Problem Set 2: Tuning Curves

Problem Set 2 will give you the opportunity to continue to develop your Python coding skills, while incorporating some of the visualization techniques we introduced to you in Unit 2. In Problem Set 2, you will create **Tuning Curves** which will display the firing characteristics of particular neurons.

Why do neuroscientists use tuning curves?

One of the main goals of neuroscientists is to understand the messages communicated by neurons. That is, neuroscientists want to "crack the neural code". For neurons in the sensory system, a tuning curve will convey information about the range of sensory input for which a neuron responds. For example, in the auditory system, we find neurons that are "tuned to" (or respond best to) a particular frequency (pitch) of sound. The neuron will respond to a range of frequencies by firing action potentials at various rates. We can characterize the range of frequencies, and the strength of the neural response, via a tuning curve. The peak of the tuning curve will indicate the pitch the neuron "prefers", while the width of the turning curve will tell us how specific the neuron is, with wide curves indicating broader tuning. We see tuning curves throughout the senses. Another classic example is found in the retina, where color-sensitive photoreceptors, known as cones, are tuned to various wavelengths of light.

In the motor system, we find neurons tuned for particular directions of motion. These neurons often fire prior to the start of the movement and are thought to contribute to the organism's behavioral responses. Similar to what we observe in the sensory systems, these neurons fire action potentials prior to movements in a range of directions, with their peak firing rate corresponding to the direction the neuron "prefers".

As you can see, tuning curves allow you to determine what stimuli lead to neural firing, and what the neural firing might represent to downstream neurons and muscles. If we want to "decode" the message of a neuron, we need to understand what the neural firing might indicate. By constructing tuning curves, we can begin to understand how a particular neuron is contributing to sensory and motor neural circuits.

To complete this problem set, you will

- Write a function that sorts neural responses based on behavioral events;
- Write functions to plot the neural responses related to various behaviors;
- Write functions to create tuning curves.

At the end of this problem set, you will

- Complete a programming assignment submission which will evaluate that your function correctly sorts and characterizes the neural responses;
- Complete a peer assessment submission to confirm that you are accurately plotting tuning curves. (To receive credit for the peer assessment, you must complete your peer evaluation by the evaluation due date.)

- Complete some short questions to assess your ability to interpret the data.

Things to keep in mind while you are coding

- It's really easy to make mistakes when you are writing code. It happens to everyone. Try not to get frustrated. Ask for help if you need it!
- Sometimes it is useful to print out the values of some of your variables to make sure things are going as you want them to (or you can use the debugger in Spyder – but printing is sometimes much easier!)
- It is often very useful to write out your logic with pencil and paper before you begin.

Experimental design

Monkey Task

For this experiment, the monkey completed a center-out task with a minipulandum (like a joystick). That is, starting with the manipulandum in the center, the monkey moved it from the center to one of eight locations, all equidistant from the center (located at 0°, 45°, 90°…315°). The monkey received a visual cue at the start of the trial which indicated the direction of motion to be performed by the subject. In this problem set, we will consider the time the monkey began moving on each trial.

Electrophysiological Recordings

The data collected for this data set were obtained from a monkey implanted with an array of 96 electrodes in the arm area of M1 (primary motor cortex). Monkeys were trained on the task prior to the recording session.

Getting started

Your first step is to open problem_set2.py in Spyder (or whatever Python environment you are choosing to use.) Just as with the last problem set, you will want to open an IPython console in Spyder. Looking at the .py file, you'll see that it starts with some comments, and some import commands. You'll also see some functions (and skeletons of functions) that we will work through as we go through the problem set. Below all the functions, you'll find some example lines of code.

Loading the experiment data

As in Problem Set 1, we've provided you with a function to read in some data. First, we will load the parameters of the experiment using a function called `load_experiment`. You can run this function as follows:

```
trials = load_experiment('trials.npy')
```

This reads in the example data set and stores the data in a 2-dimensional array. The first column gives you the direction of motion for the trial, while the second column gives you the time that the animal began his movement during that trial. (This is similar to what we saw in Problem Set 1 where you were provided with two arrays in correspondence. The difference here is that all of the data is contained in one, 2-dimensional, array.) That is, `trials[n,1]` indicates the time of the n^{th} trial and `trials[n,0]` indicates the direction of motion for that trial. (Directions of motion are indicated by an angle in degrees.)

You can explore the array returned from `load_experiment()`. You can answer questions like how many total trials did the animal perform? How many trials per direction? Were the directions performed in a particular order?

All our data comes from neurons recorded simultaneously during this experiment, so these trial parameters will not change as we look at different neurons.

Loading the real neural data

We have also provided you with the function to read the neural data.

```
spk_times = load_neuraldata('example_spikes.npy')
```

will read in the times of the action potentials recorded from our example neuron. You will use different file names here to vary the neuron you are analyzing. For this problem set, we are providing you with the spike times, as if you had ran your spike detector code from Problem Set 1 on the data set.

What do you notice about the order of the spike times in `spikes`?

Bin the data and determine the firing rates per direction

The ultimate goal of this problem set is to generate tuning curves. This means we need to convert our array of all the times our neuron fired into the average firing rate for each direction of motion.

We will grade your `bin_spikes()` function, so it must be in the format provided in the problem_set2.py

```
dir_rates = bin_spikes(trials, spk_times, time_bin)
```

where trials and spk_times are the arrays we loaded above, and time_bin is a parameter we can vary to determine how much time before and after movement began we'd like to look. (For example, `time_bin = 0.1` will count the spikes from 100ms before to 100ms after the trial began.)

