Computing_a_tuning_curve

March 26, 2021

This notebook walks us through how to compute a tuning curve using the Allen Brain Observatory data

This notebook has two parts. The first walks through computing a tuning curves for cells in response to a drifting grating stimulus. This is a classic single-cell analysis, and students can compare the tuning of several cells from the same experiment. The second part looks at correlations between cells that have the same tuning to the drifting grating stimulus from the same experiment. This highlights the fact that our dataset has populations of cells, simultaneously imaged, and allows for examinations of how those cells interact with each other. This notebook is designed to only reference a single experiment from the Allen Brain Observatory, thus requires only one NWB file to be downloaded. (as downloading NWB files can be time consuming).

0.0.1 Standard Imports

[2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import sys, os
```

0.0.2 Brain Observatory set up

[3]:
```
from allensdk.core.brain_observatory_cache import BrainObservatoryCache

#Set manifest path when outside of AWS
drive_path = '/data/allen-brain-observatory/visual-coding-2p/'
manifest_file = os.path.join(drive_path,'manifest.json')
print(manifest_file)

boc = BrainObservatoryCache(manifest_file=manifest_file)
```

```
/data/allen-brain-observatory/visual-coding-2p/manifest.json
```

0.1 Computing tuning curve for the drifting grating stimulus

We need:

fluorescence trace for our cell. We will use the DFF trace

stimulus information for the drifting grating stimulus

[97]:
```
cell_id = 541513979
```

I've created a function here that will return the DF/F trace and the stimulus table if we provide a cell id and a stimulus name. This function leverages functions in the AllenSDK, and there are examples that walk through these steps in other notebooks on the SDK page. I'm happy to explain the steps in greater detail if anyone is interested.

The key things to know about this. You provide the function with a cell_specimen_id and a stimulus name. It returns:

the timestamps for the DF/F trace. This is a numpy array.

the trace of that cell's DF/F trace for the whole session. This is also a numpy array.

a table that describes the stimulus conditions and timing. This is a pandas dataframe.

If you aren't working in AWS, this function can take some time the first time you run it for a given NWB file as it will require that file to be downloaded from the warehouse.

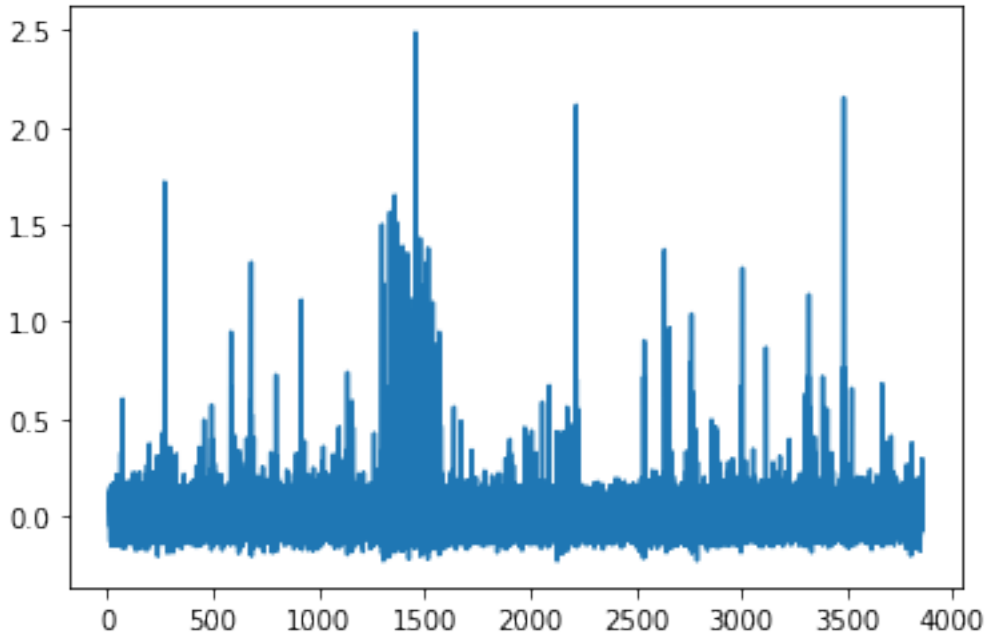
[5]:
```
def get_dff_traces_and_stim_table(cell_specimen_id, stimulus):
    #identify the session for a given cell id and stimulus
    exps = boc.get_ophys_experiments(cell_specimen_ids=[cell_specimen_id],␣
 ↪stimuli=[stimulus])
    #get the session_id for that session
    session_id = exps[0]['id']
    #access the data for that session
    data_set = boc.get_ophys_experiment_data(session_id)
    #get the DFF trace for the cell
    timestamps, dff = data_set.
 ↪get_dff_traces(cell_specimen_ids=[cell_specimen_id])
    dff_trace = dff[0,:]
    #get the stimulus table for the stimulus
    stim_table = data_set.get_stimulus_table(stimulus)
    #return everything
    return (timestamps, dff_trace, stim_table)
```

[6]:
```
timestamps, dff_trace, stim_table = get_dff_traces_and_stim_table(cell_id,␣
 ↪'drifting_gratings')
```

First let's plot the DF/F trace of our cell to see what it looks like

[7]:
```
plt.plot(timestamps,dff_trace)
```

[7]:
```
[<matplotlib.lines.Line2D at 0x7f218873bb50>]
```



Now let's look at our stimulus table to see what information we have. We just want to see the first few lines, so use the function head to see the top of this DataFrame.

[101]:
```
stim_table.head()
```

[101]:

	temporal_frequency	orientation	blank_sweep	start	end
0	1.0	45.0	0.0	736	796
1	15.0	90.0	0.0	826	886
2	1.0	270.0	0.0	917	977
3	NaN	NaN	1.0	1007	1067
4	4.0	90.0	0.0	1098	1158

The stimulus table has 5 columns. Start and end indicate the frame number when a given grating condition starts and ends, respectively. The other columns indicate what the grating codition is, including the temporal frequency of the grating (in Hz), the direction (called orientation) of the grating (in degrees), and whether the grating is a blank sweep (eg. a gray screen).

Pandas is a very useful python module for data analysis, which has an object called a DataFrame that is a flexible and powerful tool for analyzing large datasets. I highly encourage interested students to explore this analysis module. But for our purposes, we will only use it to access the stimulus information.

0.2 Pandas

Quick pandas tutorial for our purposes today!

