return phi_x
def halfgaussian_filter1d(input, sigma, axis=-1, output=None,
                          mode="constant", cval=0.0,
truncate=4.0):
    sd = float(sigma)
    lw = int(truncate * sd + 0.5)
    weights = halfgaussian_kernel1d(sigma, lw)
    origin = -lw // 2
    return scipy.ndimage.convolve1d(input, weights, axis,
output, mode, cval, origin)



# -------- Defining a Function to Convert Raster Data to PSTH
Data -------- #

def raster_to_psth(data):
```

```
    '''
    This function converts the input data from raster format
to psth format
    '''
    xyz = data.shape                          # x is total
number of cells, y is total number of trials, z is total
number of time points
    psthdata = np.zeros((xyz[1],xyz[2]))    # 'psthdata' is a
numpy array with row as the num of trials and column as the
num of time points
    for i in range(0,xyz[0]):                 # loop through
all cells in the input data (xyz[0] is the total number of
cells)
        for j in range(0,xyz[1]):             # loop through
all trials in the input data (xyz[1] is the total number of
trials)
            for k in range(0,xyz[2]):         # loop through
all time points (xyz[2] is the total number of time points)
                # at a given trial number 'j' and time point
'k', sum up all the spike activity of existing cells and
store it as a single file
                psthdata[j, k] = psthdata[j, k] + data[i, j,
k]
    # use the half-guassian smoothiing function I defined
above to reduce noise in the data
    psthdata_flt = np.zeros((xyz[1],xyz[2]))        # this is
a new numpy array to stored filtered data points from
'psthdata'
    for i in range(0,xyz[1]):                             # loop
through all trials in the data (xyz[1] is the total number of
trials)
        # at a given trial number 'j', use half guassian
smoothing function to reduce noise and store it in the
'psthdata_flt' array
        psthdata_flt[i] = halfgaussian_filter1d(psthdata[i],
sigma=100)
    return psthdata_flt                       # return smoothed
psth data
```

```
# -------- Converting Raster Data to PSTH Data -------- #

# convert each Purkinje cell raster data into psth data
psthfiles_pc = {}                           # create a new
dictionary to store Purkinje cell PSTH data
for isi in filelist:                        # loop through
the different ISIs that I prepraed in the 'filelist' list
above
    # code below converts each file in 'rasterfiles_pc' and
stores it as a new file in 'psthfiles_pc'
    # the key for each file will be the ISI number loaded
from the lsit 'filelist'
    psthfiles_pc[isi] = raster_to_psth(rasterfiles_pc[isi])

# convert each Deep Nucleus cell raster data into psth data
psthfiles_nc = {}                           # create a new
dictionary to store Deep Nucleus cell PSTH data
for isi in filelist:                        # loop through
the different ISIs that I prepraed in the 'filelist' list
above
    # code below converts each file in 'rasterfiles_nc' and
stores it as a new file in 'psthfiles_nc'
    # the key for each file will be the ISI number loaded
from the lsit 'filelist'
    psthfiles_nc[isi] = raster_to_psth(rasterfiles_nc[isi])
```

III. DATA VISUALIZATION

FIGURE 1. Peristimulus Time Histogram (PSTH) of Neuron Activity

This is a PSTH graph showing neuron spike activity. The first graph is representing the activity of Deep Nucleus cells during the 500th trial, and the second graph is representing the activity of Purkinje cells during the same trial. The x-axis is time in milliseconds, while the y-axis is number of spikes. Dashed vertical lines are drawn to show the time points where two stimuli is presented to the neurons, one after another. The activity displayed was measured after 500 trials of acquisition, which means the neurons have learned the association between the two stimuli. In the two graphs, we can see that the cells both show a conditioned response to the stimuli. Deep Nucleus cells increase their spikes in response to the first stimulus (conditioned stimulus), whereas Purkinje cells decrease their spikes. This is because Purkinje cells inhibit the Deep Cerebellar Nuclei.

In [6]:

```
# -------- Defining a Function to Plot PSTH Graph -------- #

# codes below define a function that plots PSTH graph from
input data
# this is going to be a graph for one given trial (trial
number is set by the parameter 'trialnum')
def makepsthgraph(data, celltype, trialnum, isi, ymin, ymax):
    '''
    This function plots a PSTH graph from input data\n
        data = input PSTH file\n
        celltype = 'PC' for Purkinje cell or 'NC' for Deep
Nucleus cell\n
        trialnum = choose the trial number to plot the graph
for\n
        isi = specify the interstimulus interval of the input
PSTH data\n
        ymin = minimum y value in the axis plot\n
        ymax = maximum y value in the axis plot
    '''
    yz = data.shape
# y is total number of trials, z is total number of time
points
    plt.figure(figsize=(4,2))
    plt.xlim(0,yz[1])
# x axis starts at 0 and ends at total number of time points
(yz[1]), which will be 400+isi+400 as explained above
    plt.ylim(ymin, ymax)
# y axis limits are set by parameter of the function
    x = np.arange(0,yz[1])
# make a list of numbers from 0 to total time points (which
is going to be 400+isi+400) to create the x coordinates for
line graph

    # below are codes that plot a line graph for PSTH
        # 3 different plots are written in order to set
different colors for line graphs in different time points
        # time points where the first stimulus (conditioned
stimulus) is present has darker color
```

```
    # the graph until the first stimulus is presented (at 400
ms) has light color
    plt.plot(x[0:400],data[trialnum,0:400], color='#7B8387')
    # the graph between when the first stimulus is introduced
(at 400 ms) and the second stimulus is introduced (at 400+isi
ms) has dark color
    plt.plot(x[400:400+isi],data[trialnum,400:400+isi],
color='#37474F')
    # the graph after both stimuli are presented (at 400 +
isi ms) has light color
    plt.plot(x[400+isi:yz[1]],data[trialnum,400+isi:yz[1]],
color='#7B8387')

