Assignment 4

NESC 3505

Instructions

Read the stuff below, and write code to answer the questions.

Try to make your code as Pythonic as possible, but prioritize "it's working" over "it's pretty". Likewise, try to tackle problems one piece at a time, and getting one bit of code working before building on it. For example, get drawing one plot working before trying to draw a bunch of subplots, and get a bunch of subplots working before adding labels or changing axes.

You are allowed to consult and discuss the assignment with other students, but each person should write their own code. Try to give hints or small demos rather than showing a lot of code - you should be trying to help each other learn, rather than showing off that you can do a thing.

Although this assignment stays close to the single unit chapter in the textbook, not everything you need is shown there. The rest is elsewhere in the book, but may be also found through internet searches. It's certainly valid to use internet resources to help you, but be sure you understand what you're doing.

Always use object-oriented plotting in Matplotlib for this assignment.

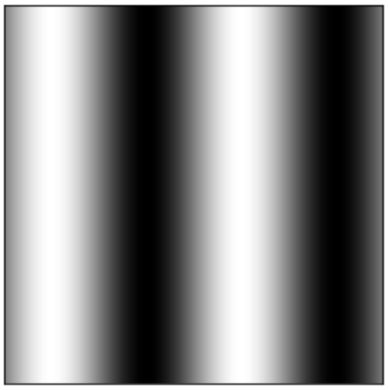
Data Source

These data were provided courtesy of Dr. Nathan Crowder, Department of Psychology & Neuroscience, Dalhousie University. Single-unit spike data were collected from several neurons in the primary visual cortex of mice.

Note that although we have data from multiple neurons, these are *not* data from a multielectrode array, as were described in the textbook. Data were recorded from single electrodes that were embedded directly into single neurons, a technique called patch clamping. We have data from multiple neurons because Dr. Crowder's team painstakingly recorded from several neurons in each of a few animals. One of the important differences between this data set, and multielectrode array data, is that multielectrode arrays record from many neurons *simultaneously*, whereas patch clamping records from only one neuron at a time, so each neuron's data were recorded at different times. But for the purposes of this assignment, we can work with the data from multiple neurons in much the same ways as shown in the textbook.

Prior to completing this assignment, you should read the Single Unit Data chapter of the textbook.

Experimental Design



The experiment involved presenting **sine wave gratings** (like in the image above) to mice while recording from electrodes in the primary visual cortex. In the experiment the gratings oscillated (drifted) at a speed of 2 Hz (faster than the animation above). This means that, 2 times per second, the colour of a given pixel in the image fluctuated between black and white, through levels of grey. This creates the impression that the grating is moving sideways, but effectively it means that the receptive field of a neuron would be stimulated by intensity varying at 2 Hz, following the formula of a sine wave.(If the word *grating* seems weird to you, think of a drain grate in the road, or a vent grate).

Each trial was 4000 ms in duration, and the sine wave grating stimulus was presented from 2000–3000 ms after the start of each trial.

Two variables were manipulated in this experiment:

Variable 1

The **contrast** of the sine wave grating stimuli were varied systematically over ten levels from 0 (no contrast between the brightest and darkest parts of the sine wave grating, which would appear as uniform grey) to 100% (as in the example above). **Contrast** is therefore one of the experimental variables; the 10 levels of contrast used were 4, 8, 12, 16, 24, 32, 48, 64, 84, and 100%.

Variable 2

The stimuli were presented under 2 experimental **conditions**, *control* (CTRL) and *adaptation* (ADAPT):

- In the **CTRL** condition, each trial began with a blank grey screen, then after 2000 ms the oscillating sine wave grating was presented, at the trial's contrast level. The stimulus was on for 1000 ms, then was replaced by a blank grey screen for the last 1000 ms of the trial.
- In the **ADAPT** condition, each trial started with the *adaptation stimulus*, which was a sine wave grating — like the main stimulus, but always at 50% contrast. This adaptation stimulus was presented for the first 2000 ms of the trial. After this, at 2000 ms, the main stimulus grating was shown, at the trial's contrast level. As in the CTRL condition, the main stimulus was on for 1000 ms and then a blank grey screen was shown for the remaining 1000 ms of the trial.

A total of 8 trials were presented at each contrast level, in each condition. In the data file below, trial number is in the column *repetition*.

The data were sampled at 1000 Hz, meaning that we have a possible data point every 1 ms. I say "possible" because the data are stored in sparse (spike time) format, so there are only data points when spikes occurred. But you'll see that that spike times are integers, in units of milliseconds.

Hypotheses

1. In the control condition, latency to first spike is expected to decrease with increasing contrast.
2. Some primary visual cortex cells are phase sensitive (so-called **simple cells**), so in response to a drifting sine wave grating they will oscillate between excitation and inhibition at the same temporal frequency of the stimulus (2 Hz in this case).