To access data from a DataFrame we must specify the column we are using and specify the row using the index. To specify a column we can use two methods:

stim_table['start']

stim_table.start

Then to specify the row we want we must use the index of that row.

stim_table['start'][0]

stim_table.start[0]

We can also subselect portions of the DataFrame using the values in the DataFrame. For example, to select only the rows of the table where the orientation is 90 degrees we can use:

stim_table[stim_table.orientation==90]

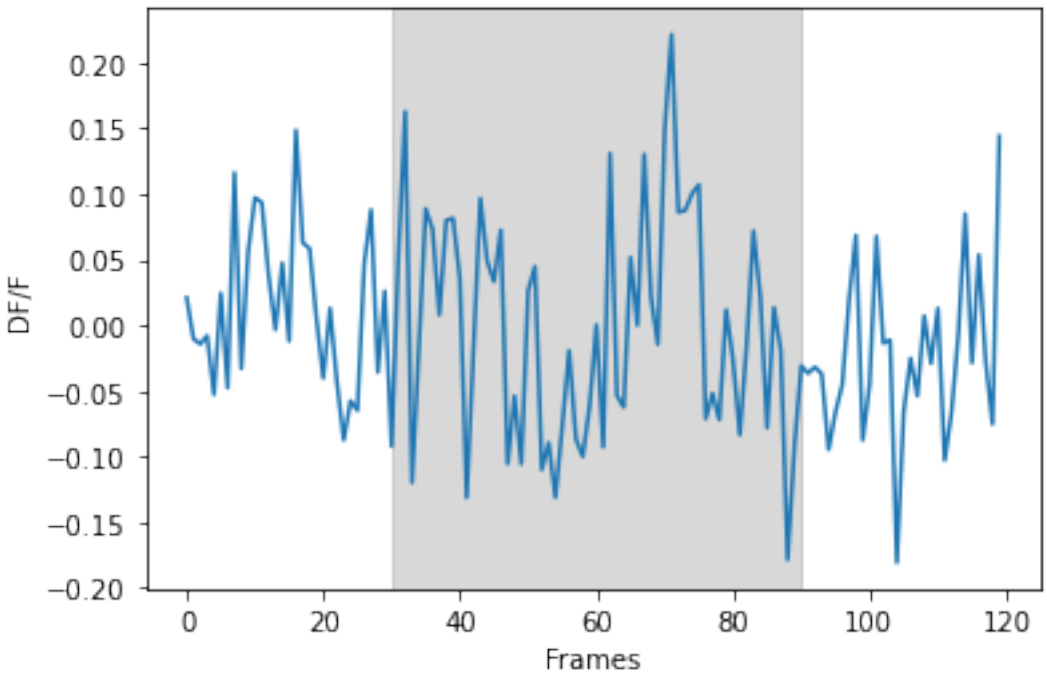
Try this yourself. Note the index. The rows of this subselected DataFrame maintain the indices of the original DataFrame. Now in order to get a specific row, you either need to know it's original index, or use iloc. For example, this will return the first row of the subselected DataFrame regardless of the original index of that row:

stim_table[stim_table.orientation==90].iloc[0]

To look at the cell's response to a given grating presentation, let's plot the DF/F of the cell during the presentation of that grating. We want to pad the plot with ~ 1 second of the DF/F trace preceding the grating presentation. 1 second = 30 frames. We'll plot the response to the first grating presentation.

[8]:
```
plt.plot(dff_trace[stim_table.start[0]-30:stim_table.end[0]+30])
plt.axvspan(30,90, color='gray', alpha=0.3) #this shades the period when the␣
 ↪stimulus is being presented
plt.ylabel("DF/F")
plt.xlabel("Frames")
```

[8]:
```
Text(0.5, 0, 'Frames')
```



We want to quantify this response. We can explore different methods of quantifying this. * mean DF/F during the grating presentation * sum of the DF/F during the grating presentation (are these different?) * maximum DF/F during grating

For now let's use the mean DF/F during the presentation of the grating.

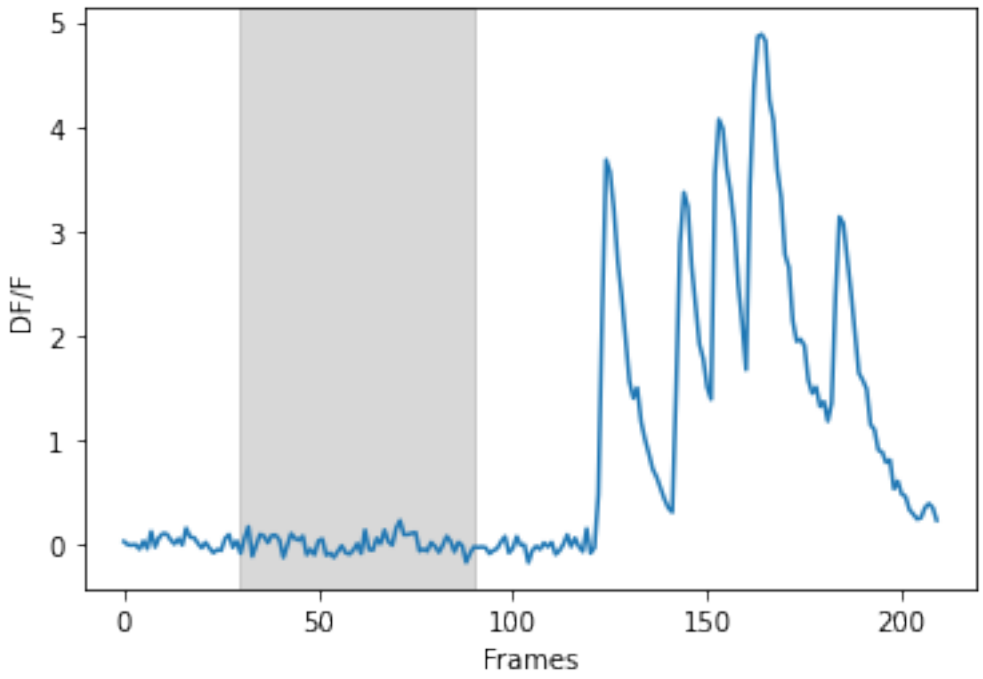
[9]:
```
dff_trace[stim_table.start[0]:stim_table.end[0]].mean()
```

[9]:
```
-0.0020014683
```

Exercise 1: Repeat this for the next grating stimulus

[10]:
```
plt.plot(dff_trace[stim_table.start[0]-30:stim_table.end[1]+30])
plt.axvspan(30,90, color='gray', alpha=0.3) #this shades the period when the␣
 ↪stimulus is being presented
plt.ylabel("DF/F")
plt.xlabel("Frames")
```

[10]:
```
Text(0.5, 0, 'Frames')
```



[11]:
```
dff_trace[stim_table.start[1]:stim_table.end[1]].mean()
```

[11]: 2.292613

Breakout 1
</div

Already we can see that some stimulus conditions elicit larger responses than others. This is what we want to quantify.

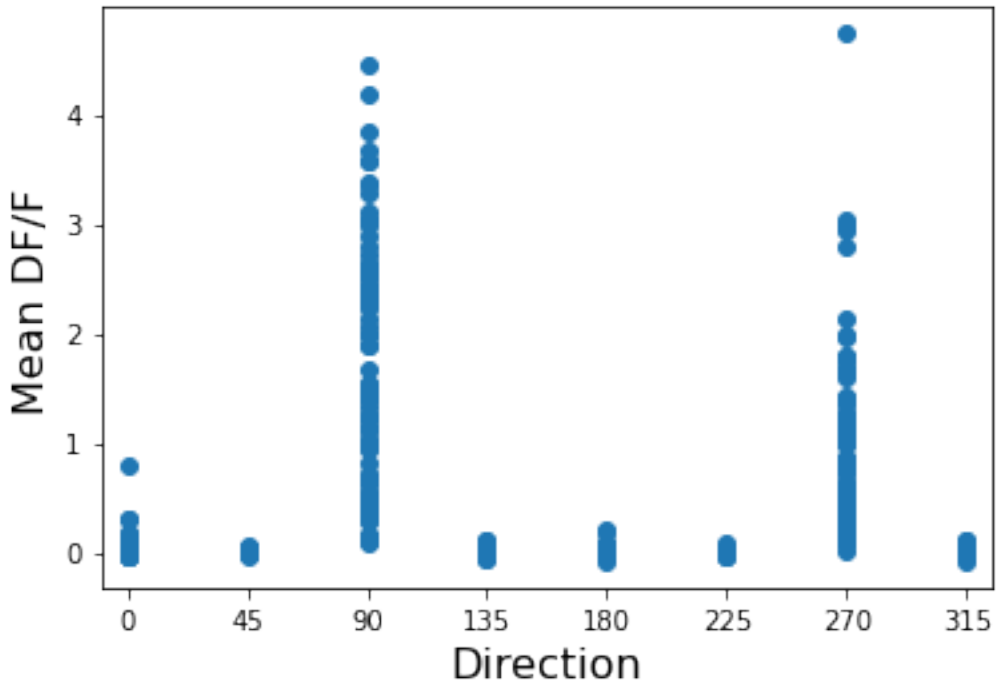
To do this, let's calculate the mean DF/F for each grating presentation in this stimulus. To start, let's create a numpy array to hold our calculated responses. We'll have three columns, one for the stimulus orientation, one for temporal frequency, and the last for the response. Then we need to iterate over all stimulus trials, populate the orientation and TF and then calculate the mean response.

[16]:
```
cell_response= np.zeros((len(stim_table),3))
for i in range(len(stim_table)):
    cell_response[i,0] = stim_table.orientation[i]
    cell_response[i,1] = stim_table.temporal_frequency[i]
    cell_response[i,2] = dff_trace[stim_table.start[i]:stim_table.end[i]].mean()
```

If we only care about one stimulus parameter, we can quickly compare the response to that parameter, say the direction. Here we will plot each grating response as a function of the grating orientation.

[17]:
```
plt.plot(cell_response[:,0], cell_response[:,2], 'o')
plt.xticks(range(0,360,45))
plt.xlim(-10,325)
plt.xlabel("Direction", fontsize=16)
plt.ylabel("Mean DF/F", fontsize=16)
```

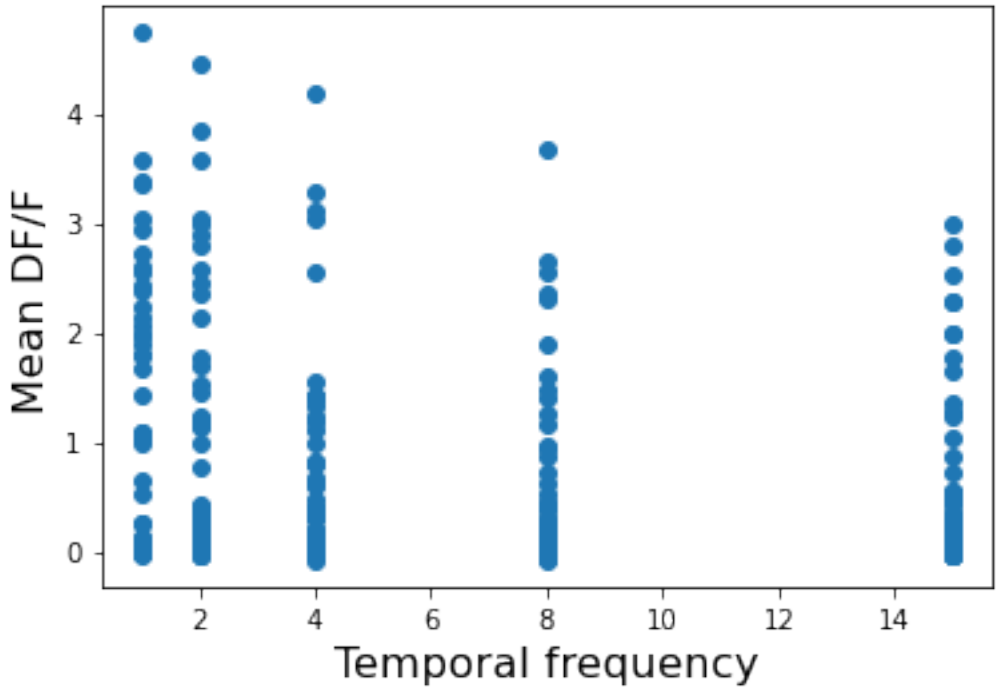
[17]: Text(0, 0.5, 'Mean DF/F')



Exercise 2: Repeat this for the temporal frequency parameter

[18]:
```
plt.plot(cell_response[:,1], cell_response[:,2], 'o')
plt.xlabel("Temporal frequency", fontsize=16)
plt.ylabel("Mean DF/F", fontsize=16)
```

[18]: Text(0, 0.5, 'Mean DF/F')



We want to quantify this more explicitly. So let's average all of the responses to each orientation together. This is the mean DF/F response to an orientation, for all temporal frequencies, for all trials. For example, for orientation=270: (Hint: use np.where)

[19]:
```
trials = np.where(cell_response[:,0]==270)[0]
cell_response[trials,2].mean()
```

[19]:
```
0.9247032543023427
```

Exercise 3: Compute and plot the mean response as a function of orientation

To start, you need to know what all the possible orientation values are. You can either find this from the website, or you can find the unique values that are not NaNs (eg. values that are finite)

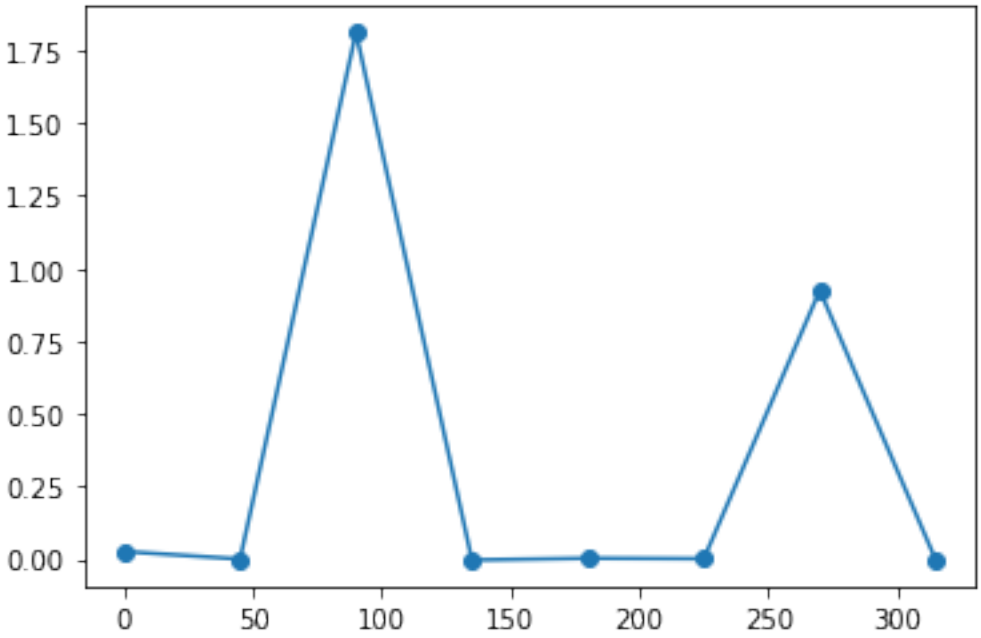
[20]:
```
all_ori = np.unique(cell_response[:,0])
orivals = all_ori[np.isfinite(all_ori)]
print(orivals)
```