    # additional features of the graph
    plt.axvline(400, color='#BFBFBF', linestyle='dashed')
# plot a vertical line at the point where the conditioned
stimulus is presented (at 400 ms)
    plt.axvline(400+isi, color='#BFBFBF', linestyle='dashed')
# plot a vertical line at the point where the unconditioned
stimulus is presented (at 400+isi ms)
    plt.xlabel('time (ms)', fontsize=8)
# x-axis is time in milliseconds
    plt.ylabel('spike num', fontsize=8)
# y-axis is spike number (after smoothing)
    plt.xticks(fontsize=8)
    plt.yticks(fontsize=8)

    # graph title depends on the type of cell
    if celltype == 'NC':
# if the cell type is 'NC' (i.e. Deep Nucleus cell)
        plt.title('Deep Nucleus Cell Activity in Trial
#'+str(trialnum)+' (ISI = '+str(isi)+')', fontsize=8)
    elif celltype == 'PC':
# if the cell type is 'PC' (i.e. Purkinje cell)
        plt.title('Purkinje Cell Activity in Trial
#'+str(trialnum)+' (ISI = '+str(isi)+')', fontsize=8)
    else:
        print('[ERROR] Invalid cell type')
```

```
# if the cell type is neither 'NC' or 'PC', print this


# -------- Using the Function to Plot PSTH Graphs -------- #

trialnum = 500                # plot PSTH graphs for trial
number 500
isi = 500                     # plot PSTH graphs for files
whose interstimulus interval is 500 ms
makepsthgraph(data=psthfiles_nc[isi], celltype='NC',
trialnum=trialnum, isi=isi, ymin=-0.5, ymax=2.5)
makepsthgraph(data=psthfiles_pc[isi], celltype='PC',
trialnum=trialnum, isi=isi, ymin=-0.5, ymax=2.5)
```

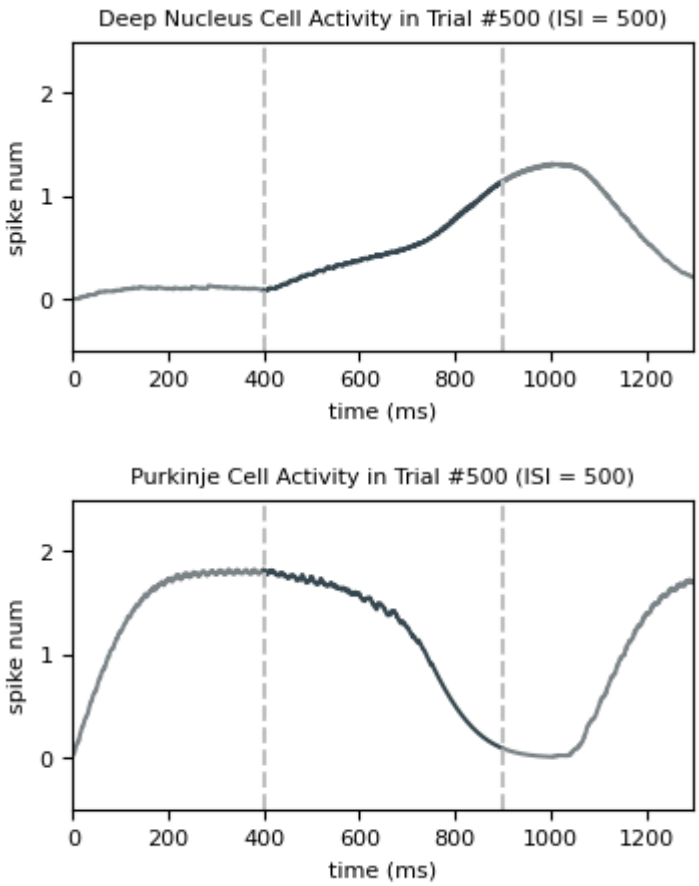


FIGURE 2. GIF of PSTH Graphs Across Trial

Using the function defined above that plots PSTH line graphs, the gif created in this section will show how neuron activities change over trials. Again, a psth data file has neuron activity across 1000 trials, the first 500 being acquisition trials and the following 500 being extinction trials. With the gif, we'll be able to observe how the shape of PSTH graphs change as the association between the two stimuli are acquired and extinguished from the neurons. Again, the x-axis is time in milliseconds while the y-axis is number of spikes. The graph on the top shows Deep Nucleus Cell activity, while the graph on the bottom shows Purkinje cell activity.

In [7]:

```
# -------- Adjusting the 'makepsthgraph' Function to be Used
for Making GIFs -------- #

# codes below has a similar function to the 'makepsthgraph'
function defined in the cell above
# some features of the function are deleted and added to make
it usable in the block of code that plots a gif
# axnum is specified because I'll be creating a figure with
subplots
def makepsthgraph_gif(data, trialnum, isi, ymin, ymax,
axnum):
    '''
    This function plots a PSTH graph for GIFs\n
        data = input PSTH file\n
        trialnum = choose the trial number to plot the graph
for\n
        isi = specify the interstimulus interval of the input
PSTH data\n
        ymin = minimum y value in the axis plot\n
        ymax = maximum y value in the axis plot\n
        axnum = the number of the axis on which this graph
will be plotted
    '''
    # nothing is much different here compared to the
'makepsthgraph' function from before
    yz = data.shape                                 # y is total
number of trials, z is total number of time points
    x = np.arange(0,yz[1])                          # make a list
of numbers from 0 to total time points (yz[1]) to create the
x coordinates for line graph

    # I made slight changes to this part so that I can plot
two different graphs in different axis later in the gif
    axnum.set_xlim(0,yz[1])                         # x axis
starts at 0 and ends at total number of time points (yz[1])
    axnum.set_ylim(ymin,ymax)                       # y axis
limits are set by parameter of the function
```