Questions

Q1

Import Packages

Import pandas, NumPy, and Matplotlibpyplot, using their conventional aliases

In [1]:
```
#See Spike Train Unit in textbook
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

Q2

Read in data

Read in the data from the CSV file `crowder_data_all.csv`, and assign it to a pandas DataFrame called `df`

In [2]:
```
#File Name = crowder_data_all.csv

df = pd.read_csv('crowder_data_all.csv')
```

Q3

Look at the data

We should, naturally, look at the data frame and see how it's structured

Show the first 15 rows of the DataFrame

In [3]:
```
#First 15 rows
df.head(15)
```

Out[3]:

	neuron	spiketime	repetition	contrast	condition
0	m1_6	2008	2	4	CTRL
1	m1_6	1782	7	4	CTRL
2	m1_6	222	1	8	CTRL
3	m1_6	1603	1	8	CTRL
4	m1_6	1767	1	8	CTRL
5	m1_6	2838	2	8	CTRL
6	m1_6	344	4	8	CTRL
7	m1_6	1487	5	8	CTRL

	neuron	spiketime	repetition	contrast	condition
8	m1_6	3828	6	8	CTRL
9	m1_6	3835	6	8	CTRL
10	m1_6	88	1	12	CTRL
11	m1_6	1753	1	12	CTRL
12	m1_6	2910	1	12	CTRL
13	m1_6	3286	3	12	CTRL
14	m1_6	443	4	12	CTRL

Q4

Sample

Show a random sample of 20 rows of the DataFrame

In [4]:
```
#Sample 20
df.sample(20)
```

Out[4]:

	neuron	spiketime	repetition	contrast	condition
17129	m3_3	1417	4	48	ADAPT
24101	m4_17	381	6	12	ADAPT
30548	m6_17b	1328	1	48	ADAPT
7611	m2_9	529	3	64	ADAPT
13433	m3_3	2824	4	8	CTRL
16982	m3_3	555	1	48	ADAPT
15895	m3_3	1579	8	8	ADAPT
8410	m2_9	299	2	100	ADAPT
16636	m3_3	92	7	24	ADAPT
26208	m4_17	1168	5	84	ADAPT
21036	m6_3b2	961	5	32	ADAPT
25285	m4_17	2115	1	48	ADAPT
22839	m4_17	2666	2	84	CTRL
1941	m1_6	1444	7	100	ADAPT
12078	m2_13	1473	5	48	ADAPT
5673	m2_9	119	5	12	ADAPT
18899	m6_3a1	709	4	16	ADAPT
24047	m4_17	1432	4	12	ADAPT
19191	m6_3a1	563	1	64	ADAPT
7590	m2_9	2621	2	64	ADAPT

Q5

Define experiment parameters

There are a few things we should hard-code based on our knowledge of the experimental design, and some others we can extract from the DataFrame itself. Assign values to the following variables (and uncomment the variable names as you go!):

In [5]:
```
# Names of all neurons  - derive from DataFrames
neurons = df['neuron'].unique()

# All condition labels - derive from DataFrame
conditions = df['condition'].unique()

# Labels for the levels of stimulus contrast - derive from DataFrame
contr_levels = df['contrast'].unique()

# Labels for different trial ID numbers Repitition is the Trial ID#
trials = df['repetition'].unique()


# Hard-code (i.e., manually enter) the rest of these parameters based on the
descriptionof the experiment and/or derive from other parameters you define
trial_length = 4000