There are multiple approaches you can take to solve this problem. Here we will walk through one option, but you can try your own approach if you wish.

- Count the number of spikes per trial.
  The data in `trials` is nicely arranged by trial for you. You could make a new array, perhaps called `spikes_per_trial`, that remains in correspondence with `trials`. We will count all the spikes that occur from bin_size before to bin_size after the animal started movement. (Recall that M1 neurons fire prior to the onset of movement.). Bin size is definitely a parameter that you may want to play with as you explore the data set, a size of 100ms (so `bin_size = 0.1`) is a reasonable place to start, and what we used to create the figures shown here.
  This is a tricky step! Think about how you might figure this out. Check the hints below for some help.
- Group the trials by direction of motion.
- Average over the trials for each direction.
- Convert from spike counts to firing rate (spikes/s). This is a more standard way to present the data. This allows for interpretation independent of bin size.

Your function must return an 8x2 array with the first column containing the directions (in degrees) and the second column containing the average firing rate for that direction.

Hints:

`np.count_nonzero` and `np.logical_and` are helpful functions that operate on arrays and may be useful to you here.

`np.zeros` and `np.arange` can help you generate arrays that may come in handy. For example, you can initialize an array using `np.zeros,` or generate an array with all the directions of motion using `np.arange(0,360,45)`. Alternatively, if you wanted your code to be more broadly applicable, you could use `np.unique(trials[:,0])` to determine the directions of motion used in the experiment.

`np.column_stack` lets you combine two 1-D arrays into columns of a 2-D array.

Remember to use help or the object inspector if you don't know how a function works!

Plot the data

Use `plt.bar` to create a histogram of your data. Be sure your axes are properly labeled.

`plt.xlim` and `plt.xticks` can help you create a prettier plot.

Use `plt.polar` to create a polar plot of your data. Recall that `plt.polar` assumes radians. `np.deg2rad` can help. (Review the Python video about polar plots if you need help getting this plot correct.)

If you'd like to label your polar plot, you can try:

```
plt.polar(theta, r, label='Your label here')
plt.legend(loc=8)
```

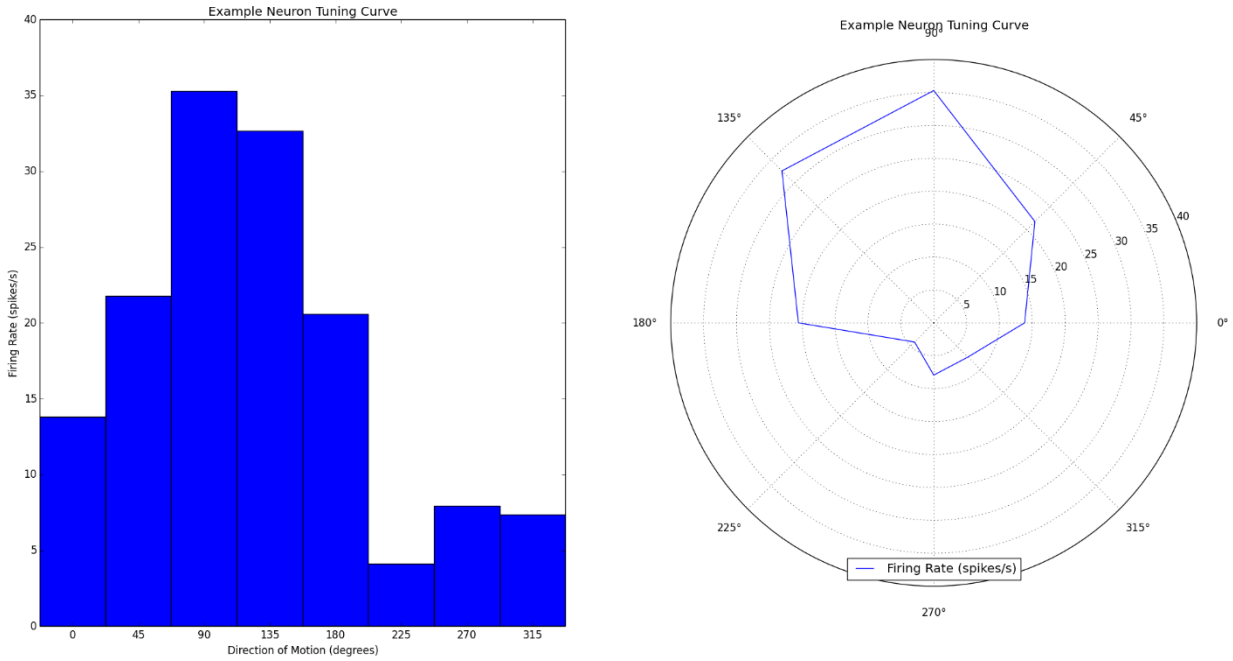
where `theta` and `r` are replaced by the appropriate variables. The location argument allows you to move the legend to different parts of the plot. See help for `plt.legend` for more information.

For your peer assessment assignment, we would like you to use subplot to make one figure that presents this data in both ways (and also presents the curve fit results – see below). You can use subplot as follows:

```
plt.subplot(2,2,1)
plt.bar(….your code here….)
#additional code here
plt.subplot(2,2,2,polar=True)
plt.polar(…,your code here…)
#additional code here
```

The first two arguments to `plt.subplot` are the numbers of rows and columns (respectively) of sub-axes you'd like in your plot. The third argument is the location within this grid of sub-axes and runs from 1 to number of rows * number of columns. The last argument allows your subplot to be a polar plot. You may want to make a function that draws these subplots.