```
    # below are codes that plot a line graph for PSTH
    # again, these three plots are separated for setting
different colors for different time points
    # details are the same as the 'makepsthgraph' function
from before
    axnum.plot(x[0:400],data[trialnum,0:400],
color='#7B8387')
    axnum.plot(x[400:400+isi],data[trialnum,400:400+isi],
color='#37474F')
    axnum.plot(x[400+isi:yz[1]],data[trialnum,400+isi:yz[1]],
color='#7B8387')


# -------- Making Multiple PSTH Graphs Across Trials to
Create a GIF -------- #

# codes below are for making multiple PSTH plots

# this is an array for storing file names of every image,
which will be used later to reference to when creating the
gif
filenames = []

# this 'for loop' will:
    # loop through each trial number (from 0 to 1000, in
increments of 10)
    # plot the PSTH graph at the given trial
    # save it plotted image in local working directory under
the chosen file name
for i in range(0,1000,10):
    # plot a figure with two subplots
        # ax1 will have the PSTH graph of Deep Nucleus cells
        # ax2 will have the PSTH graph of Purkinje cells
    fig, (ax1, ax2) = plt.subplots(2, 1)

    # this 'if' function is to specify the title of the plot,
distinguishing between acquisition trials and extinction
```

```
trials
        # for the first 500 trials, the title will indicate
that they are acquisition trials
        # for the trials numbered 501 to 1000, the title will
indicate that they are extinciton trials
    if i <= 500:
        ax1.text(x=410, y=2.7, s='Trial #'+str(i)+' -
Acquisition', fontsize=12)
    else:
        ax1.text(x=410, y=2.7, s='Trial #'+str(i)+' -
Extinction', fontsize=12)

    # plot Deep Nucleus cell PSTH data (with isi 500) on axis
number 1
    makepsthgraph_gif(data=psthfiles_nc[500], trialnum=i,
isi=500, ymin=-0.5, ymax=2.5, axnum=ax1)
    # plot Purkinje cell PSTH data (with isi 500) on axis
number 2
    makepsthgraph_gif(data=psthfiles_pc[500], trialnum=i,
isi=500, ymin=-0.5, ymax=2.5, axnum=ax2)

    # this part is to create a dashed vertical line at the
points where the two stimuli are presented
    isi = 500
# this is because the isi of the files we are using is 500 ms
    ax1.axvline(400, color='#BFBFBF', linestyle='dashed')
# this is when the conditioned stimulus is presented (at 400
ms)
    ax1.axvline(400+isi, color='#BFBFBF', linestyle='dashed')
# this is when the unconditioned stimulus is presented (at
400+isi ms)
    ax2.axvline(400, color='#BFBFBF', linestyle='dashed')
# this is when the conditioned stimulus is presented (at 400
ms)
    ax2.axvline(400+isi, color='#BFBFBF', linestyle='dashed')
# this is when the unconditioned stimulus is presented (at
400+isi ms)
    # additional features of the graph
```

```
    ax1.set_ylabel('spike num', fontsize=10)
# set y label for graph in ax1
    ax2.set_xlabel('time (ms)', fontsize=10)
# set x label for graph in ax2, we don't have to do this for
graph in ax1 because they are vertically stacked up on each
other
    ax2.set_ylabel('spike num', fontsize=10)
# set y label for graph in ax2

    filename = f'{i}.png'
# file name will be 'i'.png, where i is the trial number
    filenames.append(filename)
# save the file name to 'filename' list we made earlier

    # save each images to the local working directory
    plt.savefig(filename)
    plt.close()


# -------- Building the GIF -------- #

# use a function from imageio library to create a gif
with imageio.get_writer('mygif.gif', mode='I', duration=0.1)
as writer:
    for filename in filenames:                          # loop
through each files by referencing to the 'filenames' list
created above
        image = imageio.imread(filename)                # store
the image in 'image' variable
        writer.append_data(image)                       # append
the image to the writer, which will create a gif


# -------- Removing the Files Used to Make the GIF -------- #

# codes below will remove files that were used for the gif
from the local working directory
# this is not necessary to make the gif but it helps keep the
```

```
local folder clean
for filename in set(filenames):                    # loop
through each files by referencing to the 'filenames' list
created above
    os.remove(filename)                            # remove
the file


# -------- Displaying the GIF -------- #

# codes below will display the created gif in jupyter
notebook
with open('mygif.gif','rb') as file:
    display(Image(data=file.read(), format='png'))
```