# Stimulus timing
stim_on_time = 2000
stim_off_time = 3000

adapt_on_time = 0
adapt_off_time = 2000
```

Each trial was 4000 ms in duration, and the sine wave grating stimulus was presented from 2000–3000 ms after the start of each trial.

In the CTRL condition, each trial began with a blank grey screen, then after 2000 ms the oscillating sine wave grating was presented, at the trial's contrast level. The stimulus was on for 1000 ms, then was replaced by a blank grey screen for the last 1000 ms of the trial. In the ADAPT condition, each trial started with the adaptation stimulus, which was a sine wave grating — like the main stimulus, but always at 50% contrast. This adaptation stimulus was presented for the first 2000 ms of the trial. After this, at 2000 ms, the main stimulus grating was shown, at the trial's contrast level. As in the CTRL condition, the main stimulus was on for 1000 ms and then a blank grey screen was shown for the remaining 1000 ms of the trial. A total of 8 trials were presented at each contrast level, in each condition. In the data file below, trial number is in the column repetition.

The data were sampled at 1000 Hz, meaning that we have a possible data point every 1 ms. I say "possible" because the data are stored in sparse (spike time) format, so there are only data points when spikes occurred. But you'll see that that spike times are integers, in units of milliseconds.

Run this cell to check the results of above

In [6]:
```
print("Trial numbers:", trials)
print("Conditions:", conditions)
print("Neuron IDs:", neurons)
print("Contrast Levels:", contr_levels)
```

Out[6]:
```
Trial numbers: [2 7 1 4 5 6 3 8]