```
[  0.  45.  90. 135. 180. 225. 270. 315.]
```

[21]:
```
tuning = np.empty((8))
for i, ori in enumerate(orivals):
    trials = np.where(cell_response[:,0]==ori)[0]
    tuning[i] = cell_response[trials,2].mean()
```

[22]:
```
plt.plot(orivals,tuning, 'o-')
```

[22]: [<matplotlib.lines.Line2D at 0x7f08ccd23810>]



Exercise 4: Compute and plot the mean response as a function of temporal frequency for all orientations.

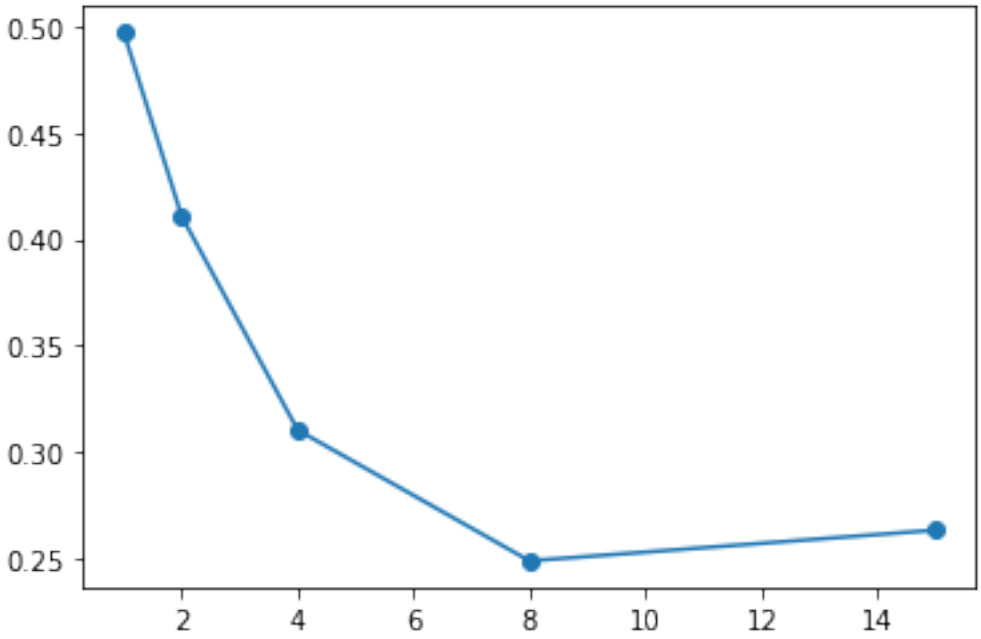
[23]:
```
tfvals = np.unique(cell_response[:,1])
tfvals = tfvals[np.isfinite(tfvals)]
print(tfvals)
```