```
/var/folders/2m/mxjq630x5psgvt__r1b0z2sr0000gn/T/ipykernel_1212/3230299049.py:
86: DeprecationWarning: Starting with ImageIO v3 the behavior of this function
will switch to that of iio.v3.imread. To keep the current behavior (and make t
his warning disappear) use `import imageio.v2 as imageio` or call `imageio.v2.
imread` directly.
  image = imageio.imread(filename)             # store the image in 'image' var
iable
```

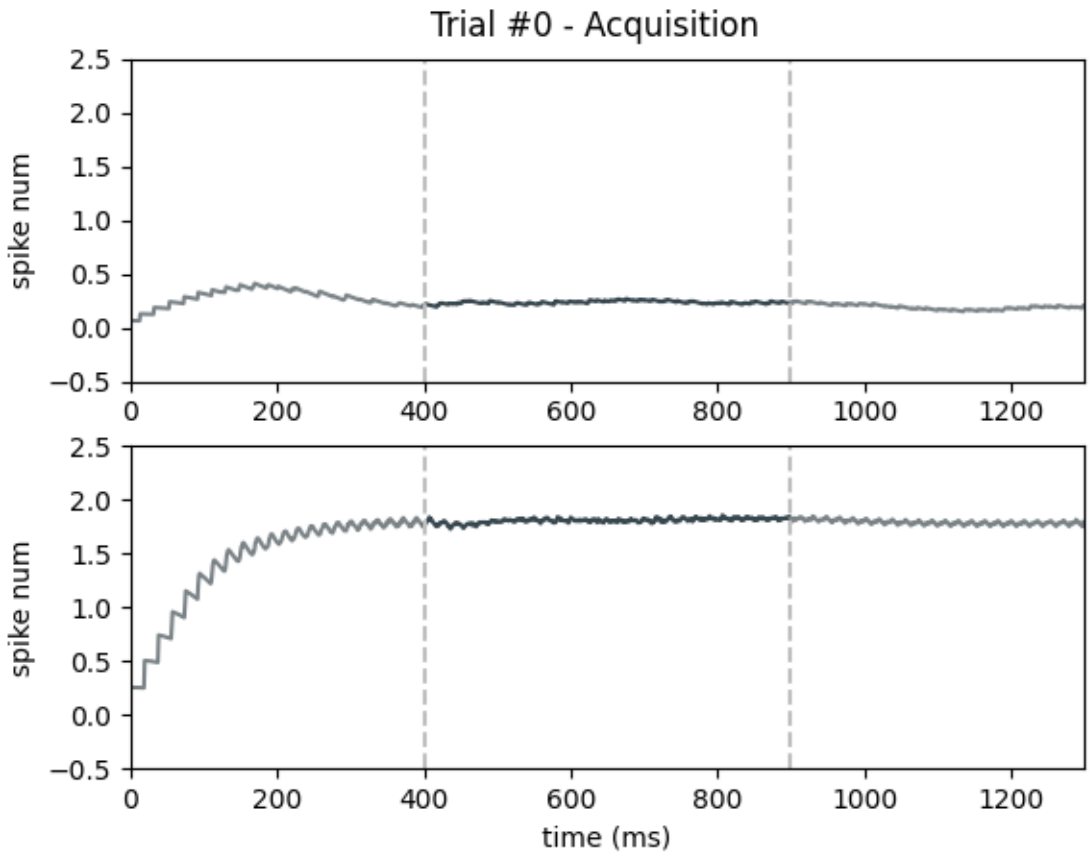


FIGURE 3. Graph of Purkinje Cell Activity (Minimum Spike Number) Across Trials

In order to compute the rate of acquisition and rate of extinction, I took the minimum spike number of the Purkinje cells from each trial. This is because Purkinje cells were observed to drastically decrease their spike numbers during conditioned response. In other words, when the neurons learn the association between the two stimuli, it starts decreasing their spikes right after the first stimuli is presented. By finding the minimum point of spike number, we can analyze whether learning has occured or not and by how much.

I plotted a series of line graphs that display the change of minimum spike activity in Purkinje cells across trials. There are 10 subplots in total. Each column represents data from different interstimulus intervals (i.e. 250 ms, 500 ms, 750 ms, 1000 ms, and 1250 ms). Within a single isi data, the graph on top represents how minimum spike number changes during acquisition trials and the graph on bottom represents how minimum spike number changes during extinction trials. The x-axis in each graph is trial number and the y-axis is minimum spike number of that given trial.

Overall, during acquisition trials, we can observe that the minimum spike number of Purkinje cells gradually decrease over trial. Decrease means that the neurons learned the association. During extinction trials, minimum spike number of Purkinje cells gradually increase as they extinguish the learned association. I plotted vertical dashed lines to highlight the time points where complete acquisition or complete extinction happens.

In [8]:

```
# -------- Defining a Function to Plot Purkinje Cell Activity
Across Trials -------- #

# this function plots a graph of minimum spike number across
trials
# x-axis is each trial and y-axis is minimum spike number of
that given trial
# this function is for Purkinje cell PSTH data
def minspikegraph_subplot(data, isi, start_trial, end_trial,
vertical_line, axnum):
    '''
    This function plots minimum spike number of Purkinje
cells during each trial\n
    data = input PSTH file\n
    isi = specify the interstimulus interval of the input
PSTH data\n
    start_trial = specify the first trial number to plot in
the graph\n
    end_trial = specify the last trial number to plot in the
graph\n
    vertical_line = x coordinate of the vertical dashed line
```

```
which indicates where complete acquisition/extinction
happens\n
    axnum = the index of the axis on which this graph will be
plotted
    '''
    num_trial = end_trial - start_trial
# calculate the total number of trials to plot
    min_spikes = np.zeros(num_trial)
# make an array with the size of total number of trials to
plot
    for i in range(0, num_trial):
        # find the minimum spike number of Purkinje cells
after the stimulus was presented (400 ms ~) and store it in
the variable 'min_spike'
        min_spike = min(data[i+start_trial, 400:400+isi+400])
        min_spikes[i] = min_spike
# store the 'min_spike' of each trial to the array
'min_spikes'
    x = np.arange(start_trial, end_trial)
# this array is for the x coordinates of the data

    # additional features of the graph
    axnum.set_xlabel('trial num', fontsize=8)
# x-axis label
    axnum.set_ylabel('minimum spike num', fontsize=8)
# y-axis label
    axnum.tick_params(labelsize=8)
# label font size

    # plotting dashed lines to help the viewers interpret
data
    axnum.axhline(1.77, color='#BFBFBF', linestyle='dashed')
# horizontal line to show the baseline spike number when
neuron has not learned
    axnum.axhline(0, color='#BFBFBF', linestyle='dashed')
# horizontal line to show the spike number when neuron has
learned the association
    axnum.axvline(vertical_line, color='#BFBFBF',
```

```
linestyle='dashed')   # vertical line to highlight the trial
number where acquisition/extinciton happens