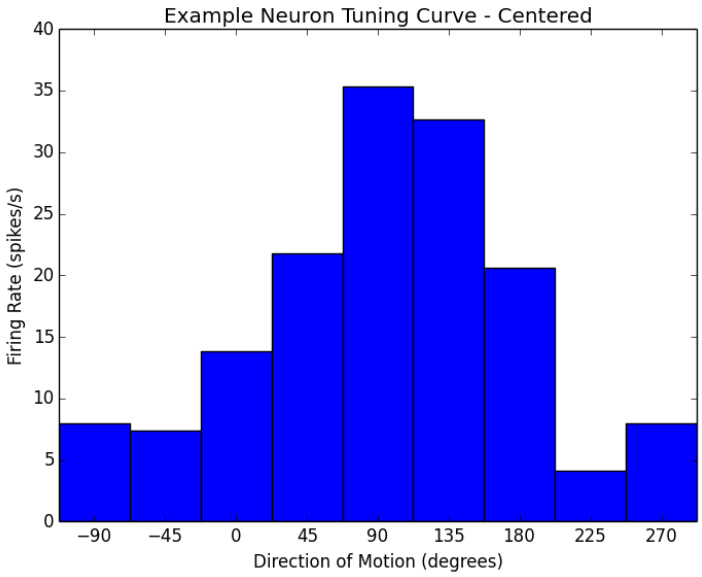
Your subplots should look something like this:



Fit a curve to the data

Now let's try to fit a curve to our data. Our curve roughly looks like a normal distribution, so we'll try to fit a normal curve. But there's one big caveat we need to think about with this – the distribution is split up! In our example, it's only a minor split, but think about what your histogram might look like if the preferred direction of the neuron was 0°.The tricky thing here is that 360° = 0°, and the linear way we have graphed this doesn't account for that. If we keep things, the way they are, we won't be able to fit a normal curve to our data.

This means we need to move our curve so the peak is near the middle. Were we to graph it (you don't have to – but you can to check your work) it should look something like this:



Essentially, all I've done here is shift the x-axis to center the curve. We created a new array of x- values for my histogram, and shifted my y-values to center the maximum value. The function `np.roll` is an easy way to "roll" or shift the values of your array in a circle to accomplish this. You will also notice that we duplicated first value at the end (like we do to make our polar plot work correctly). This is helpful because this now gives us an odd number of bars (9), which means we can put our largest bar in the center, and it helps us to fit a normal distribution (which is symmetric by definition) to the data.

We started the function `roll_axes` for you, where you can change your x- and y-values as above. While it's fairly straightforward to use `np.roll` and `np.append` for the y-values, the x-values are trickier because you need to actually change the values. It can be helpful to use some paper and pencil to think about how you might get the new x values.

Now we can fit a normal distribution to this data. We will follow the method introduced to you in the curve fitting video. We defined for you a new function, `normal_fit` that we will use with the `optimize.curve_fit` to optimize the parameters for our normal distribution.

```
def normal_fit(x,mu, sigma, A):
    n = A*mlab.normpdf(x,mu,sigma)
return n
```

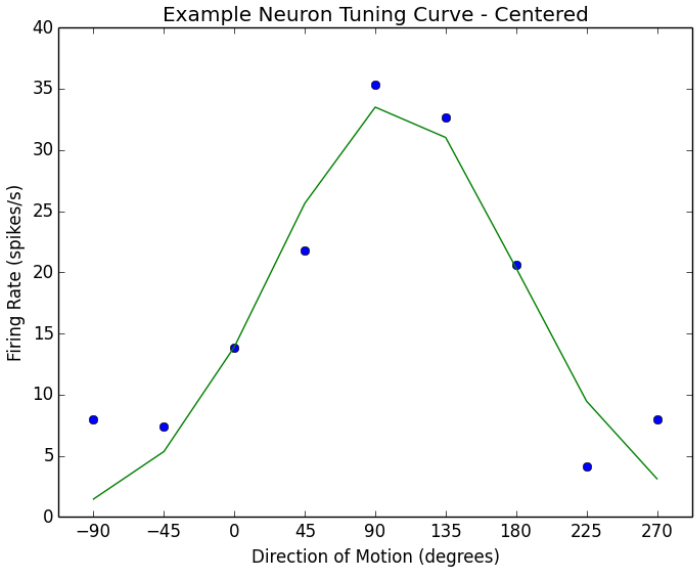
The curve fitting function is trying to optimize the parameters of our normal curve (its mean, variance and amplitude) calculated over the x-values we give it, so that the n returned by `normal_fit` is very close to our y-values. To help the `optimize.curve_fit` function accomplish this, we will give it some starting guesses as follows:

```
max_y = np.amax(y)          #What is the biggest y-value? (This
                            #estimates the amplitude of the curve,
                            #A)
max_x = x[np.argmax(y)]     #Where is the biggest y-value? (This
                            #estimates the mean of the curve, mu)
sigma = 90                  #Here we are approximating one standard
                            #deviation of our normal distribution
                            #(which should be around the width of 2
                            #bars).
```

```
p, cov = optimize.curve_fit(normal_fit,x, y, p0=[max_x, sigma,
max_y])
```

where the variable names in italics should be changed to match your variable names.