```
[ 1.  2.  4.  8. 15.]
```

[24]:
```
tuning_tf = np.empty((len(tfvals)))
for i,tf in enumerate(tfvals):
    trials = np.where(cell_response[:,1]==tf)
    tuning_tf[i] = cell_response[trials,2].mean()
```

[25]:
```
plt.plot(tfvals, tuning_tf, 'o-')
```

[25]: [<matplotlib.lines.Line2D at 0x7f08eaab4fd0>]

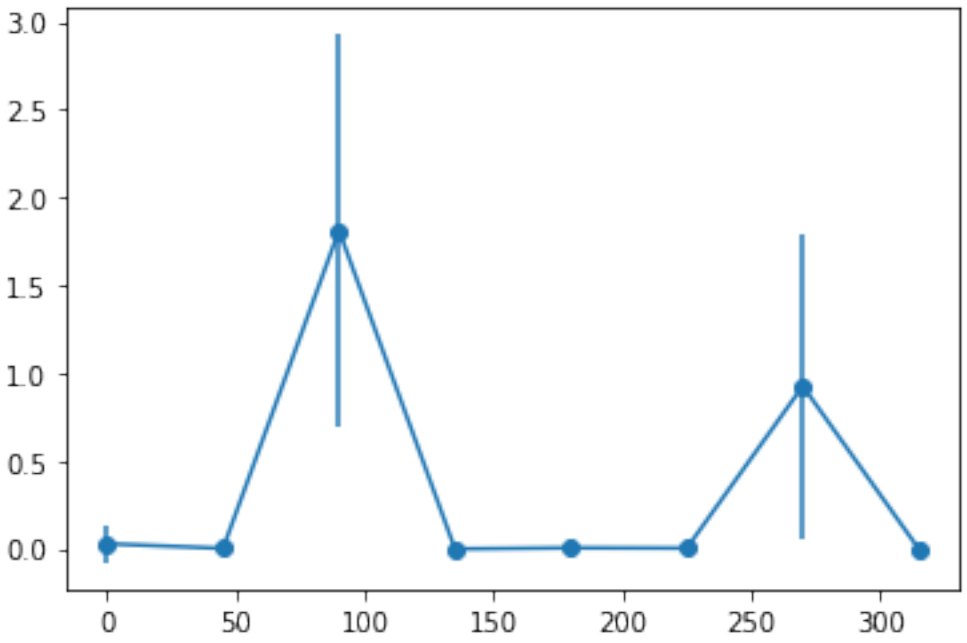


Exercise 5: Add errorbars to the above tuning curves. They can be standard deviation or standard error or the mean. (Hint: plt.errorbar might be a useful function).

[26]:
```
tuning = np.empty((8,2))
for i, ori in enumerate(orivals):
    trials = np.where(cell_response[:,0]==ori)[0]
    tuning[i,0] = cell_response[trials,2].mean()
    tuning[i,1] = cell_response[trials,2].std()
```

[27]:
```
plt.errorbar(orivals, tuning[:,0], yerr=tuning[:,1], fmt='o-')
```

[27]:
```
<ErrorbarContainer object of 3 artists>
```



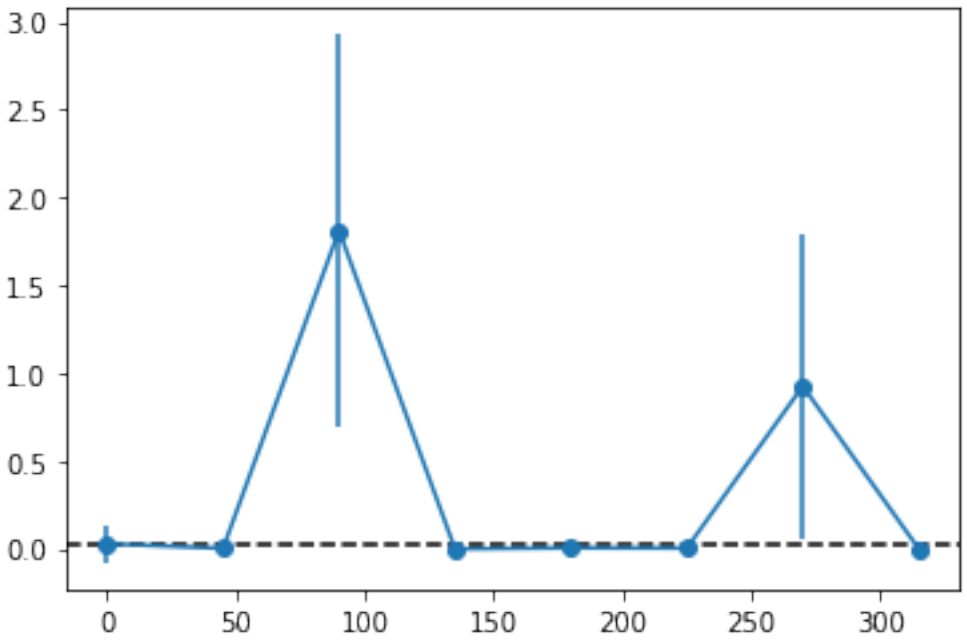
Exercise 6: Add a black line showing the mean response to the blank sweep (Hint 1: orientation and temporal frequency are NaN for the blank sweep condition. Hint 2: plt.axhline might be a useful function).

[28]:
```
blank_trials = np.isnan(cell_response[:,0])
blank_mean = cell_response[blank_trials,2].mean()
print(blank_mean)
```