    # plotting the line graph
    axnum.plot(x, min_spikes, color='#37474F')


# -------- Using the Function to Plot Purkinje Cell Activity
Across Trials -------- #

# make a figure that has subplots of 2 rows and 5 columns
    # each column will be data from different ISI
    # top row will be graphs during acquisition
    # bottom row will be graphs during extinction
fig, (axs) = plt.subplots(2, 5, figsize=(20,4))
fig.tight_layout(pad=1)

# title of the entire graph
axs[0][2].text(-170, 2.5, 'Purkinje Cell Spike Activity
During Acquisition and Extinction', fontsize=12)

# titles to indicate graphs from different ISIs
axs[0][0].set_title('Interstimulus Interval: 250 ms',
fontsize=10)
axs[0][1].set_title('Interstimulus Interval: 500 ms',
fontsize=10)
axs[0][2].set_title('Interstimulus Interval: 750 ms',
fontsize=10)
axs[0][3].set_title('Interstimulus Interval: 1000 ms',
fontsize=10)
axs[0][4].set_title('Interstimulus Interval: 1250 ms',
fontsize=10)

# codes below make subplots of Purkinje cell activity graphs
# the x coordinates of vertical lines were determined by
visual inspection of the trial number where the graph reaches
acquisition/extinction
```

```
# plotting the graph for when the interstimulus interval is
250 ms
isi = 250
# acquisition trial graph
minspikegraph_subplot(data=psthfiles_pc[isi], isi=isi,
start_trial=0, end_trial=500, vertical_line=250, axnum=axs[0]
[0])
# extinction trial graph
minspikegraph_subplot(data=psthfiles_pc[isi], isi=isi,
start_trial=500, end_trial=1000, vertical_line=765,
axnum=axs[1][0])

# plotting the graph for when the interstimulus interval is
500 ms
isi = 500
# acquisition trial graph
minspikegraph_subplot(data=psthfiles_pc[isi], isi=isi,
start_trial=0, end_trial=500, vertical_line=110, axnum=axs[0]
[1])
# extinction trial graph
minspikegraph_subplot(data=psthfiles_pc[isi], isi=isi,
start_trial=500, end_trial=1000, vertical_line=580,
axnum=axs[1][1])

# plotting the graph for when the interstimulus interval is
750 ms
isi = 750
# acquisition trial graph
minspikegraph_subplot(data=psthfiles_pc[isi], isi=isi,
start_trial=0, end_trial=500, vertical_line=110, axnum=axs[0]
[2])
# extinction trial graph
minspikegraph_subplot(data=psthfiles_pc[isi], isi=isi,
start_trial=500, end_trial=1000, vertical_line=570,
axnum=axs[1][2])

# plotting the graph for when the interstimulus interval is
1000 ms
```

```
isi = 1000
# acquisition trial graph
minspikegraph_subplot(data=psthfiles_pc[isi], isi=isi,
start_trial=0, end_trial=500, vertical_line=130, axnum=axs[0]
[3])
# extinction trial graph
minspikegraph_subplot(data=psthfiles_pc[isi], isi=isi,
start_trial=500, end_trial=1000, vertical_line=590,
axnum=axs[1][3])

# plotting the graph for when the interstimulus interval is
1250 ms
isi = 1250
# acquisition trial graph
minspikegraph_subplot(data=psthfiles_pc[isi], isi=isi,
start_trial=0, end_trial=500, vertical_line=335, axnum=axs[0]
[4])
# extinction trial graph
minspikegraph_subplot(data=psthfiles_pc[isi], isi=isi,
start_trial=500, end_trial=1000, vertical_line=565,
axnum=axs[1][4])
```

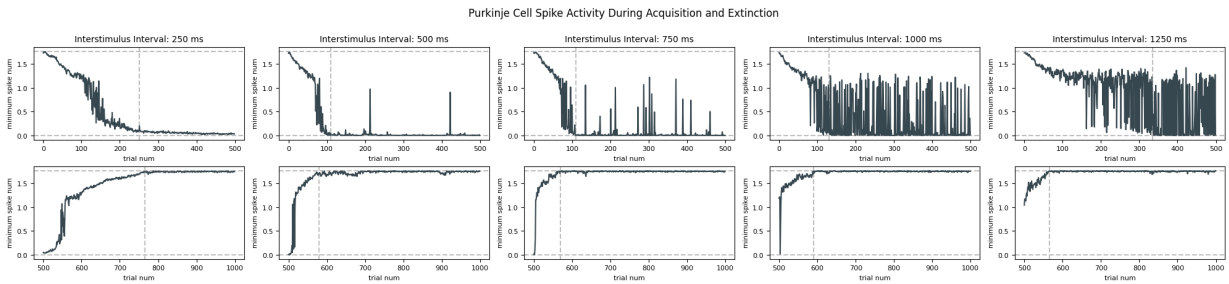


FIGURE 4. Linear Regression Analysis for Rates of Acquisition and Extinction in Purkinje Cells

In FIGURE 3., we were able to obtain the trial numbers where acquisition and extinction happens in Purkinje cells across different interstimulus intervals. Using this information, I conducted a linear regression analysis to examine the rates of acquisition and extinction in each case. The figure in this section visualizes the linear regression lines that represent acquisition and extinction rates.

Just like FIGURE 3., there are 10 subplots in this figure. Each column of subplots represent data from different ISIs. The top graph shows the acquisition trials from that ISI, while the bottom graph shows the extinction trials from that ISI. The x-axis is trial number and y-axis is minimum spike number of Purkinje cells at each trial. Unlike FIGURE 3., the minimum spike numbers from each trial are plotted as scatter plots. Regression lines are superimposed in a darker color.

In [9]:

```
# -------- Defining a Function to Plot Rates of Acquisition
and Extinction -------- #

def spikeregression_subplot(data, isi, reg_start_trial,
reg_end_trial, plot_start_trial, plot_end_trial, axnum):
    '''
    This function plots linear regression lines above the
original graph of minimum spike number of Purkinje cells\n
    data = input PSTH file\n
    isi = specify the interstimulus interval of the input
PSTH data\n
    reg_start_trial = specify the first trial number to
conduct the regression analysis\n
    reg_end_trial = specify the last trial number to conduct
the regression analysis\n
    plot_start_trial = specify the first trial number to plot
in the graph\n
    plot_end_trial = specify the last trial number to plot in
the graph\n
    axnum = the index of the axis on which this graph will be
plotted
    '''
    num_trial = plot_end_trial - plot_start_trial
# calculate the total number of trials to plot
    min_spikes = np.zeros(num_trial)
# make an array with the size of total number of trials to
plot
    for i in range(0, num_trial):
        # find the minimum spike number of Purkinje cells
after the stimulus was presented (400 ms ~) and store it in
the variable 'min_spike'
        min_spike = min(data[i+plot_start_trial,
400:400+isi+400])
        min_spikes[i] = min_spike
# store the 'min_spike' of each trial to the array
'min_spikes'

    x = np.arange(reg_start_trial, reg_end_trial)
```

```
# this array is for the x coordinates of regression analysis
data
    x = x.reshape(-1,1)
# reshape into this format because that's how the 'reg.fit()'
function likes it
    # store a subset of the 'min_spikes' array into 'regdata'
variable
    # this variable stores the minimum spike number for the
trials that we're doing a regression analysis on
    regdata = min_spikes[reg_start_trial -
plot_start_trial:reg_end_trial -
plot_start_trial].reshape(-1, 1)

    # codes below are for linear regressiona analysis
    reg = LinearRegression()
    reg.fit(x, regdata)
# conduct linear regression analysis using 'regdata' subset
of 'min_spikes'
    regdata_pred = reg.predict(x)
# store the fitted prediction line

    plot_x = np.arange(plot_start_trial, plot_end_trial)
# this array is for the x coordinates of the data to plot

    axnum.scatter(plot_x, min_spikes, s=1, color='#7B8387')
# scatter plot the minimum spike number for each trial number
    axnum.plot(x, regdata_pred, color='#37474F')
# plot the fitted prediction line graph

    # additional features of the graph
    axnum.set_xlabel('trial num', fontsize=8)
# x-axis label
    axnum.set_ylabel('minimum spike num', fontsize=8)
# y-axis label
    axnum.tick_params(labelsize=8)
# label font size
    axnum.set_ylim(-0.1, 1.87)
# y-axis limits
```

```
    # plot vertical dashed lines at the trial numbers where
regression analysis started and ended
    axnum.axvline(reg_start_trial, color='#BFBFBF',
linestyle='dashed')
    axnum.axvline(reg_end_trial, color='#BFBFBF',
linestyle='dashed')


# -------- Using the Function to Plot Rates of Acquisition
and Extinction -------- #

# make a figure that has subplots of 2 rows and 5 columns
    # each column will be data from different ISI
    # top row will be graphs during acquisition
    # bottom row will be graphs during extinction
fig, (axs) = plt.subplots(2, 5, figsize=(20,4))
fig.tight_layout(pad=1)

# title of the entire graph
axs[0][2].text(-170, 2.5, 'Regression Analysis for Rates of
Acquisition and Extinction', fontsize=12)

# titles to indicate graphs from different ISIs
axs[0][0].set_title('Interstimulus Interval: 250 ms',
fontsize=10)
axs[0][1].set_title('Interstimulus Interval: 500 ms',
fontsize=10)
axs[0][2].set_title('Interstimulus Interval: 750 ms',
fontsize=10)
axs[0][3].set_title('Interstimulus Interval: 1000 ms',
fontsize=10)
axs[0][4].set_title('Interstimulus Interval: 1250 ms',
fontsize=10)

# codes below make subplots using the
'spikeregression_subplot' function defined above
# the points of acquisition/extinction gathered from FIGURE
```

```
3. were used to determine the trial numbers to do regression
analysis on

# plotting the graph for when the interstimulus interval is
250 ms
isi = 250
# acquisition trial graph
spikeregression_subplot(data=psthfiles_pc[isi], isi=isi,
reg_start_trial=0, reg_end_trial=250, plot_start_trial=0,
plot_end_trial=500, axnum=axs[0][0])
# extinction trial graph
spikeregression_subplot(data=psthfiles_pc[isi], isi=isi,
reg_start_trial=500, reg_end_trial=765, plot_start_trial=500,
plot_end_trial=1000, axnum=axs[1][0])

# plotting the graph for when the interstimulus interval is
500 ms
isi = 500
# acquisition trial graph
spikeregression_subplot(data=psthfiles_pc[isi], isi=isi,
reg_start_trial=0, reg_end_trial=110, plot_start_trial=0,
plot_end_trial=500, axnum=axs[0][1])
# extinction trial graph
spikeregression_subplot(data=psthfiles_pc[isi], isi=isi,
reg_start_trial=500, reg_end_trial=580, plot_start_trial=500,
plot_end_trial=1000, axnum=axs[1][1])

# plotting the graph for when the interstimulus interval is
750 ms
isi = 750
# acquisition trial graph
spikeregression_subplot(data=psthfiles_pc[isi], isi=isi,
reg_start_trial=0, reg_end_trial=110, plot_start_trial=0,
plot_end_trial=500, axnum=axs[0][2])
# extinction trial graph
spikeregression_subplot(data=psthfiles_pc[isi], isi=isi,
reg_start_trial=500, reg_end_trial=570, plot_start_trial=500,
plot_end_trial=1000, axnum=axs[1][2])
```

```
# plotting the graph for when the interstimulus interval is
1000 ms
isi = 1000
# acquisition trial graph
spikeregression_subplot(data=psthfiles_pc[isi], isi=isi,
reg_start_trial=0, reg_end_trial=130, plot_start_trial=0,
plot_end_trial=500, axnum=axs[0][3])
# extinction trial graph
spikeregression_subplot(data=psthfiles_pc[isi], isi=isi,
reg_start_trial=500, reg_end_trial=590, plot_start_trial=500,
plot_end_trial=1000, axnum=axs[1][3])

# plotting the graph for when the interstimulus interval is
1250 ms
isi = 1250
# acquisition trial graph
spikeregression_subplot(data=psthfiles_pc[isi], isi=isi,
reg_start_trial=0, reg_end_trial=335, plot_start_trial=0,
plot_end_trial=500, axnum=axs[0][4])
# extinction trial graph
spikeregression_subplot(data=psthfiles_pc[isi], isi=isi,
reg_start_trial=500, reg_end_trial=565, plot_start_trial=500,
plot_end_trial=1000, axnum=axs[1][4])
```

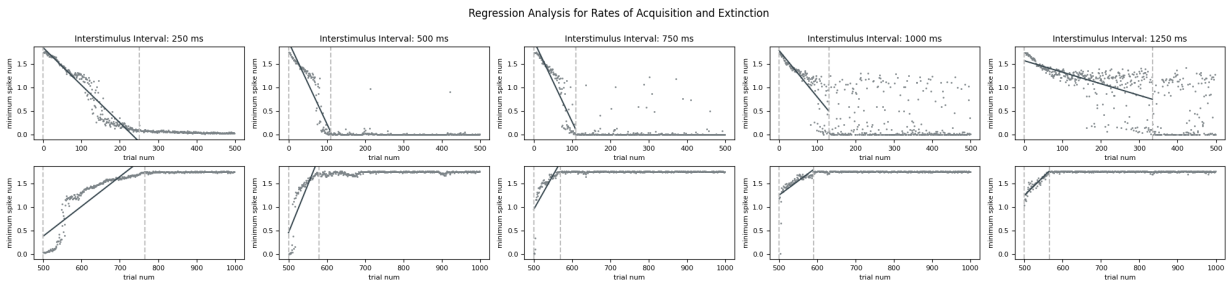


FIGURE 5. Acquisition and Extinction Rates in Trials with Different Interstimulus Intervals

Finally, using information gathered from FIGURE 3. and FIGURE 4., I made a grouped bar plot showing the acquisition and extinction rates from trials with different ISIs. Acquisition and extinction rates were obtained using slope coefficient values from linear regression analysis in FIGURE 4. Light-colored bars show the acquisition rates and dark-colored bars show the extinction rates. The x-axis is different ISIs and y-axis is slope coefficient. Higher y value means the acquisition/extinction happened in a higher rate.

In [10]:

```
# -------- Defining a Function to Get Slope Coefficient from Linear Regression Analysis -------- #

def get_regressioncoef(data, start_trial, end_trial):
    '''
    This function conducts linear regression and calculates the slope coefficient of values between given start trial number and end trial number\n
    data = input PSTH file\n
    start_trial = specify the first trial number to conduct the regression analysis\n
    end_trial = specify the last trial number to conduct the regression analysis
    '''
    num_trial = end_trial - start_trial
# calculate the total number of trials to do regression analysis
    min_spikes = np.zeros(num_trial)
# make an array with the size of total number of trials to plot
    for i in range(0, num_trial):
        # find the minimum spike number of Purkinje cells after the stimulus was presented (400 ms ~) and store it in the variable 'min_spike'
        min_spike = min(data[i+start_trial, 400:400+isi+400])
        min_spikes[i] = min_spike
# store the 'min_spike' of each trial to the array 'min_spikes'

    x = np.arange(start_trial, end_trial)
# this array is for the x coordinates of regression analysis data
    x = x.reshape(-1,1)
# reshape into this format because that's how the 'reg.fit()' function likes it
    regdata = min_spikes.reshape(-1, 1)
# reshape into this format because that's how the 'reg.fit()' function likes it
```

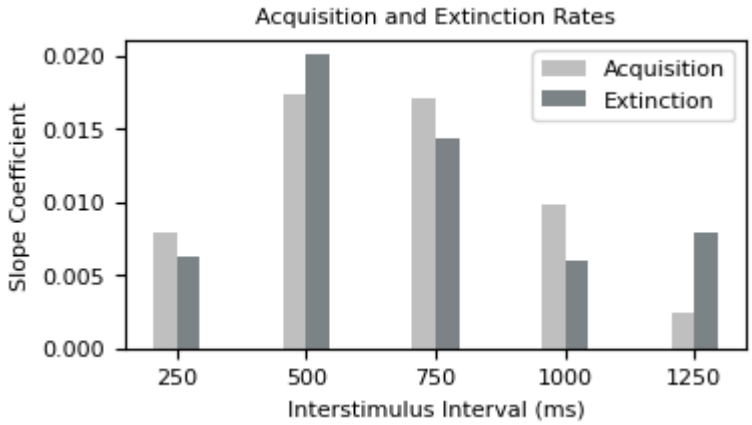
```
    # codes below are for linear regressiona analysis
    reg = LinearRegression()
    reg.fit(x, regdata)

    # return slope coefficient
    return reg.coef_


# -------- Using the Function to Obtain Rates of Acquisition
and Extinction -------- #

reg_coefs = {}                          # create a dictionary
to store slope coefficients
for isi in filelist:                    # loop through the
different ISIs that I prepared in the 'filelist' list above
    reg_coefs[isi] = np.zeros(2)        # using isi as key,
each value will be a list of 2 elements (acquisition rate,
extinction rate)

# plotting the graph for when the interstimulus interval is
250 ms
isi = 250
# calculate acquisition rate and store it as first element in
the value
reg_coefs[isi][0] =
get_regressioncoef(data=psthfiles_pc[isi], start_trial=0,
end_trial=250)[0,0]
# calculate extinction rate and store it as second element in
the value
reg_coefs[isi][1] =
get_regressioncoef(data=psthfiles_pc[isi], start_trial=500,
end_trial=765)[0,0]

# plotting the graph for when the interstimulus interval is
500 ms
isi = 500
# calculate acquisition rate and store it as first element in
```

```
the value
reg_coefs[isi][0] =
get_regressioncoef(data=psthfiles_pc[isi], start_trial=0,
end_trial=110)[0,0]
# calculate extinction rate and store it as second element in
the value
reg_coefs[isi][1] =
get_regressioncoef(data=psthfiles_pc[isi], start_trial=500,
end_trial=580)[0,0]

# plotting the graph for when the interstimulus interval is
750 ms
isi = 750
# calculate acquisition rate and store it as first element in
the value
reg_coefs[isi][0] =
get_regressioncoef(data=psthfiles_pc[isi], start_trial=0,
end_trial=110)[0,0]
# calculate extinction rate and store it as second element in
the value
reg_coefs[isi][1] =
get_regressioncoef(data=psthfiles_pc[isi], start_trial=500,
end_trial=570)[0,0]

# plotting the graph for when the interstimulus interval is
1000 ms
isi = 1000
# calculate acquisition rate and store it as first element in
the value
reg_coefs[isi][0] =
get_regressioncoef(data=psthfiles_pc[isi], start_trial=0,
end_trial=130)[0,0]
# calculate extinction rate and store it as second element in
the value
reg_coefs[isi][1] =
get_regressioncoef(data=psthfiles_pc[isi], start_trial=500,
end_trial=590)[0,0]
```

```
# plotting the graph for when the interstimulus interval is
1250 ms
isi = 1250
# calculate acquisition rate and store it as first element in
the value
reg_coefs[isi][0] =
get_regressioncoef(data=psthfiles_pc[isi], start_trial=0,
end_trial=335)[0,0]
# calculate extinction rate and store it as second element in
the value
reg_coefs[isi][1] =
get_regressioncoef(data=psthfiles_pc[isi], start_trial=500,
end_trial=565)[0,0]



# -------- Using the Calculated Rates of Acquisition and
Extinction to Make Grouped Bar Plot -------- #

fig, ax = plt.subplots(figsize=(4,2))

x = np.array(filelist)                          # make an
array of ISIs to use as x coordinates
width = 45                                      # set width
of each bar plot

acq_coefs = []                                  # make a list
to store slope coefficients for acquisition rates
for isi in filelist:                            # loop
through different ISIs
    acq_coefs.append(abs(reg_coefs[isi][0]))    # grab
acquisition rates from the 'reg_coefs' dictionary created
above and append it to the 'acq_coefs' list
ext_coefs = []                                  # make a list
to store slope coefficients for extinction rates
for isi in filelist:                            # loop
through different ISIs
    ext_coefs.append(reg_coefs[isi][1])         # grab
extinction rates from the 'reg_coefs' dictionary created
```

```
above and append it to the 'ext_coefs' list

rects1 = ax.bar(x - width/2, acq_coefs, width,
label='Acquisition', color='#BFBFBF')        # plot bar plots
for acquisition rate
rects2 = ax.bar(x + width/2, ext_coefs, width,
label='Extinction', color='#7B8387')         # plot bar plots
for extinction rate

# additional features of the graph
ax.set_xticks(filelist)
ax.tick_params(labelsize=8)
ax.set_xlabel('Interstimulus Interval (ms)', fontsize=8)
# x-axis label
ax.set_ylabel('Slope Coefficient', fontsize=8)
# y-axis label
ax.legend(prop={'size': 8})
# make a legend
ax.set_title('Acquisition and Extinction Rates', fontsize=8)
# title of the graph
```

Out[10]:
```
Text(0.5, 1.0, 'Acquisition and Extinction Rates')
```



STATISTICAL COMPARISON

To investigate whether there is a significant statistical difference between rates of acquisition and rates of extinction across trials with different ISI, I conducted a paired t-test. The null hypothesis is that there are no difference between the rates of acquisition and extinction. According to the paired t-test result, I failed to reject the null hypothesis (t=0.01, p=0.99). There was no significant difference between rates of acquisition and extinction.

In [11]:
```
stats.ttest_rel(acq_coefs, ext_coefs)
```

Out[11]: Ttest_relResult(statistic=0.010965481997950362, pvalue=0.9917760945121041)

IV. DISCUSSION

Overall, I learned that the patterns of acquisition and extinction vary greatly by ISI. Although rates of acquisition and extinction differed within a single ISI data, there was no significant pattern of difference when datasets from all ISI were used to conduct a paired t-test.

In FIGURE 3., there was a slight pattern where, as the interstimulus interval of the data increased, the number of trials to get to extinction decreased, which means extinction happened relatively faster. Except for ISI 250 ms, the number of trials to get to acquisition increased as ISI increased as well. However, this pattern was not directly represented in my acquisition/extinction rate analysis, since I conducted linear regression analysis that takes the y-axis (minimum spike num) into account as well as the x-axis (trial number).

In FIGURE 5., the rates of acquisition and extinction tend to decrease as ISI increases, although the trial with ISI 250 ms did not fit this pattern.

In conclusion, no significant difference in the rates of acquisition and extinction were found from my analysis. Instead, interesting differences in acquisition/extinction rates among different ISIs were revealed.

A limitation of my analysis was that I had only 1 data file from each interstimulus intervals. Having higher number of datsets would have helped the statistical comparison. Paired t-test analysis could have yielded a more significant result if I had gathered multiple data from each ISI. Future studies could be done by gathering multiple datasets and conducting further statistical analysis using ANOVA test or paired t-test.