Now we can use `p` to fit our points with a curve. We can generate the fit as follows:
`fit_ys = normal_fit(new_xs,p[0],p[1],p[2])`
if you plot this, it should look like this:



We need to make two adjustments to this plot. First, one of the main reasons to fit a tuning curve to our data is to estimate the firing rate at directions we did not test. Therefore, we should fit our curve at more x values. For example,

```
curve_xs = np.arange(new_xs[0],new_xs[-1])
```
which can then be used to generate the `fit_ys`.

Additionally, we need to get back to our original x-axis (from 0° to 360°). So we need to roll our y-values.

You may find it helpful to make a function that does the rolling, fitting, and rolling back.

Preferred Direction

There are several ways to characterize a tuning curve, but for our purposes, we will determine the neuron's preferred direction. This is the direction where we expect the highest neural response.

Please complete the preferred direction function given in the code. The function should take a 2-dimensional array with the x-values of your fit curve in the first column and the y-values of your fit curve in the second. It should return the preferred direction of the neuron (in degrees).

Depending on the fit you have used, the preferred direction of the example neuron is around 100°-105°.

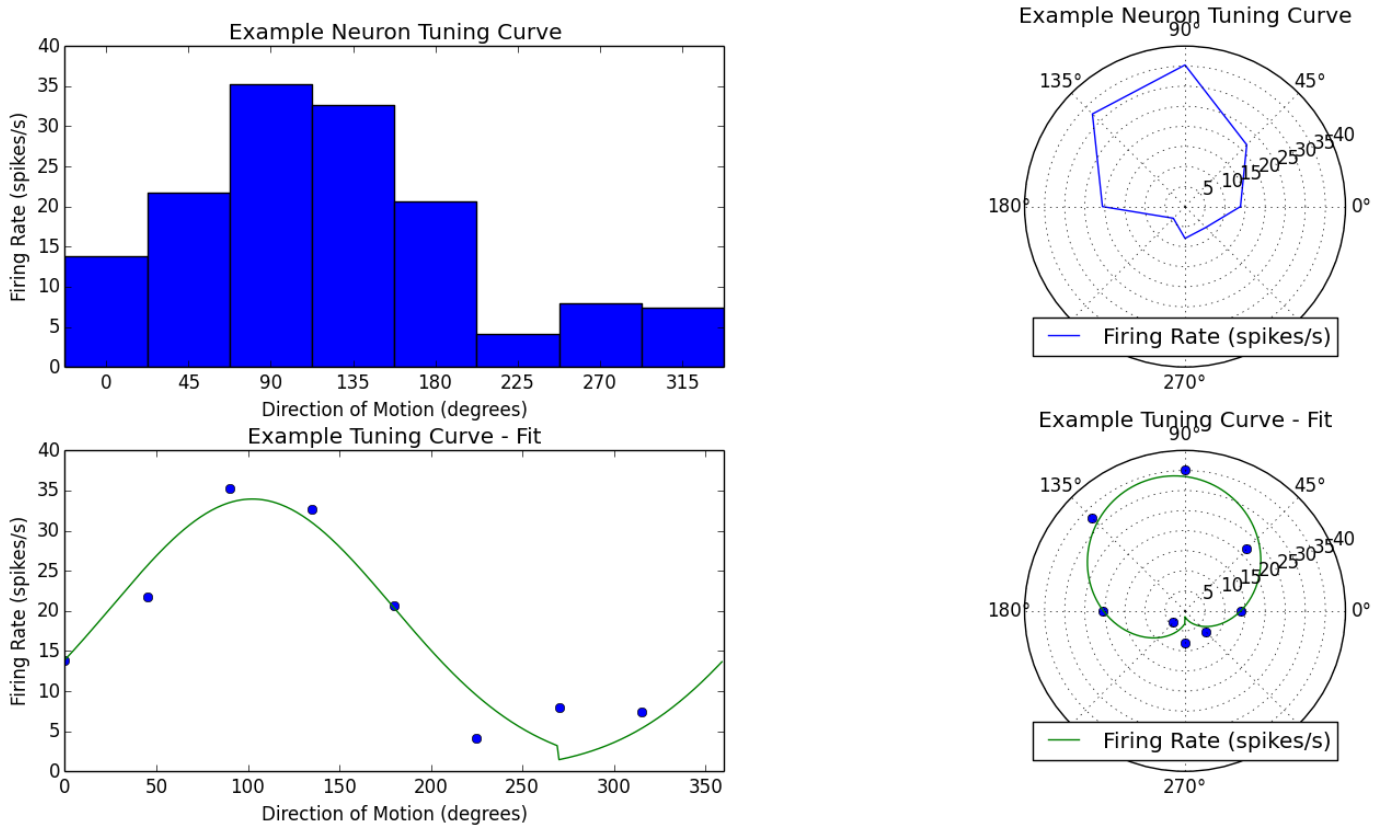
As an optional exercise, you can consider measures of the broadness of the tuning curve. One common measure is the full width at half maximum, which is the difference between the two x-

values when the y-value is equal to half of the peak value. This is a bit tricky to find, and you need to consider the cases where the curve is split.

Create the figure!

To create the overall figure to represent the tuning curve for this neuron, you should add two subplots to your original figure. In position 3, you should have the curve fit on the linear axes and in position 4, you should show the fit in polar coordinates.

Your figure should look something like this:



Additional notes about curve fits:
There are lots of ways to fit curves. You are welcome to try other fits! You may notice that the fit curve has a little discontinuity. That is from the rolling and unrolling. We can fix that by using a different distribution to fit.