```
0.019221892748991477
```

[29]:
```
plt.errorbar(orivals, tuning[:,0], yerr=tuning[:,1], fmt='o-')
plt.axhline(y=blank_mean, ls='--', color='k')
```

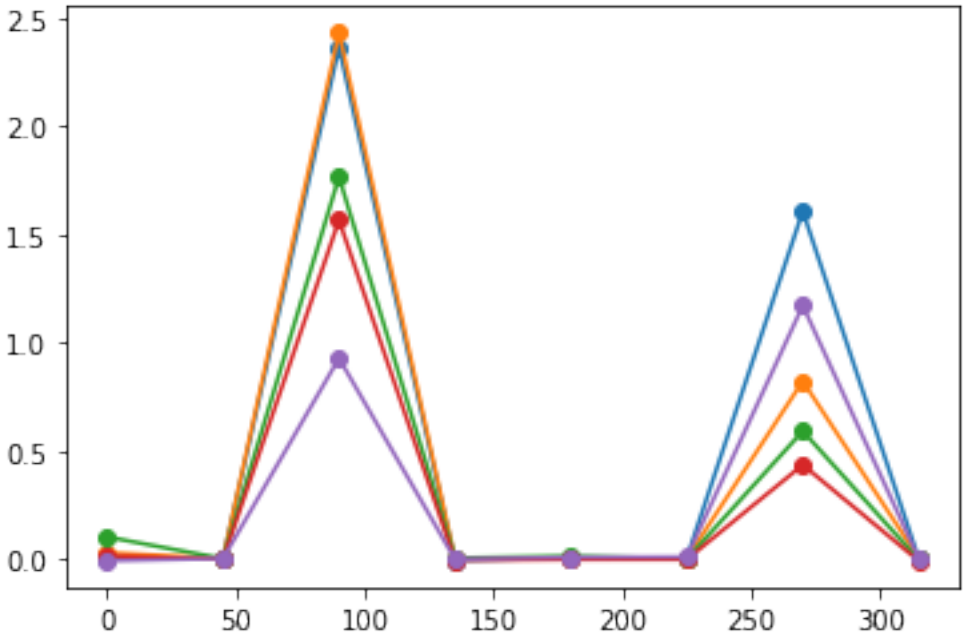
[29]: `<matplotlib.lines.Line2D at 0x7f08d0035fd0>`



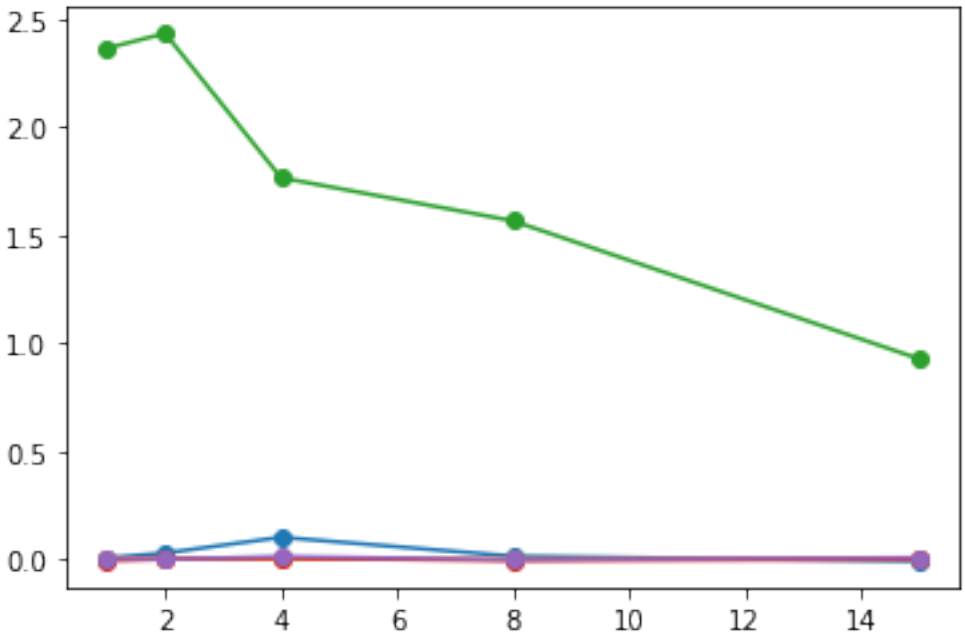
Exercise 7: Compute and plot the direction tuning curve separately for each of the 5 temporal frequencies.

[33]:
```
tuning_array = np.empty((8,5))
for i,tf in enumerate(tfvals):
    tf_trials = np.where(cell_response[:,1]==tf)[0]
    subset = cell_response[tf_trials]
    for j,ori in enumerate(orivals):
        trials = np.where(subset[:,0]==ori)
        tuning_array[j,i] = subset[trials,2].mean()
```

[34]:
```
for i in range(5):
    plt.plot(orivals, tuning_array[:,i], 'o-')
```



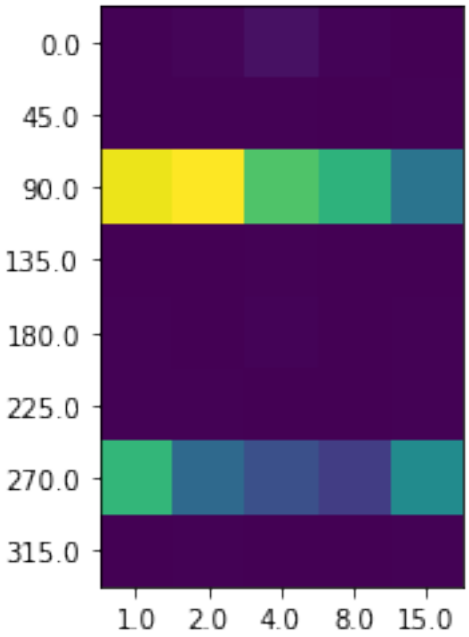
[35]:
```
for i in range(5):
    plt.plot(tfvals, tuning_array[i,:], 'o-')
```



Exercise 8: Are there other ways to visualize these tuning responses?

[36]:
```
plt.imshow(tuning_array)
plt.xticks(range(5), tfvals)
plt.yticks(range(8), orivals)
```

[36]:
```
([<matplotlib.axis.YTick at 0x7f08e9cd5550>,
  <matplotlib.axis.YTick at 0x7f08e9de5550>,
  <matplotlib.axis.YTick at 0x7f08e9ccb750>,
  <matplotlib.axis.YTick at 0x7f08e95dc290>,
  <matplotlib.axis.YTick at 0x7f08e95dc7d0>,
  <matplotlib.axis.YTick at 0x7f08e95dcd50>,
  <matplotlib.axis.YTick at 0x7f08e95e52d0>,
  <matplotlib.axis.YTick at 0x7f08e95e5810>],
 [Text(0, 0, '0.0'),
  Text(0, 1, '45.0'),
  Text(0, 2, '90.0'),
  Text(0, 3, '135.0'),
  Text(0, 4, '180.0'),
  Text(0, 5, '225.0'),
  Text(0, 6, '270.0'),
  Text(0, 7, '315.0')])
```



Breakout 2
</div

We've looked at the tuning curve for one cell. There are a lot of other cells in this experiment. Now we're going to see what some of the other cells in this experiment look like, and start the explore how the different cells might interact.

Exercise 9: Compute the tuning curves for cell_ids 541512490, 541512611, 541512645, 541512079, 541511403, 541511670, 541511373, 541513771, 541511385, 541512607. (Hint: it might be helpful to write a function) In what ways do these tuning curves differ? In what ways are they the same? What are interesting parameters of a cell's response to this stimulus?

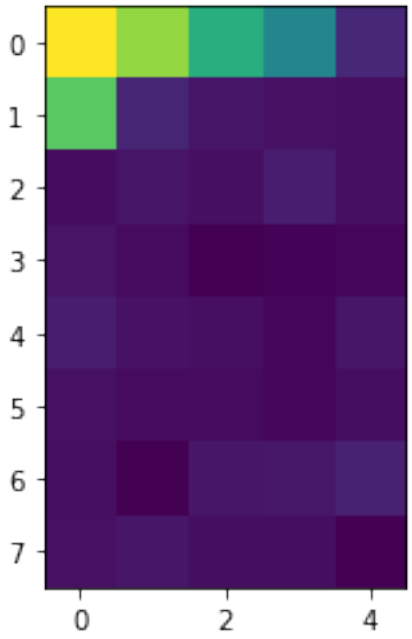
[37]:
```
def compute_ori_tf_tuning(cell_id):
    timestamps, dff_trace, stim_table = get_dff_traces_and_stim_table(cell_id,␣
 ↪'drifting_gratings')

    cell_response= np.zeros((len(stim_table),3))
    for i in range(len(stim_table)):
        cell_response[i,0] = stim_table.orientation[i]
        cell_response[i,1] = stim_table.temporal_frequency[i]
        cell_response[i,2] = dff_trace[stim_table.start[i]:stim_table.end[i]].
 ↪mean()