Conditions: ['CTRL' 'ADAPT']
Neuron IDs: ['m1_6' 'm2_9' 'm2_13' 'm3_3' 'm6_3a1' 'm6_3b2' 'm4_17' 'm6_17b']
Contrast Levels: [  4   8  12  16  24  32  48  64  84 100]
```

Q6

Subset data

Create a DataFrame `dat` that contains the data from neuron `m1_6` in the control condition, at 48% contrast:

In [9]:
```
# YOUR CODE HERE
dat =  df[ (df['neuron'] == 'm1_6') & (df['condition'] == 'CTRL') & (df['contrast'] == 48) ]
dat
# dat.sample(20)
```

Out[9]:

	neuron	spiketime	repetition	contrast	condition
85	m1_6	1504	1	48	CTRL
86	m1_6	2361	1	48	CTRL
87	m1_6	2374	1	48	CTRL
88	m1_6	2508	1	48	CTRL
89	m1_6	2657	1	48	CTRL
...	...	...	...	...	...
152	m1_6	2602	8	48	CTRL
153	m1_6	2634	8	48	CTRL
154	m1_6	2689	8	48	CTRL
155	m1_6	2717	8	48	CTRL
156	m1_6	2838	8	48	CTRL

72 rows × 5 columns

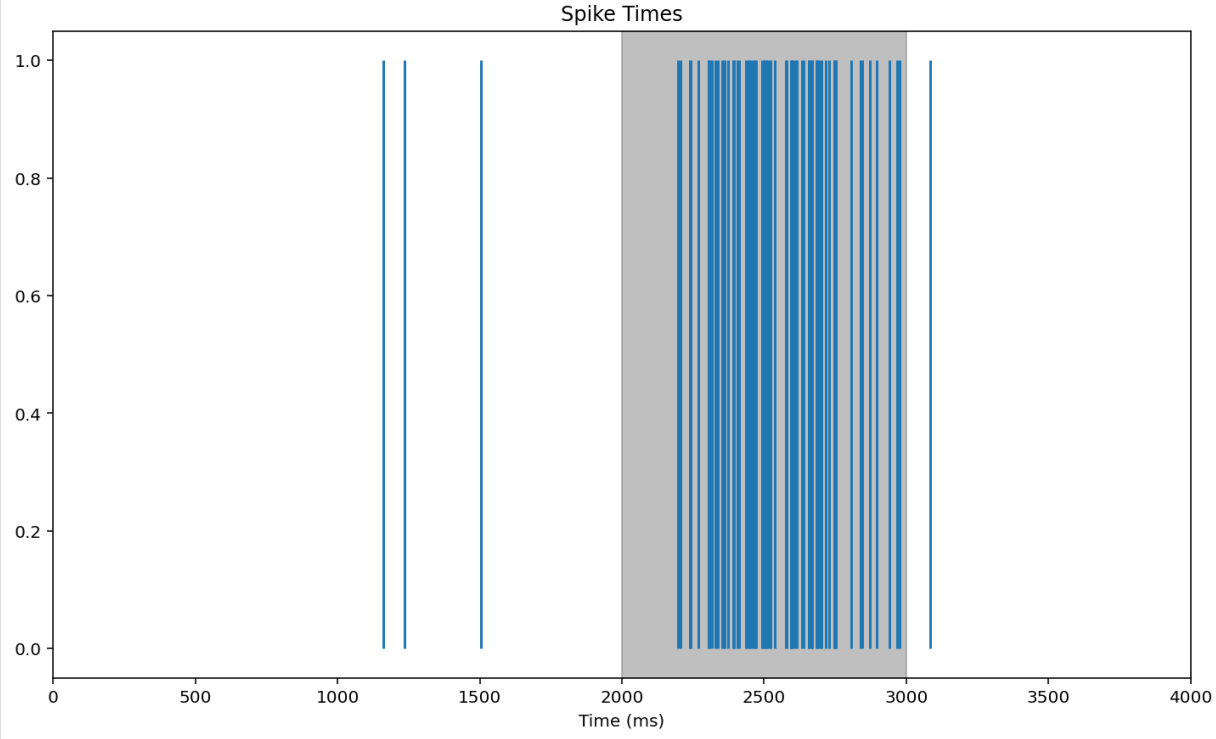
Q7

Raster Plot

Draw a raster plot of the spike time data in `dat`. Make sure that the *x* axis starts at 0 and ends at the end of the trial.

In [10]:
```
fig, ax = plt.subplots()
ax.vlines(dat['spiketime'], 0, 1)
ax.axvspan(stim_on_time, stim_off_time, alpha=0.5, color='grey')
ax.set_xlim([0,trial_length] )
ax.set_xlabel('Time (ms)')
ax.set_title('Spike Times')
plt.show()
```

Out[10]:



Q8

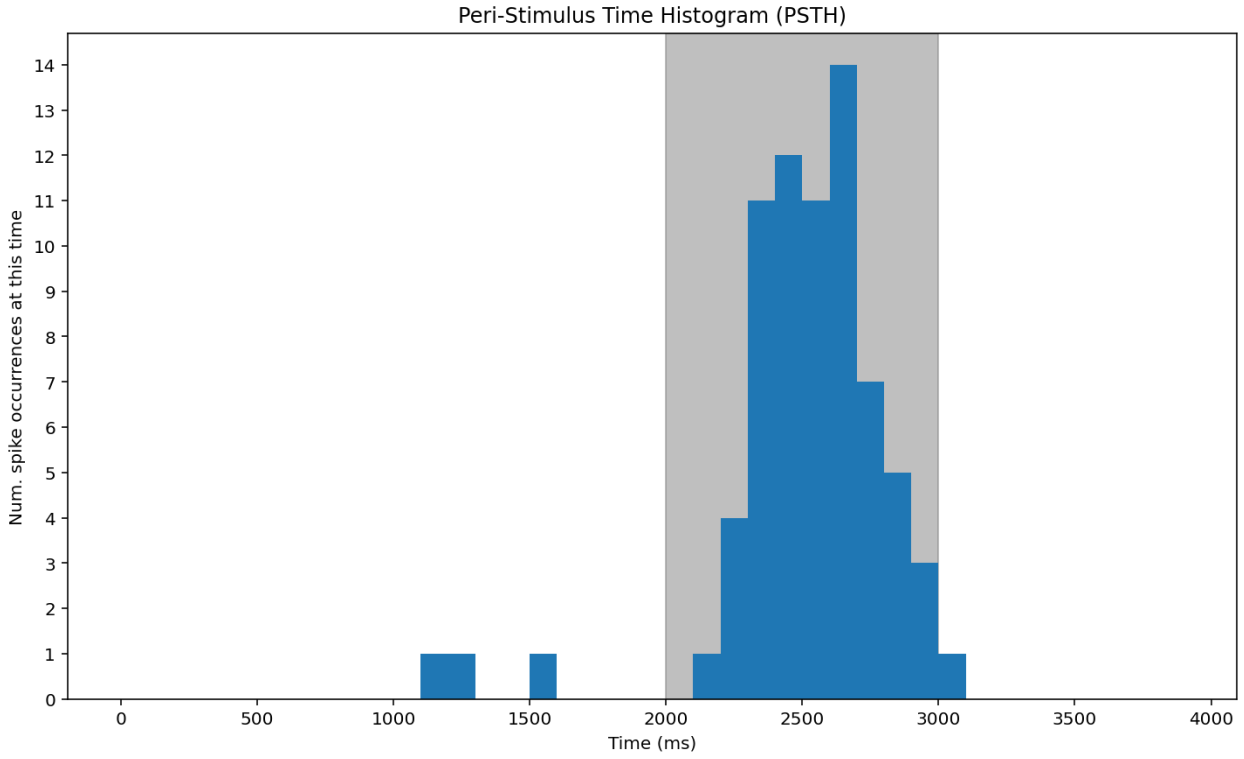
PSTH

Draw a peri-stimulus time histogram for the spike times in `dat`. Use 100 ms bins. Add `grey` shading during the time when the main grating stimulus was on.

--SIDE NOTE-- Stimulus timing stim_on_time = 2000 stim_off_time = 3000 adapt_on_time = 0 adapt_off_time = 2000

In [11]:
```
# Draw the PSTH using histogram since we have the values and need a 100ms bin size
# range(min,max,step)

fig, ax = plt.subplots()
ax.axvspan(stim_on_time, stim_off_time, alpha=0.5, color='grey')
plt.yticks(np.arange(0, 100, 1))
ax.hist(dat['spiketime'], bins= range(0,4000,100))

# Details
ax.set_title('Peri-Stimulus Time Histogram (PSTH)')
ax.set_xlabel('Time (ms)')
ax.set_ylabel('Num. spike occurrences at this time')

plt.show()
```

Out[11]:



Q9

Rasters for all conditions and contrasts

Generate a figure with rasters for each condition (as 2 columns of subplots, with control on the left), and all contrast levels (rows of subplots), for neuron `m1_6`. For full marks, do all of the following as well:

- Use variables rather than directly entering numbers, to set the number of rows and columns of subplots.
- Set the size of the figure to be 15 x 15
- Use `grey` shading to indicate when stimuli were on (both the experimental stimuli and, when appropriate, the adaptation stimuli). Use a shading level equivalent to the stimulus contrast in all cases.
- Be sure that the range of the *x* and *y* axes on all subplots are the same, and that the *x* axis spans the entire time range of a trial from start to finish
- Make the tick marks on the *y* axis occur every 2 trials
- Make the *figure* title the name of the neuron
- Make *each column of plots* have a title saying which condition the plots in that column are from (only above each entire column of subplots)
- Make the labels of the left column *y* axes indicate the stimulus contrast, as a number plus the `%` symbol
- Make the labels on the right column *Y* axes all read "Trial Num"
- Make the labels for the *x* axes on the bottom row of subplots only, read "Time (ms)"
- Ensure subplot elements do not overlap

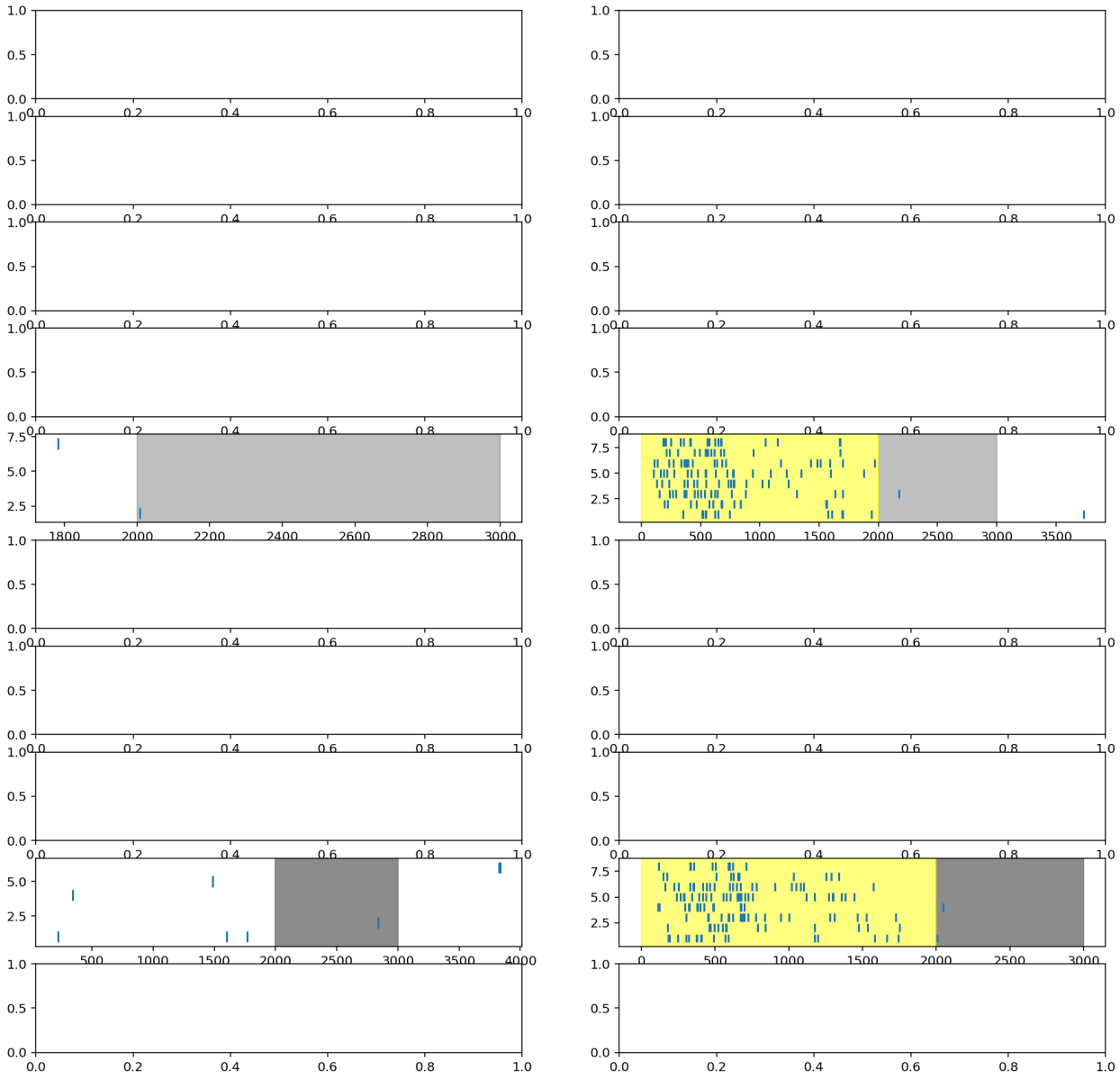
HINT: Much of this is demonstrated in the textbook and just requires a bit of modification. One tip though, that's not shown there — you will find it helpful to use the `enumerate()` function in your `for` loop statement, to loop through contrast levels and conditions (and use these to specify which subplot you're drawing into), rather than a simple `for` loop. Using `enumerate()` is demonstrated elsewhere in the textbook, and of course in the Googleverse.

In [17]:
```
datcontrol =  df[ (df['neuron'] == 'm1_6') & (df['condition'] == 'CTRL')  ]
datadaption =  df[ (df['neuron'] == 'm1_6') & (df['condition'] == 'ADAPT')  ]
int_levels = sorted(datcontrol['contrast'].unique())
trials = sorted(df['repetition'].unique())
############################TEXTBOOK
https://neuraldatascience.io/single_unit/ten_intensities.html
#########################################################

fig, axs = plt.subplots(len(int_levels), 2, figsize=[15, 15])
#Accumulator
y_max = 0

#Loop Start
for i in int_levels:
    datcontrol1 = datcontrol[datcontrol['contrast'] == i]
    datadaption1 = datadaption[datadaption['contrast'] == i]

#Control Raster

    for trial in trials:
        spike_times = datcontrol1[datcontrol1['repetition'] == trial]['spiketime']
        axs[i,0].vlines(spike_times, trial - 0.4, trial + 0.4)
    axs[i,0].axvspan(stim_on_time, stim_off_time, alpha=i / 10 + .1, color='grey')


#Adapt Raster

    for trial in trials:
        spike_times = datadaption1[datadaption1['repetition'] == trial]['spiketime']
        axs[i,1].vlines(spike_times,trial - 0.4, trial + 0.4)
    axs[i,1].axvspan(stim_on_time, stim_off_time, alpha=i / 10 + .1, color='grey')
    axs[i,1].axvspan(adapt_on_time, adapt_off_time, alpha=.5, color='yellow')
```

```
#### Loop end

fig.suptitle('Neuron m1_6 RASTERS')
plt.tight_layout()
plt.show()
```

Out[17]:

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
/tmp/ipykernel_718/1269279342.py in <module>
     18     for trial in trials:
     19         spike_times = datcontrol1[datcontrol1['repetition'] == trial]
['spiketime']
---> 20         axs[i,0].vlines(spike_times, trial - 0.4, trial + 0.4)
     21     axs[i,0].axvspan(stim_on_time, stim_off_time, alpha=i / 10 + .1,
color='grey')
     22
IndexError: index 12 is out of bounds for axis 0 with size 10
```



In [18]:

```
spike_times
```

Out[18]:

```
10      88
11    1753
12    2910
Name: spiketime, dtype: int64
```

USELESS CODE LOOK AT THE FORMATING

datcontrol = df[(df['neuron'] == 'm1_6') & (df['condition'] == 'CTRL')] datadaption = df[(df['neuron'] == 'm1_6') & (df['condition'] == 'ADAPT')] int_levels = sorted(datcontrol['contrast'].unique())

##

#FIGURES fig, axs = plt.subplots(len(int_levels), 2, figsize=[15, 15], sharex=True) y_max = 0 # Accumulator variable

for i in int_levels: ## Raster Control tempdat = datcontrol[datcontrol['contrast'] == i]

```
# Draw the raster one trial at a time
for trial in trials:
    # get spike times for this trial
    spike_times = tempdat[tempdat['repetition'] == trial]['spiketime']
    # Draw the raster
    axs[i, 0].vlines(spike_times, trial -.4 , trial +.4)

# Shade time when stimulus was on
axs[i, 0].axvspan(stim_on_time, stim_off_time,
                  alpha= i / 10 + .1,
                  color='grey')

# Label the y axis with intensity level
axs[i, 0].set_ylabel('Contrast  ' + str(i))

#title above the first row of plots:
if i == 0:
    axs[i, 0].set_title('Raster Control Contrast', fontsize=10)


## Raster Adaption
tempdat = datadaption[datadaption['contrast'] == i]
 for trial in trials:
    # get spike times for this trial
    spike_times = tempdat[tempdat['repetition'] == trial]['spiketime']
    # Draw the raster
    axs[i, 0].vlines(spike_times, trial -.4 , trial +.4)
# Shade time when stimulus was on
axs[i, 1].axvspan(adapt_on_time, adapt_off_time,
                  alpha= i / 10 + .1,
                  color='grey')
```

Set the x tickmarks to every 2 ms axs[i, 1].set_xticks(range(0, trial_length + 1, 2)) #y axis axs[i, 1].set_ylabel('Trail #') # y max of current plot cur_min, cur_max = axs[i, 1].get_ylim() # update if cur_max > y_max: y_max = cur_max # title above first row of plots: if i == 0: axs[i, 1].set_title('Raster Adaption Trails', fontsize=10) # x label below bottom row of plots: if i == max(int_levels): axs[i, 1].set_xlabel('Time (ms)') axs[i, 0].set_xlabel('Time (ms)')

Rescaler

for a in int_levels: axs[a, 0].set_ylim(0, trials) axs[a, 1].set_ylim(0, y_max)

title

fig.suptitle('neuron m1_6') plt.tight_layout() plt.show()

--SIDE NOTE-- Stimulus timing stim_on_time = 2000 stim_off_time = 3000 adapt_on_time = 0 adapt_off_time = 2000

Q10

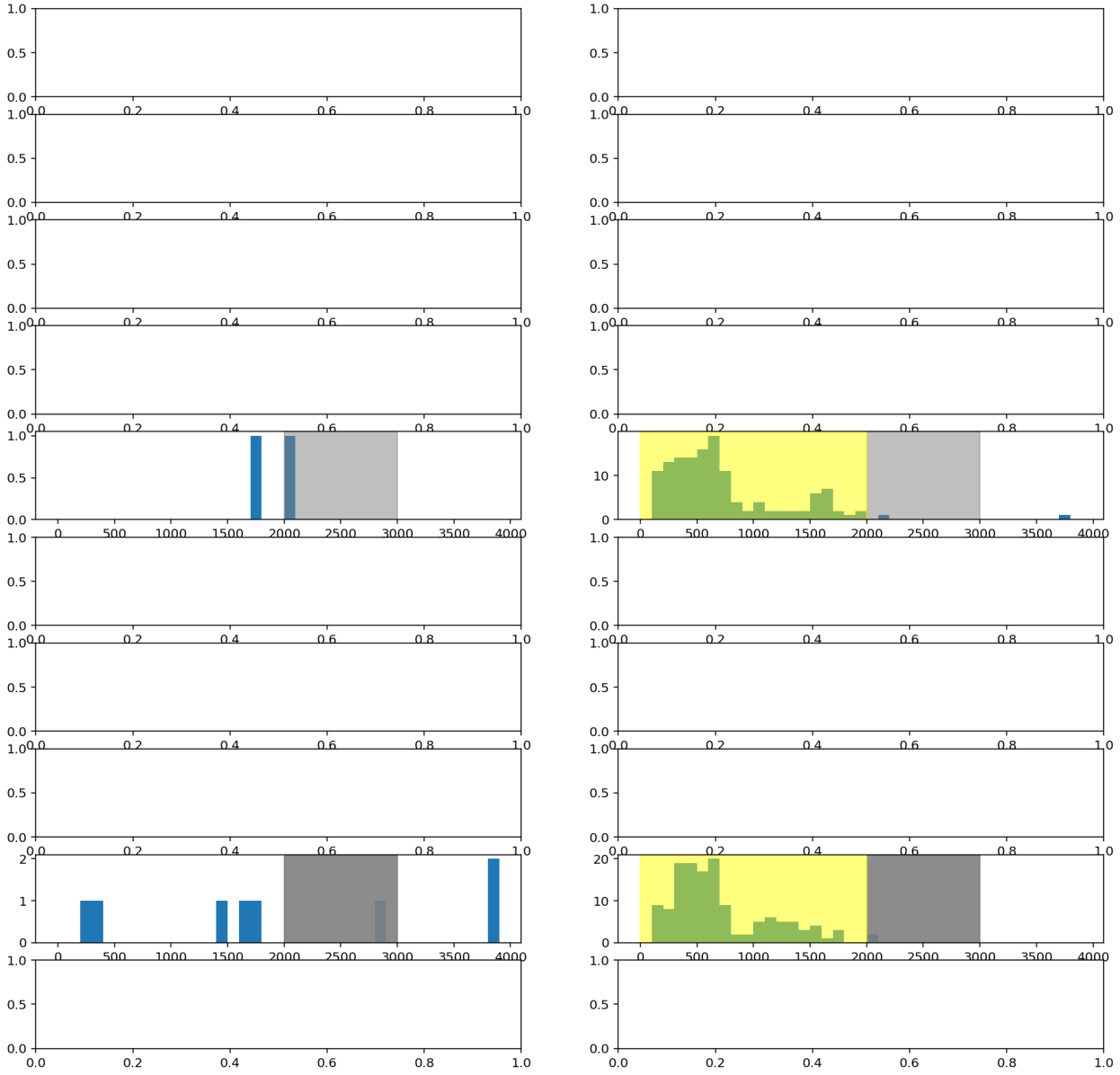
PSTH for all conditions and contrasts

Generate a figure for all conditions (columns) and contrast levels (rows), nearly identical to the raster plots above. The only changes should be:

- plot PSTHs for each condition/contrast rather than rasters
- label the right-hand column *y* axes as "number of spikes"
- Ensure that the range of values on all *y* axes is the same, with a minimum of zero and a maximum corresponding to the maximum number of spikes in any histogram bin in the plot

In [43]:
```
fig, axs = plt.subplots(len(int_levels), 2, figsize=[15, 15])
#Accumulator
y_max = 0

#Loop Start
for i in int_levels:
    datcontrol1 = datcontrol[datcontrol['contrast'] == i]
    datadaption1 = datadaption[datadaption['contrast'] == i]

## PSTH Control



    axs[i, 0].hist(datcontrol1['spiketime'], bins=range(0,4000,100))
    axs[i,0].axvspan(stim_on_time, stim_off_time, alpha=i / 10 + .1, color='grey')



## PSTH Adapt


    axs[i, 1].hist(datadaption1['spiketime'], bins=range(0,4000,100))
    axs[i,1].axvspan(stim_on_time, stim_off_time, alpha=i / 10 + .1, color='grey')
    axs[i,1].axvspan(adapt_on_time, adapt_off_time, alpha=.5, color='yellow')


plt.tight_layout()
plt.show()
```

Out[43]:
```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
/tmp/ipykernel_347/159570377.py in <module>
     12
     13
---> 14     axs[i, 0].hist(datcontrol1['spiketime'], bins=range(0,4000,100))
     15     axs[i,0].axvspan(stim_on_time, stim_off_time, alpha=i / 10 + .1,
color='grey')
     16
IndexError: index 12 is out of bounds for axis 0 with size 10
```



################## "Lasciate ogne speranza, voi ch'entrate"!!!

y_max = 0 # Accumulator variable fig, axs = plt.subplots(len(int_levels), 2, figsize=[15, 15])

for i in int_levels: ## PSTH contorl # Shade time when stimulus was on axs[i, 0].axvspan(stim_on_time, stim_off_time, alpha= i / 10 + .1, color='greenyellow')

```
axs[i, 0].hist(datcontrol['spiketime'], bins=range(0, 4000, 1))


## PSTH Adapt
# Shade time when stimulus was on
axs[i, 1].axvspan(adapt_on_time, adapt_off_time,
                  alpha= i / 10 + .1,
                  color='greenyellow')
axs[i, 1].set_ylabel('Num. Spikes')
axs[i, 1].hist(datadaption['spiketime'], bins=range(0, 4000, 1))
```

for a in int_levels: axs[a, 0].set_ylim(0, 4000) axs[a, 1].set_ylim(0, y_max) fig.suptitle('PSTH Graphs') plt.tight_layout() plt.show()

Q11

Pre-compute histograms for heat maps

Continuing to work with the same neuron, we're now going to make two heat maps. In this first step we'll create the histograms we need for this. In the cell below, create a 3-dimensional NumPy array, `hist`, and fill it with PSTHs. Use the same number and width of bins as above (i.e., spanning the whole time range of a trial, with 100 ms bin widths)

- Start by creating a NumPy array full of zeros, called `hist`, with:
    - the first dimension (row) length equal tot he number of contrast levels
    - the second dimension (column) length equal to the number of histogram bins, minus one
    - the third dimension length equal to the number of conditions
- Then, loop through contrasts and conditions (again, `enumerate()` is your friend) and fill the NumPy array with PSTHs
- As you might expect, indexing a 3D NumPy array is a lot like a 2D one, except you need a third argument inside the square brackets, for the third dimension

In [12]:
```
hist = np.zeros((10, 3999, 2))
```

Run this to check your work

Check the dimensions of the histogram against what you would expect given the number of contrasts, bins, and conditions. The arrays should not contain all zeros.

In [13]:
```
print('Histogram dimensions:', hist.shape)
print()
print(conditions[0])
print(hist[:, :, 0])
print()
print(conditions[1])
print(hist[:, :, 1])
```

Out[13]:
```
Histogram dimensions: (10, 3999, 2)

CTRL
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]

ADAPT
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

Q12

Plot heat maps

Fill in the blanks and add lines to the cell below to:

- Make a figure with two subplots in one row, and a size of 15 x 15
- Loop through conditions (with `enumerate()`) and generate a heat map for each. Use `i, cond` as your loop variables (this will ensure the code provided work rights)
- In the left subplot, show the heat map for the control condition, and on the right, for the adaptation condition
- for each heat map, include the following kwargs (plus others to meet the instructions here):
    - `aspect=2, vmin=0, vmax=np.amax(hist)`
    - This will make your heat maps a nice shape (more square than the default), and it will scale the colour range to the maximum number of spikes in the array
- Choose a colour scale that is isoluminant and perceptually uniform
- Use bicubic interpolation
- The *y* axis of each heat map should be contrast, with the lowest contrast at the top and 100% contrast at the bottom (like the rasters and PSTHs above)
- Label the left subplot's *y* axis as "Stimulus Contrast". No label for the right subplot
- *Do not* add a colourbar to the figure. With subplots, this gets messy

In [14]:
```
#fig, axs = plt.subplots(1, 2, figsize=[15, 15])

#for i, cond in ____:
#     ___.imshow(___,
#                ___,
#                ___,
#                aspect=2,
#                vmin=0, vmax=np.amax(hist)
#               )

#     # Label x axis

#     # Label y axis for left column subplot


#     # Don't change the lines below
#     # We have to be a bit tricky to map the contrast levels and time points onto
the
#     #   appropriate rows and columns of the plot
#     axs[i].axvline(np.where(bins==stim_on_time), color='white', linestyle='--')
#     axs[i].axvline(np.where(bins==stim_off_time), color='white', linestyle='--')

#     axs[i].set_xticks(range(0, len(bins), 5))
#     axs[i].set_xticklabels(range(0, trial_length, 500))

#     # Plot only every third contrast level
#     axs[i].set_yticks(range(0, len(contr_labels), 3))
#     axs[i].set_yticklabels(contr_labels[::3])

# plt.tight_layout()
# plt.show()

# YOUR CODE HERE
raise NotImplementedError()
```

Out[14]:
```
---------------------------------------------------------------------------
NotImplementedError                       Traceback (most recent call last)
```

```
/tmp/ipykernel_1121/2522706793.py in <module>
     31
     32 # YOUR CODE HERE
---> 33 raise NotImplementedError()

NotImplementedError:
```

Q13

Loop through neurons

Now create a figure containing heat maps for *all* neurons. As above, put the control condition in the left column and adapt in the right column, and use one row of heat maps for each neuron.

Be sure to do the following:

- Create a loop to cycle through neurons
- Inside this loop, first create a NumPy array of histograms for the current neuron
- Then, plot the heat maps
- Most aspects of the heat maps and overall figure should be the same as above. However, each subplot's title should be the name of the neuron, plus the condition

In [15]:
```
# YOUR CODE HERE
raise NotImplementedError()
```

Out[15]:
```
---------------------------------------------------------------------------
NotImplementedError                       Traceback (most recent call last)
/tmp/ipykernel_1121/1067236896.py in <module>
      1 # YOUR CODE HERE
----> 2 raise NotImplementedError()

NotImplementedError:
```

Interpretation

Recall that the predictions we made for this experiment were:

Hypotheses

1. In the control condition, latency to first spike is expected to decrease with increasing contrast.
2. Some primary visual cortex cells are phase sensitive (so-called **simple cells**), so in response to a drifting sine wave grating they will oscillate between excitation and inhibition at the same temporal frequency of the stimulus (2 Hz in this case).

What evidence do you see in the data for each of these hypotheses?

Make reference to the raster plots, PSTHs, and heat maps in your answers

Q14

Hypothesis 1

Does spike latency decrease with increasing contrast? If so, is this for all or only some neurons? If only some neurons, which ones? What evidence shows this? Be clear which figure(s) and subplot(s) you're referring to in your answer.

Q15

Hypothesis 1b

Does contrast influence spike rate? If so, in what way(s)? What evidence shows this?

YOUR ANSWER HERE

Q14

Hypothesis 2

Which neurons appear to be simple cells, and why?

YOUR ANSWER HERE

Q15

Effects of adaptation stimulus

What is the effect of the 50% contrast adaptation stimulus on a neuron's subsequent response to the presentation of the "main" stimulus at 2000 ms?

YOUR ANSWER HERE

The End