Optional: The Von Mises distribution
The Von Mises distribution is also known as the circular normal distribution, which takes care of our linear plotting issues. We aren't going to go into the mathematics behind the distribution

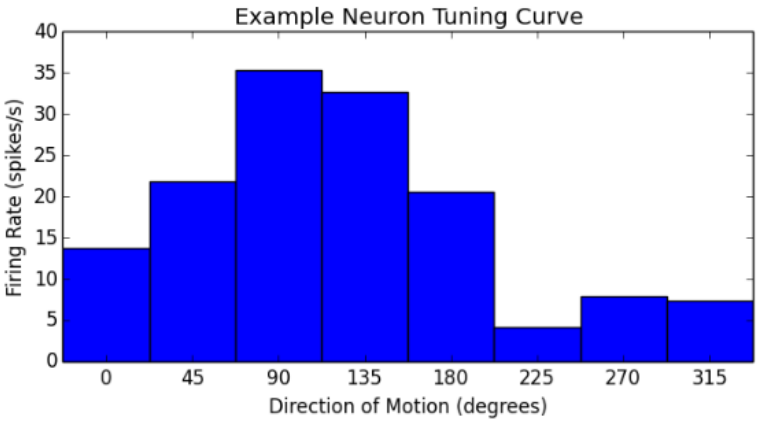
here, but if you'd like to try to use it, we've provided `von_mises_fitfunc` to you. The function works over radians, not degrees, so be sure to convert your values! Decent parameter guesses are:

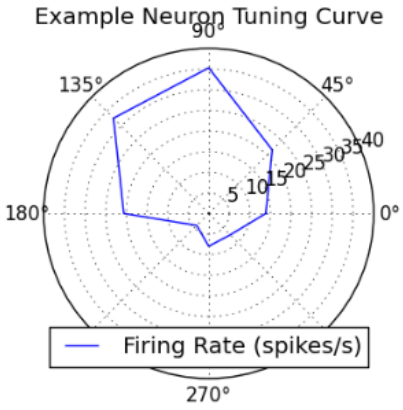
```
max_y = np.amax(y)          #What is the biggest y-value? (This
                            #estimates the amplitude of the curve,
                            #A)

max_x = x[np.argmax(y)]     #Where is the biggest y-value? (This
                            #estimates the mean of the curve, mu)
```

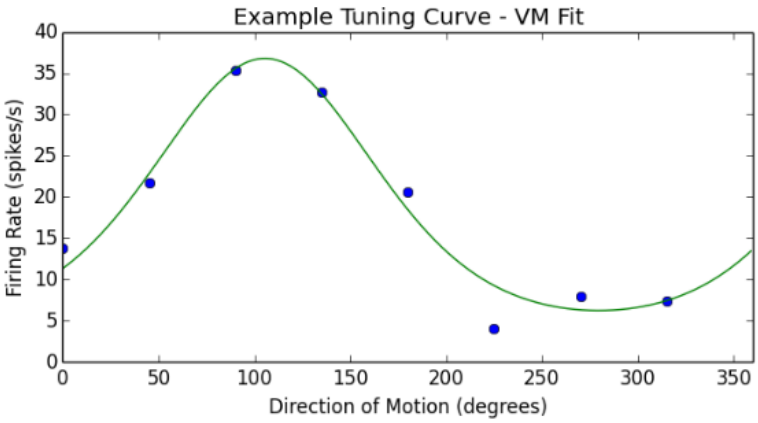
and you can call it as follows:

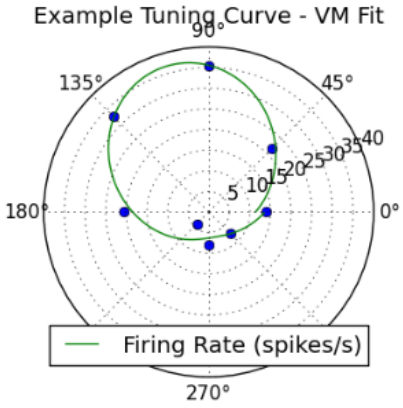
```
p, cov = optimize.curve_fit(von_mises_fitfunc,x, y, p0=[max_y,
4, max_x,1])
```

If you do this, your figure should look something like this:









Assignment problems:

To complete the assignments, we'd like you to create the final figure for the three neural data files, `neuron1.npy, neuron2.npy` and `neuron3.npy.` You will submit the results of your tuning curve code using the `submit_problem_set2.py` code (see **Submitting your code** below) and upload the results of your plotting routines using the Peer Assignment tab in Coursera (see **Submitting your figures for Peer Assessment** below). You will also complete some very short questions **Problem Set 2: Homework Questions** under the Quizzes tab in Coursera.

Submitting your code:
Once you have updated problem_set2.py to include your `bin_spikes()` function, you can submit this part of your problem set by opening the file `submit_problem_set2.py` in Spyder. With that file open, you can run it (press the green play button or hit F5) and it will prompt you for your email address and a "one time password". You get this one time password from Coursera. On the Programming Assignments page, at the top, you should see your own *Submission Password*. This is NOT your Coursera password.

If you have logged in correctly, the submit script will run your `bin_spikes()` on the neuron1, neuron2 and neuron3 data sets.