    tuning_array = np.empty((8,5))
    for i,tf in enumerate(tfvals):
        tf_trials = np.where(cell_response[:,1]==tf)[0]
        subset = cell_response[tf_trials]
        for j,ori in enumerate(orivals):
            trials = np.where(subset[:,0]==ori)
            tuning_array[j,i] = subset[trials,2].mean()
    return tuning_array
```

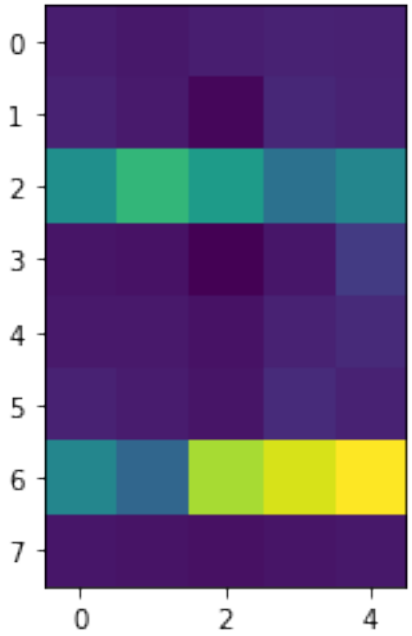
[38]:
```
tuning_a = compute_ori_tf_tuning(541512490)
plt.imshow(tuning_a)
```

[38]:
```
<matplotlib.image.AxesImage at 0x7f08e955cad0>
```



[39]:
```
tuning_b = compute_ori_tf_tuning(541512611)
plt.imshow(tuning_b, interpolation='none')
```

[39]: <matplotlib.image.AxesImage at 0x7f08e5ec9910>



[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

Some of these cells show the same tuning for orientation and temporal frequency. Now let's look at these more closely to see how they are correlated.

If you've overwritten the dff_trace, go back and get the trace for our original cell: cell_id = 541513979

Also get the trace for cell 541511905, let's call this trace "dff_trace_2"

[40]:
```
cell_id = 541513979
timestamps, dff_trace, stim_table = get_dff_traces_and_stim_table(cell_id,␣
↪'drifting_gratings')
```

[41]:
```
new_cell_id = 541511905
timestamps, dff_trace_2, stim_table = get_dff_traces_and_stim_table(new_cell_id,␣
↪'drifting_gratings')
```

These two cells have some similarities in their tuning preferences, but how similar are their responses?

Correlations

We can compute a correlation coefficient to measure the relationship between two variables - in this case the responses of two different cells. When cell A has a large response, does cell B also have a strong response?

We will use Pearson correlation coefficient, which is defined as:

where x and y are the two variables in question. We will use a function from scipy.stats to compute this for us.

If the two variables are positively correlated, a large response from cell A will be matched with a large response in cell B. If they are weakly correlated, the response size of cell A will have not relationship with the response size of cell B. And if they are negatively correlated, cell B will have small responses when cell A has large responses.

Exercise 10: Compute the correlation between these two cells' traces. We're going to use st.pearsonr from scipy.stats

[42]:
```
import scipy.stats as st
```

[43]:
```
r,p = st.pearsonr(dff_trace, dff_trace_2)
print(r)
```

```
0.5628775961334622
```

Exercise 11: Calculate the response of the second cell to all the stimulus trials and compute the correlation between these two cells' responses.

[44]:
```
cell_response_2 = np.zeros((len(stim_table),3))
for i in range(len(stim_table)):
    cell_response_2[i,0] = stim_table.orientation[i]
    cell_response_2[i,1] = stim_table.temporal_frequency[i]
    cell_response_2[i,2] = dff_trace_2[stim_table.start[i]:stim_table.end[i]].
 ↪mean()
```

[45]:
```
r,p = st.pearsonr(cell_response[:,2], cell_response_2[:,2])
print(r)
```

```
0.7640585294224912
```

Exercise 12: Calculate the orientation tuning curve for the second cell and compute the correlation between these two cells' tuning. Hint: you will need to flatten the tuning array.

[46]:
```
tuning_array_2 = np.empty((8,5))
for i,tf in enumerate(tfvals):
    tf_trials = np.where(cell_response_2[:,1]==tf)[0]
    subset = cell_response_2[tf_trials]
    for j,ori in enumerate(orivals):
        trials = np.where(subset[:,0]==ori)
        tuning_array_2[j,i] = subset[trials,2].mean()
```

[47]:
```
r,p = st.pearsonr(tuning_array.flatten(), tuning_array_2.flatten())
print(r)
```

```
0.9347812227928618
```

We've looked at three different correlations. Discuss why these correlations are different and what it might tell you about these two cells.

Exercise 13: Repeat these three correlation computations for cells 541511373 and 541511385. Notice anything similar or different from the previous pair?

[48]:
```
cell_id = 541511373
timestamps, dff_trace_3, stim_table = get_dff_traces_and_stim_table(cell_id,␣
 ↪'drifting_gratings')
cell_id = 541511385
timestamps, dff_trace_4, stim_table = get_dff_traces_and_stim_table(cell_id,␣
 ↪'drifting_gratings')
```

[ ]:
```

```

[ ]:

[ ]:

#Amath 342 HW #8

##Zhiyang (Coco) Cheng

Breakout 3
</div

Everything so far has looked at the responses of these cells to 'driftings_gratings'. Now let's look at their responses to 'natural_movie_three' and see how it compares to their grating responses.

Use the function from the start of this notebook to get the timestamps, dff trace, and stimulus table for our original cell (cell_id = 541513979) for 'natural_movie_three'. Look at the stimulus table. How is it different from the stimulus table for drifting gratings?

[71]:
```
cell_id = 541513979
timestamps, dff_trace, stim_table_nm = get_dff_traces_and_stim_table(cell_id,␣
 ↪'natural_movie_three')
```

[72]:
```
stim_table_nm.head()
```

[72]:

	frame	start	end	repeat
0	0	19746	19747	0
1	1	19747	19748	0
2	2	19748	19749	0
3	3	19749	19750	0
4	4	19750	19751	0

###How is it different from the stimulus table for drifting gratings?

[102]:
```
stim_table.head()
```

[102]:

	temporal_frequency	orientation	blank_sweep	start	end
0	1.0	45.0	0.0	736	796
1	15.0	90.0	0.0	826	886
2	1.0	270.0	0.0	917	977
3	NaN	NaN	1.0	1007	1067
4	4.0	90.0	0.0	1098	1158

###As we can see the stim_table is the stimulus table for drifting gratings, it is different from stimulus table for our cell (cell_id = 541513979)

The movie is repeated 10 times. Each repeat begins when frame 0 is presented. Find the start times for each movie repeat.

[73]:
```
start_times = stim_table_nm[stim_table_nm.frame==0].start.values
```

[74]:
```
start_times
```

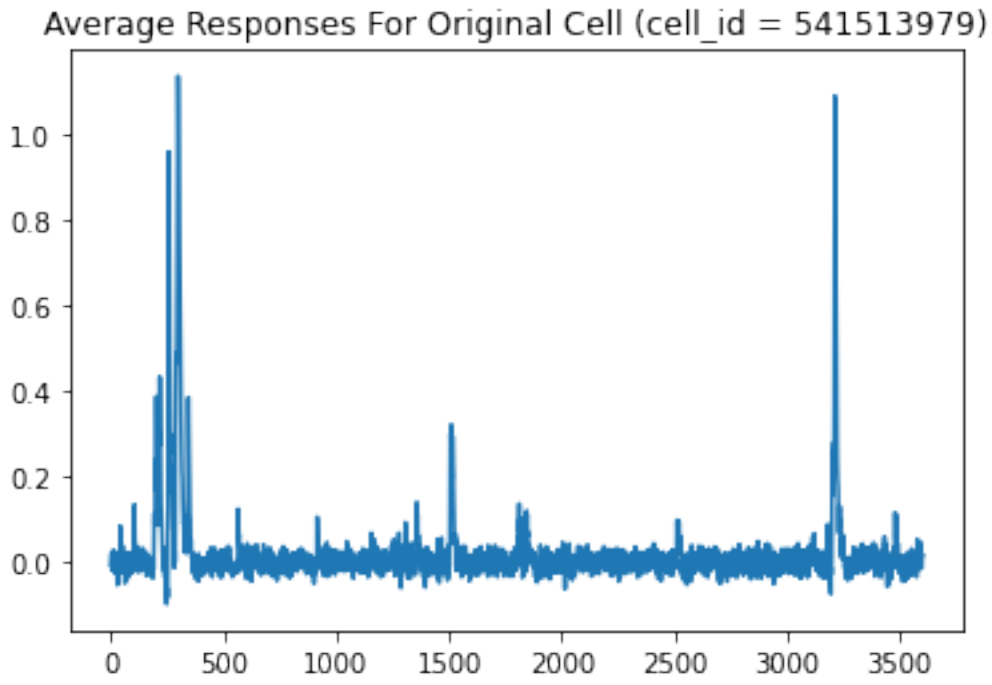
[74]:
```
array([19746, 23367, 26987, 30608, 34229, 75869, 79490, 83110, 86731,
       90353])
```

Now we're going to average each cell's response to the movie. Start from the start time you just found above. The movie is 3600 frames long (you can check that by finding how many unique frames there are in the stimulus table). So make an array that is size (10,3600) and put the response of the cell for each of the 10 trials in the array. Then average across those 10 trials and plot the result. Do this for both of the two cells we looked at in exercises 10-12.

[76]:
```
len(stim_table_nm.frame.unique())
nm_res = np.empty((10,3600))
for i in range(10):
    nm_res[i,:] = dff_trace[start_times[i]:start_times[i]+3600]
ans1 = nm_res.mean(axis=0)
```

[78]:
```
plt.plot(ans1)
plt.title("Average Responses For Original Cell (cell_id = 541513979)")
```

[78]:
```
Text(0.5, 1.0, 'Average Responses For Original Cell (cell_id = 541513979)')
```



[79]:
```
cell_id = 541511905
timestamps, dff_trace, stim_table_nm = get_dff_traces_and_stim_table(cell_id,␣
↪'natural_movie_three')
```

[80]:
```
stim_table_nm.head()
```

[80]:

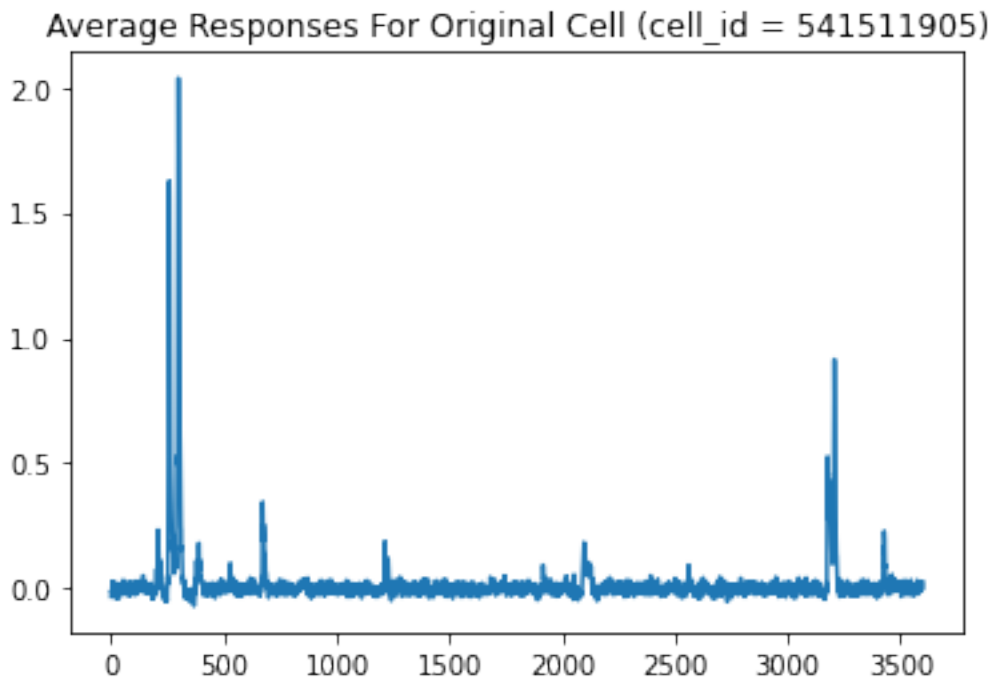
	frame	start	end	repeat
0	0	19746	19747	0
1	1	19747	19748	0
2	2	19748	19749	0
3	3	19749	19750	0
4	4	19750	19751	0

[81]:
```
start_times = stim_table_nm[stim_table_nm.frame==0].start.values
```

[83]:
```
len(stim_table_nm.frame.unique())
nm_res = np.empty((10,3600))
for i in range(10):
    nm_res[i,:] = dff_trace[start_times[i]:start_times[i]+3600]
ans2 = nm_res.mean(axis=0)
```

[84]:
```
plt.plot(ans2)
plt.title("Average Responses For Original Cell (cell_id = 541511905)")
```

[84]:
```
Text(0.5, 1.0, 'Average Responses For Original Cell (cell_id = 541511905)')
```



The two cells we're looking at have similar tuning curves for their drifting grating responses (see above). Do you expect they have similar responses for the natural movie? Why or why not? Compute the correlation between the averaged movie responses of these two cells. How does it

compare to the correlation of their tuning to the drifting gratings? Why might they be similar? Why might they be different?

###As we can see, we know that these two have similar tuning curves for their drifting grating responses. So before plotting, we expect they have similar responses for the natural movie since I think they have similarities in their tuning prefrences.

[59]:
```
r,p = st.pearsonr(ans, ans2)
print(r)
```