Submitting your figures for Peer Assessment:

To complete the problem set, you will also need to complete the **Problem Set 2 Peer Assessment**. Here you will be asked to upload three final figure files (from `neuron1, neuron2` and `neuron3` data sets.). Please read the question prompts carefully to ensure that you submit the correct figure in the correct locations! You will need to save the figures (for inline figures, right click on the figure and select "Save As", for figures in their own window, just choose the save icon). You can save the files as png, jpg, gif or pdf for peer assessment.

The purpose of the peer assessment is to ensure that you are presenting your data in a meaningful way that can be easily interpreted by others. Therefore, a lot of the assessment is based on the format of your figure, including axes labels and the figure title. Each figure will be assessed in the following categories (1 or 2pts possible per category): Please note, there are multiple ways to receive full credit – only examples are given:

The axes are properly labeled:

- Histogram x-axes should be labeled and include units (it is sufficient to only label the bottom plot since it applies to the whole column). The x-axes should be labeled with "Direction", "Direction of Motion" or something similar, with units of degrees.
- Histogram y-axes should be labeled and include units (both plots must be labeled). The y-axes should be labeled with "Firing Rate", "Spike Rate" or something similar with units of spikes/s, Hz, or something equivalent.

0: axes are unlabeled or all incorrectly labeled

1: Some axes have missing or incorrect labels or some units are incorrect or not provided.

2: All axes are correctly labeled (as described above) with descriptive labels and accurate units.

The sub-plots are properly titled. There are lots of options for the figure titles, they just need to make sense. For example "Tuning Curve", "Neuron 1 Tuning Curve", "Tuning Curve Histogram"/"Tuning Curve Fit" are all acceptable titles.
If the title provided applies to both subplots in the same column, only the top figure needs to be titled.

0: All subplots are missing titles, or all provided titles are irrelevant.

1: Some subplots (or figure columns) are correctly titled, while others are missing or not provided

2: All subplots (or figure columns) are descriptively titled.

Histogram data are clearly displayed. Columns are suitably wide, and the appropriate number of columns (either 8 or 9 if the first value is repeated at the end) are present. The x and y-axes appropriately span the values of the data.

0: Histograms are not clear, have an inappropriate number of columns, or do not properly span the range of the data

1: The histograms are clear and easy to read

(Please consider both polar plots - so the right-hand column of subplots - when assigning your score.)
Polar plots are clearly displayed. The legend does not obscure too much of the tuning curve.

0: One or both of the polar plots is not clear or is highly obscured by the legend.

1: Both polar plots are clearly displayed.

(Please consider both fit curves - so the bottom row of subplots - when assigning your score.) The fit curves are clearly displayed. The actual data is presented clearly with a marker, while the curve can easily be seen. (Accuracy of the fit is not required to receive full points for this criterion.)

0: One or both of the fit curves is not clear, or does not contain markers.

1: Both fit curves are clearly displayed.

(Please consider both fit curves - so the bottom row of subplots - when assigning your score.) The fits appear reasonable. (We do not expect perfect fits, but the curve should be a reasonable approximation of the data.)

0: The fit curves do not resemble the data.

1: One of the fit curves does not appropriately fit the data.

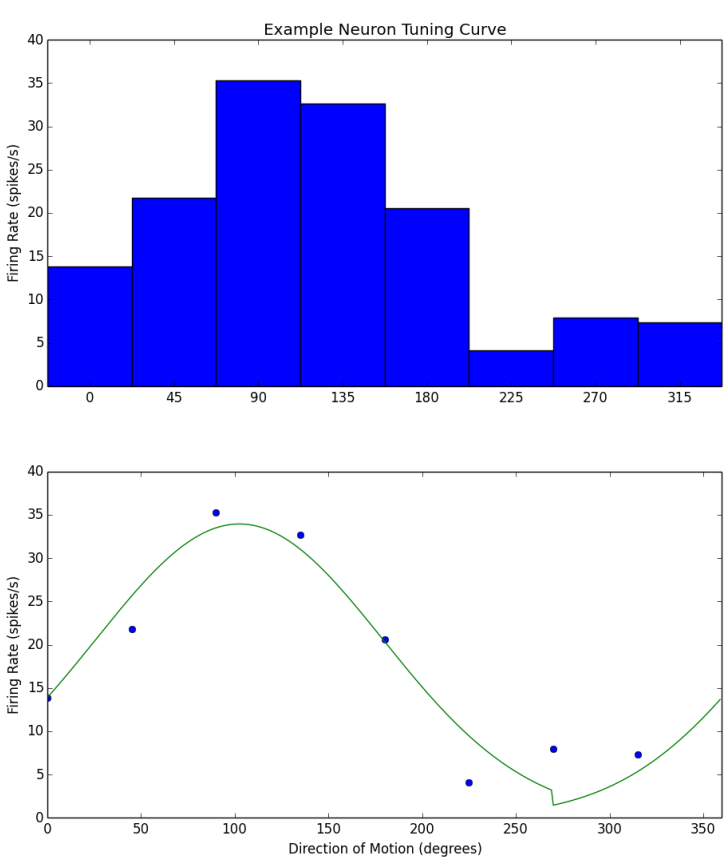
2: Both fit curves appear to be reasonable approximations of the data.

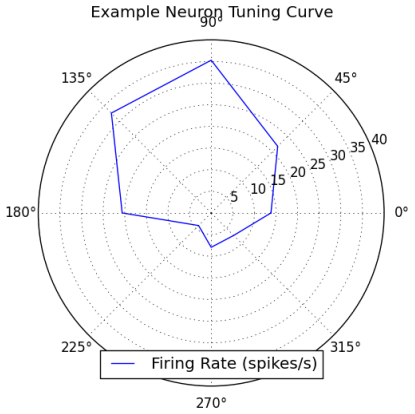
Overall, the figure is clear. The arrangement of the subplots is appropriate and clear. The subplots are not overlapping. There are no additional distractors in the figure.

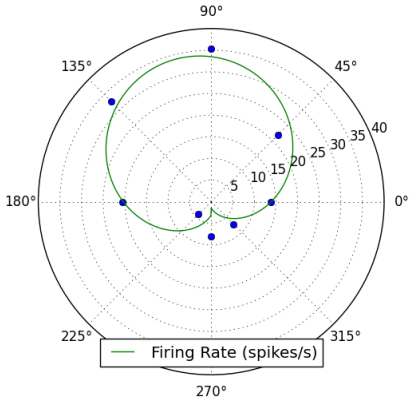
0: The figure as a whole is hard to read because of overlapping subplots or some additional distractor.

1: Overall, the figure is clear.

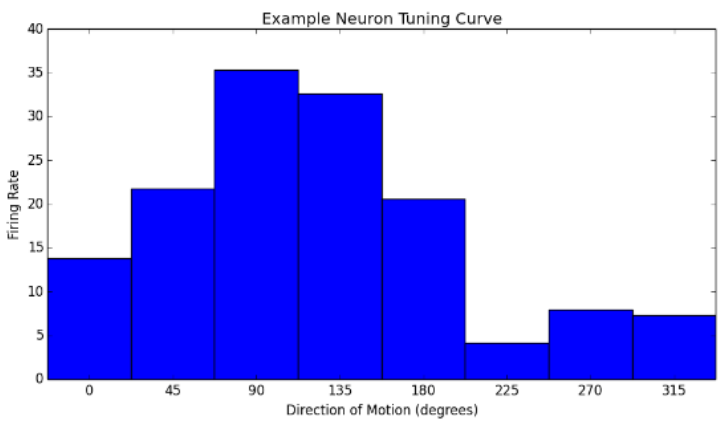
Here are some examples (these show data from the example file, so your data will look

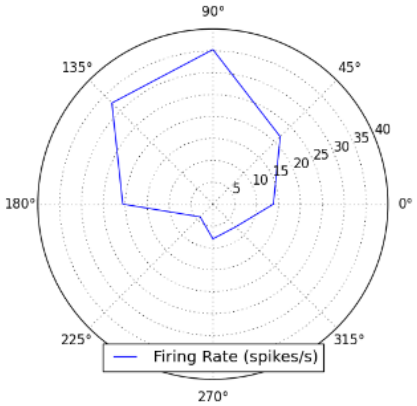


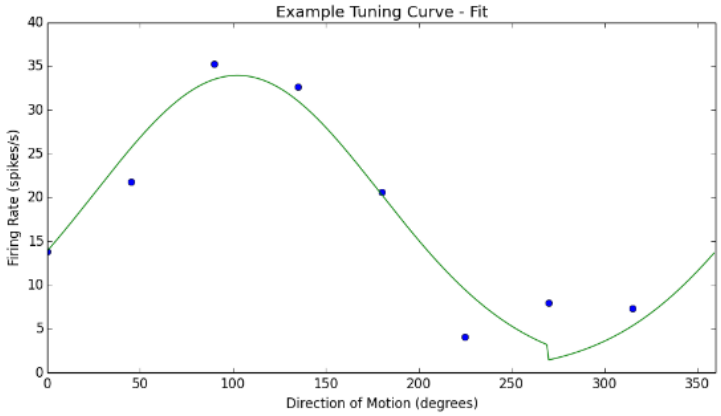


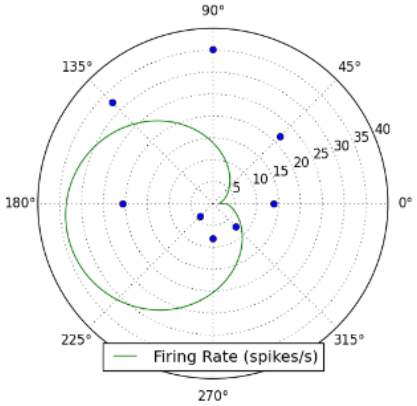


different!):
This figure would receive full credit. The axes are properly labeled and contain units (the x-axis is labeled once for the entire column). The titles are descriptive (again, only printed once per column). The histogram, polar plots and fits are clear. The fits looks reasonable, and the overall figure is clear.



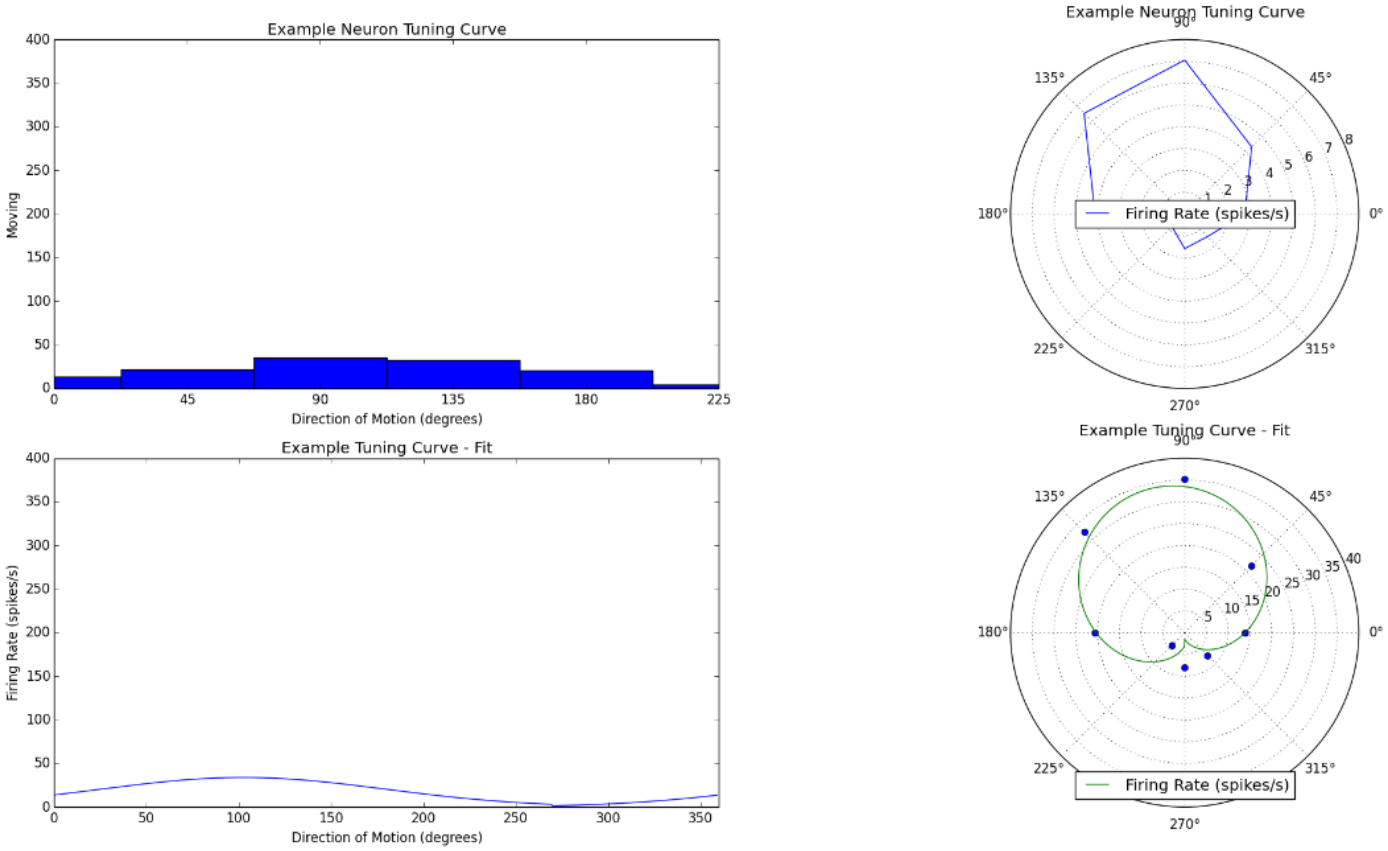






This figure would receive 7/10 points:

Axes:	1 pt.	The y-axis of the histogram does not contain units.
Title:	1 pt.	The polar plots are not titled.
Histograms:	1 pt.	The histogram is clear.
Polar:	1 pt.	The polar plots are clear.
Fit Clarity:	1 pt.	The fits are clear.
Fit Accuracy:	1 pt.	The polar fit does not resemble the data.
Overall:	1 pt.	Clear figure.
		Total Score: 7 pts.



This figure would receive 6/10 points:

Axes:	1 pt.	The y-axis of the histogram is not descriptive and has no units.
Title:	2 pts.	The titles are clear.
Histograms:	0 pt.	The histogram is squished and doesn't span the range of directions.
Polar:	0 pt.	The top polar plot is obscured by the legend.
Fit Clarity:	0 pt.	The tuning curve fit is not clearly displayed and does not contain markers.
Fit Accuracy:	2 pts.	The fits resemble the data.
Overall:	1 pt.	Clear figure.
		Total Score: 6 pts.

Submitting your Homework Questions:

You will have one opportunity to correctly answer the homework questions by following the link on the Coursera page for Problem Set 2 Homework Questions. For that reason, we have provided you with the questions here so you can finalize your answers before entering the questions page on Coursera. It is probably best to complete and submit the programming assignment before answering the questions because some of the answers depend on your code working properly.

Questions 1 – 4 are worth 1 point each.

1. How many total trials did the animal perform in the experiment for Problem Set 2?

2. How many trials did the animal perform in the 45 degree direction?

3. True or False: The animal performed all the trials in order. That is, first he completed all 0 degree trials, followed by all 45 degree trials, etc.

4. Which of the three neurons that you looked at (Neuron 1, Neuron 2, or Neuron 3) was most broadly tuned?

Questions 5 – 8 are worth 2 points each.

5. Assume that you are recording spikes from a neuron in M1. You have already done experiments to create the turning curve for this neuron. Imagine that you do not know the direction the monkey is moving, but you can calculate the firing rate from this neuron. Is that firing rate enough information for you to guess the direction of movement (within a few degrees)?

6. What is the preferred direction of Neuron 1 (rounded to the nearest degree, and given in the range of 0-360 degrees)?

7. What is the preferred direction of Neuron 2 (rounded to the nearest degree, and given in the range of 0-360 degrees)?

8. What is the preferred direction of Neuron 2 (rounded to the nearest degree, and given in the range of 0-360 degrees)?