```
0.7509293926390078
```

###After computing the correlation between the averaged movie responses of these two cells, we got the number 0.75. Compared to before, I think they are similar because they would be both cosidered as highly correlated.

Do the same comparison for the second pair of cells that you looked at for the drifting gratings, cells 541511373 and 541511385.

[85]:
```
cell_id = 541511373
timestamps, dff_trace, stim_table_nm = get_dff_traces_and_stim_table(cell_id,␣
 ↪'natural_movie_three')
```

[86]:
```
stim_table_nm.head()
```

[86]:

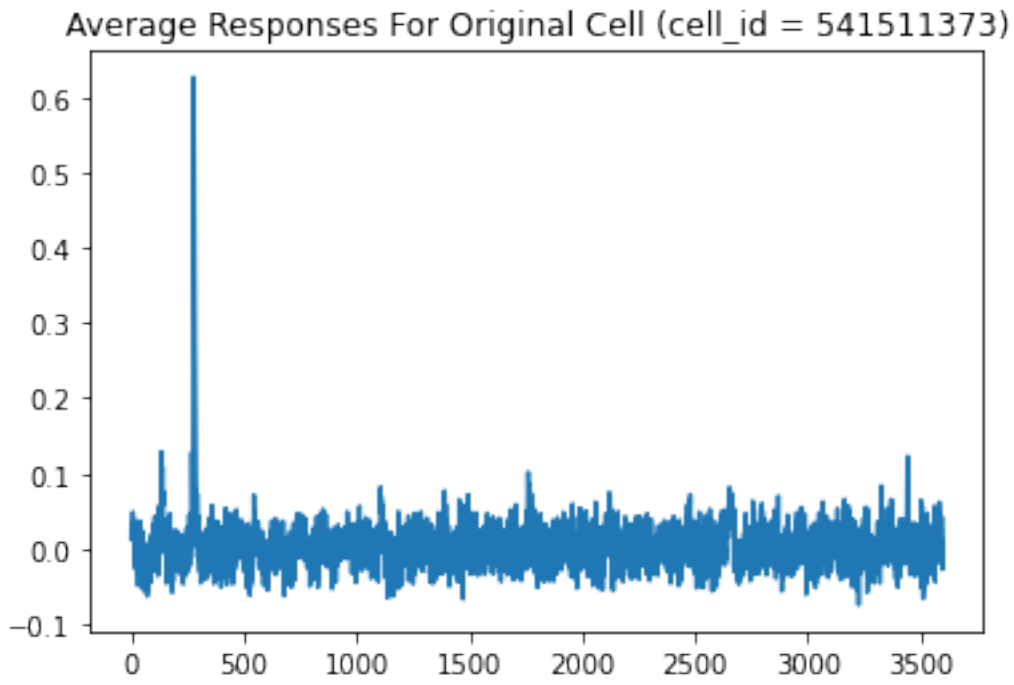
	frame	start	end	repeat
0	0	19746	19747	0
1	1	19747	19748	0
2	2	19748	19749	0
3	3	19749	19750	0
4	4	19750	19751	0

[87]:
```
start_times = stim_table_nm[stim_table_nm.frame==0].start.values
```

[88]:
```
len(stim_table_nm.frame.unique())
nm_res = np.empty((10,3600))
for i in range(10):
    nm_res[i,:] = dff_trace[start_times[i]:start_times[i]+3600]
ans3 = nm_res.mean(axis=0)
```

[89]:
```
plt.plot(ans3)
plt.title("Average Responses For Original Cell (cell_id = 541511373)")
```

[89]:
```
Text(0.5, 1.0, 'Average Responses For Original Cell (cell_id = 541511373)')
```



[8]:
```
cell_id = 541511385
timestamps, dff_trace, stim_table_nm = get_dff_traces_and_stim_table(cell_id,␣
 ↪'natural_movie_three')
```

[9]:
```
stim_table_nm.head()
```

[9]:

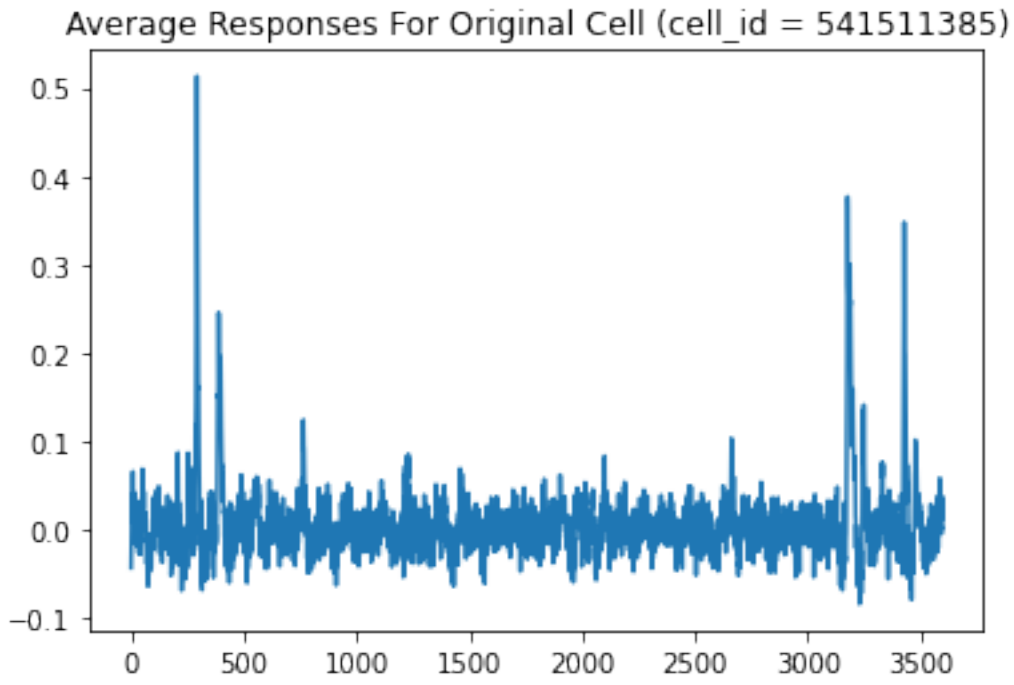
	frame	start	end	repeat
0	0	19746	19747	0
1	1	19747	19748	0
2	2	19748	19749	0
3	3	19749	19750	0
4	4	19750	19751	0

[10]:
```
start_times = stim_table_nm[stim_table_nm.frame==0].start.values
```

[11]:
```
len(stim_table_nm.frame.unique())
nm_res = np.empty((10,3600))
for i in range(10):
    nm_res[i,:] = dff_trace[start_times[i]:start_times[i]+3600]
ans4 = nm_res.mean(axis=0)
```

[12]:
```
plt.plot(ans4)
plt.title("Average Responses For Original Cell (cell_id = 541511385)")
```

[12]:
```
Text(0.5, 1.0, 'Average Responses For Original Cell (cell_id = 541511385)')
```



[70]:
```
r,p = st.pearsonr(ans3, ans4)
print(r)
```

```
0.12840115710291727
```

For an extra challenge, what is the relationship between the correlation of grating responses to the correlation of movie responses across the whole population? To do this, you'll want to get the traces for all of the cells in the experiment. Look at our function at the beginning called get_dff_traces_and_stim_table(). If you use some of the code in there, and change the get_dff_traces line to:

timestamps, dff = data_set.get_dff_traces()

you will get an array of the DFF traces of all of the cells. From there you can calculate tuning for all the cells and calculate the pairwise correlations. What do you expect the result to be?

[125]:
```
def get_dff_traces_and_stim_table(cell_specimen_id, stimulus):
    #identify the session for a given cell id and stimulus
    exps = boc.get_ophys_experiments(cell_specimen_ids=[cell_specimen_id],␣
 ↪stimuli=[stimulus])
    #get the session_id for that session
    session_id = exps[0]['id']
    #access the data for that session
    data_set = boc.get_ophys_experiment_data(session_id)
```

```
    #get the DFF trace for the cell
    timestamps, dff = data_set.get_dff_traces()
    dff_trace = dff[0,:]
    #get the stimulus table for the stimulus
    stim_table = data_set.get_stimulus_table(stimulus)
    #return everything
    return (dff_trace, stim_table)
```

[126]:
```
stim_table.head()
```

[126]:

	temporal_frequency	orientation	blank_sweep	start	end
0	1.0	45.0	0.0	736	796
1	15.0	90.0	0.0	826	886
2	1.0	270.0	0.0	917	977
3	NaN	NaN	1.0	1007	1067
4	4.0	90.0	0.0	1098	1158

[131]:
```
cell_ids = [541512490, 541512611, 541512645, 541512079, 541511403, 541511670,␣
 ↪541511373, 541513771, 541511385, 541512607]
dffs= np.empty((len(cell_ids), 115741))
```

[132]:
```
def compute_tuning_m(dff_trace,stim_table):
    cell_response= np.zeros((len(stim_table),3))
    for i in range(len(stim_table)):
        cell_response[i,0] = stim_table.orientation[i]
        cell_response[i,1] = stim_table.temporal_frequency[i]
        cell_response[i,2] = dff_trace[stim_table.start[i]:stim_table.end[i]].
 ↪mean()
    all_ori = np.unique(cell_response[:,0])
    orivals = all_ori[np.isfinite(all_ori)]
    all_tf = np.unique(cell_response[:,1])
    tfvals= all_tf[np.isfinite(all_tf)]
    tuning_array = np.empty((len(orivals),len(tfvals)))
    for i,tf in enumerate(tfvals):
        tf_trials = np.where(cell_response[:,1]==tf)[0]
        subset = cell_response[tf_trials]
        for j,ori in enumerate(orivals):
            trials = np.where(subset[:,0]==ori)
            tuning_array[j,i] = subset[trials,2].mean()


    return tuning_array.flatten()
```

[133]:
```
dff_trace, stim_table_nm = get_dff_traces_and_stim_table(cell_ids[0],␣
 ↪'natural_movie_three')
tuning = compute_tuning_m(dff_trace, stim_table_nm)
```

```
plt.imshow(tuning, interpolation='none')
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)

<ipython-input-133-718cf56385e8> in <module>
  1 dff_trace, stim_table_nm = get_dff_traces_and_stim_table(cell_ids[0],␣
↪'natural_movie_three')
----> 2 tuning = compute_tuning_m(dff_trace, stim_table_nm)
  3 plt.imshow(tuning, interpolation='none')


<ipython-input-132-dac829d81644> in compute_tuning_m(dff_trace, stim_table)
  2     cell_response= np.zeros((len(stim_table),3))
  3     for i in range(len(stim_table)):
----> 4         cell_response[i,0] = stim_table.orientation[i]
  5         cell_response[i,1] = stim_table.temporal_frequency[i]
  6         cell_response[i,2] = dff_trace[stim_table.start[i]:stim_table.
↪end[i]].mean()


~/SageMaker/custom-miniconda/miniconda/envs/python37_allensdk/lib/python3.
↪7/site-packages/pandas/core/generic.py in __getattr__(self, name)
  5177             if self._info_axis.
↪_can_hold_identifiers_and_holds_name(name):
  5178                 return self[name]
-> 5179             return object.__getattribute__(self, name)
  5180
  5181     def __setattr__(self, name, value):


AttributeError: 'DataFrame' object has no attribute 'orientation'
```

[ ]: