Spike Train Analysis Tutorial

In this tutorial, we will analyze real data recorded from an orientation-selective neuron in the primary visual cortex. We will plot the cell's tuning curves as a function of stimulus orientation, see the dynamics of its response over time, and analyze how reliable its response is to multiple repetitions of the same stimulus.

I. Background: Orientation-sensitive cells in the primary visual cortex

Cells in the primary visual cortex (also known as "V1" for short) respond primarily to the appearance or movement of oriented bars located in a particular region of space. Any particular neuron will respond better to some orientations and less well to other orientations. A plot of the *tuning curve* of such a neuron gives the average firing rate of the neuron as a function of the orientation of an oriented stimulus. This average usually reflects both an averaging across time (i.e. counting the number of spikes in a given time window) and an average across multiple presentations (or "trials") of the identical stimulus. Although the tuning curve characterizes the *average* behavior of a neuron's response to a given stimulus, we often are interested in knowing more about the exact nature of the response firing rate of the neuron. For example, we might want to know:

1) How does the firing rate of the neuron vary over time?
2) How regular ('clock-like') or irregular is the spacing between individual spikes during a trial?
3) How different are the trials from one another?
4) How correlated are the spike trains across time, i.e. if we observe the firing rate at one point during the trial, can we predict anything about the firing rate at other times?

In today's lab, we will address the first three of these questions by analyzing spike trains generated by an orientation-tuned cell recorded from a cat's primary visual cortex in Dr. Mriganka Sur's laboratory at MIT. The cells were stimulated by moving gratings (i.e. alternating stripes of black and white) oriented at various angles.

II. Loading in the data

To start, we need to load in the data that was provided to us by the Sur laboratory at MIT. Typically, when one receives such data from an experimental lab, the person who took the data has organized it into some logical structure. In our case, the data was organized into a 3-dimensional array in MATLAB-ready format, as described next. Let's load in this data and check out the formatting of the data:

First, download the files "Sur_Orientation_SpikeData.mat" and "Sur_Orientation_Annotation.m" and save them to your desktop (note: if you do not have the MATLAB curve-fitting toolbox, also download the file smooth.m, which is used in this tutorial). These files contain the data and notes about it, respectively.

Then change MATLAB's current directory to the Desktop and open a .m file. Save your file to the same location as the .mat file and name it something appropriate like "Sur_SpikeTrainAnalysis.m".

Let's comment our file as follows and add our usual "clear all" and "close all" statements as the first lines of the file:

```
%Analyze data from recordings of V1 cells responding to oriented
%gratings presented at various angles

clear all
close all
```

Next, we will load in the data and take a look at its format. To load in data from a .mat formatted file all you have to do is type

```
%LOAD IN DATA AND GATHER INFORMATION
load Sur_Orientation_SpikeData
```

Now run your file and you should see that an array of size 18 x 3500 x 30 named "spikes" appears in your Workspace panel.

You will notice that the spikes array is of Class uint8, meaning that it contains integers. This formatting is to save memory, but it will actually mess us up later on for technical reasons (you can't multiply uint8 variables by normal MATLAB numbers!). To convert this to the usual MATLAB number format (called 'double' for double-precision decimal number) add the line:

```
spikes = double(spikes);
```

Ok. Back to the array size: You could also get this size by typing size(spikes) at the Command Window prompt. Try this. The meaning of such an array size is that it is organized into three dimensions that you could picture as points arranged in a 3-dimensional grid. Thus, for example, typing "spikes(2, 2000, 28)" (try this at the command prompt) will give us the value of the spikes array at the grid location with coordinates (2, 2000, 28), which in this case is zero.

To find out what these 3 dimensions of sizes 18, 3500, and 30 correspond to, open up the file "Sur_Orientation_Annotation.m". This file has notes from the provider of the data which tell us that:

- The first dimension of the data corresponds to the 18 conditions tested: in the first 16 sets of experiments, the angle of the grating was oriented and moved at a 0 degree angle, then a 22.5^0 angle, then a 45^0 angle, and so on in steps of 22.5^0 all the way around a circle (up to 337.5^0). In the final two experiments, a blank screen of the same average brightness (luminance) of the gratings was presented. These final two experiments were control experiments to test the responsiveness of the neuron to the same average light level in the absence of a grating.

- The second dimension corresponds to the spike train values over time, where a value of 1 indicates that there was a spike at this time, and a value of 0 indicates no spike. The experiment ran for 3500 ms, with data gathered every 1 ms. At t=0, the oriented grating was turned on; at t=500 ms the grating started moving (with orientation and direction specified by the first dimension of this data set; gratings were always oriented perpendicular to the direction of movement), and at t=2500 ms the stimulus was turned off.

- The third dimension of the data corresponds to the 30 trials that were run for each condition. That is, for each direction of movement, they took 30 separate spike train recordings.

Thus, we now know what "spikes(2,2000,8) = 0" means: it says that, during the 8th trial of recording from a bar oriented at 22.5^0 (the 2nd condition tested), there was no spike ("0") at the 2000th time point (=2000 ms after the stimulus was turned on, and 1500 ms after the stimulus began drifting). Whew!

Next let's put some of the data into MATLAB variables that might prove useful:

```
%SET UP USEFUL VARIABLES
NumControls = 2;  %number of control experiments with no grating
dt = 1;  %spacing between sampled time points [ms]
t_On = 0;  %time stimulus turns on [ms]
t_Move = 500; %time stimulus begins moving [ms]
t_Off = 2500; %time stimulus turns off [ms]
NumAngles = size(spikes,1) - NumControls  %number of angles tested, equally spaced;
                                          %last 2 sets of recordings are controls
NumTimePoints = size(spikes,2)  %number of time points; time was sampled every 1 ms
NumTrials = size(spikes,3)      %number of trials performed at each angle
```

The first five lines are information that we only know from reading the annotation file information. The last three lines (NumAngles, NumTimePoints, NumTrials) we could have likewise just assigned "by hand" (i.e. just written in the numbers) but by programming it this way we can more easily generalize to future data sets we might receive from this laboratory that might, for example, have more trials per orientation angle or more angles recorded from. Note the syntax of the "size" command: "**size(*array, n*)**" gives the length of the nth dimension of an array. You can run your code now.

Let's add a final line that will be very useful in plots:

```
t_vect = t_On:dt:(NumTimePoints-1)*dt; %time vector for each trial
```

This gives the time vector that we'll want to use in our plots. It runs from time dt to the final time of the recording (=NumTimePoints*dt). Note that, to make this vector the same length as the spike train data (i.e. 3500 points) we needed to make the time plot go from t=0 ms to t=3499 ms (the value of (NumTimePoints-1)*dt).

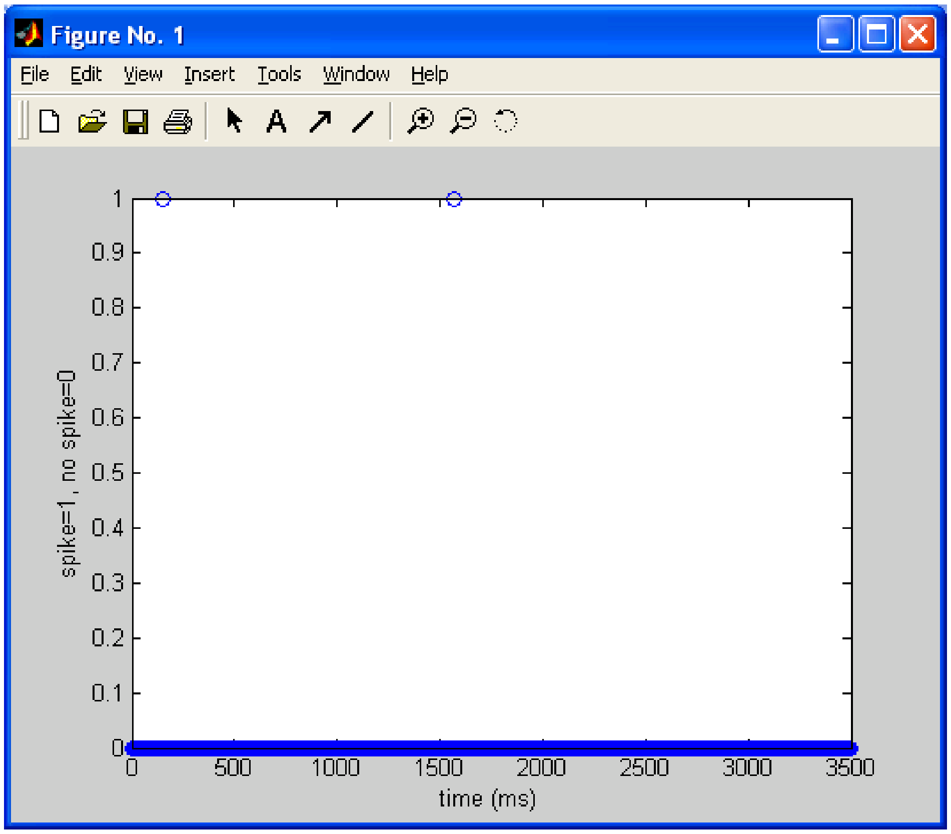
III. Making a raster plot

A) Plot a single trial

Next let's make a plot of a spike train for one of the trials. To do this, we want to plot the vector of 0's (no spike occurred) and 1's (spike occurred) over time for a given trial and a given orientation. Let's plot the 20th trial of the 2nd (22.5^0 orientation) condition. We'll plot time along the x-axis and points representing the 0 or 1 (no spike or spike) value on the y-axis. We can do this with:

```
%PLOT RASTERS FOR ONE PARTICULAR ANGLE
figure(1)
plot(t_vect,spikes(2,:,20),'o')
xlabel('time (ms)')
ylabel('spike=1, no spike=0')
```

We've introduced a new MATLAB syntax here: the **colon** in spikes(2,:,20) is MATLAB's shorthand way of writing spikes(2,1:end,20) or equivalently spikes(2,1:3500,20) since there are 3500 time points. If you run this, you should see that there were 2 spikes on this trial:



Since we know the stimulus has stopped by 2500 ms, we might only want to plot the first 2500 ms of data. To do this, we would need to revise our plot command to read:

```
plot(t_vect,spikes(2,1:2500,20),'o')
```

Try this. Did you get an error?

```
??? Error using ==> plot
Vectors must be the same lengths.
```

Can you figure out why? (try using the length command to figure this out). The problem is that we told MATLAB to plot the full, 3500-element t_vect vs. only 2500 elements of the spikes vector. To plot only the first 2500 elements of t_vect, you can revise your plot line to read:

```
plot(t_vect(1:2500),spikes(2,1:2500,20),'o')
```

Run this. Your plot's x-axis should now only extend to 2500 ms.

Another "fancy" command that is sometimes nice for visualizing spike trains is to replace the "plot" command by the command "**stem**" which for each x point draws a line from the x-axis to your data value. Try it by replacing the plot command by:

```
stem(t_vect,spikes(2,:,20),'.')
```

The last argument to this function specifies the shape of the end point of the line (and the color if you choose to add this. The format of the stem command is just like that of the plot command). Try plotting a few more spike trains, corresponding to different trials and/or orientations.

B) Make a raster plot of all trials for a given orientation

Next let's make a 'raster plot' that shows time on the x-axis and the spike trains of *all* trials on the y-axis. How can we do this? Let's think: we want to do the same basic routine over and over again for each trial… sounds like a for loop, with the loop going from trial = 1 to trial = NumTrials. We also want each trial to be located at a separate y-axis location, so let's plot the first trial at y=1, the second at y=2, the 3rd at y=3, etc.

Let's try it. Replace your previous plotting section by:

```
%PLOT RASTERS FOR ONE PARTICULAR ANGLE
figure(1)
for trial=1:NumTrials
   plot(t_vect,trial*spikes(2,:,trial),'+')
   hold on
end
xlabel('time (ms)')
ylabel('trial number')
hold off
```

We use hold on to make sure the information is not erased from trial to trial (and turn hold back off at the end of this code). If we had simply given the plot command plot(t_vect,spikes(2,:,trial),'+') then every single plot would have appeared at the vertical (i.e. y) location y=1. By multiplying the spikes array by trial, we make each trial plot spikes at y-values equal to the given trial.

The plot you have made, showing spike times for multiple trials with the same stimulus, is called a "raster" plot. Trial 20 should look familiar from part A of this tutorial. Our raster plot displays the spike train data for all trials conducted with this bar orientation.

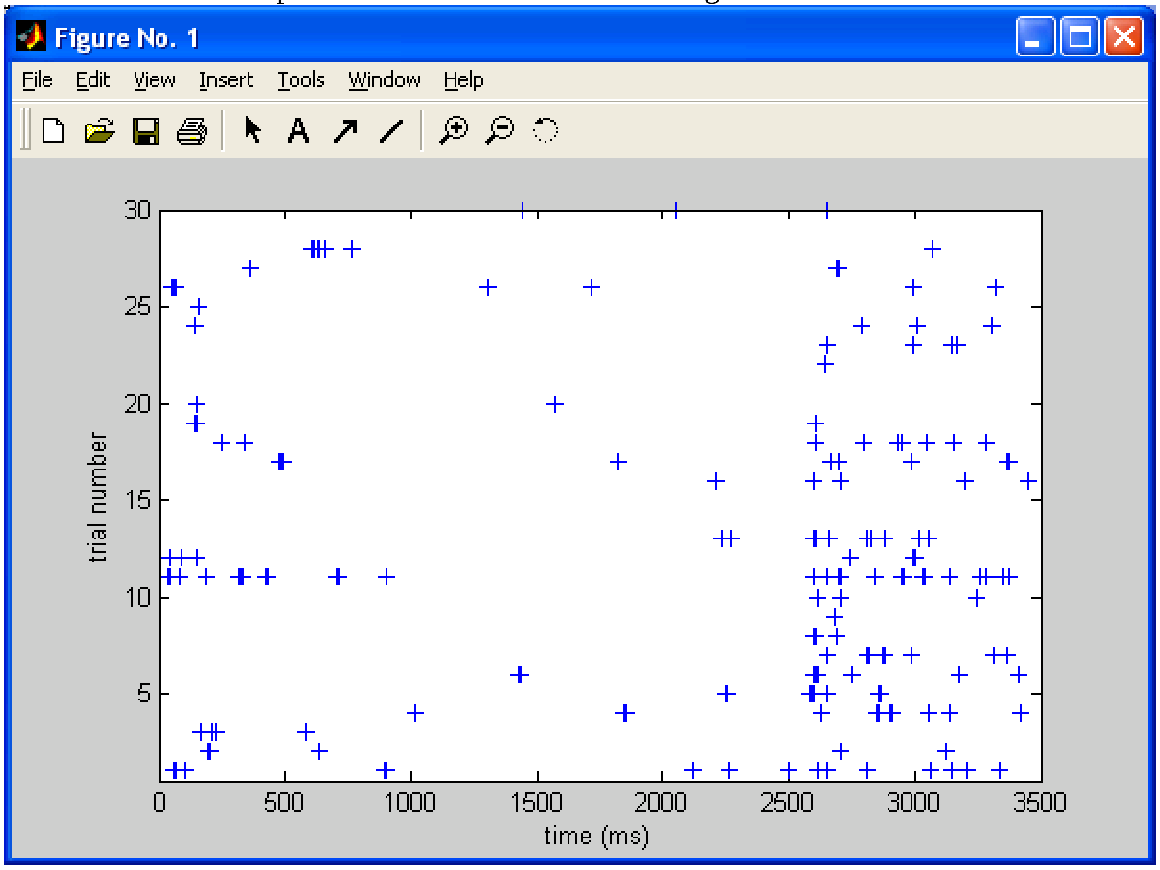
Sometimes when you plot this long a time period, the spikes will almost overlap on your screen. To see whether this is occurring, you can zoom in on portions of the data by clicking on the

zoom-in tool (magnifying glass with the "+" inside it **(**magnifying glass with the "+" inside it) in the figure window and then dragging a rectangle around the region you want to zoom in on. To return to the original plot format, double-click on the graph.

Finally, let's get rid of all of the zeros in the bottom row, as they are superfluous. We can do this by adjusting our y-axis. Do this by adding an "axis" command between the ylabel and hold off lines (type help axis if you can't remember the format of this command):

```
axis([0 3500 0.5 NumTrials])
```

Run this. Your final plot should look like the following:



I recommend saving your work at this point if you have not done so recently.

Your code at this point should read:

```
%Analyze data from recordings of V1 cells responding to oriented
%gratings presented at various angles

clear all
close all

%LOAD IN DATA AND GATHER INFORMATION
load Sur_Orientation_SpikeData
spikes = double(spikes);
```

```
%SET UP USEFUL VARIABLES
NumControls = 2;  %number of control experiments with no grating
dt = 1;  %spacing between sampled time points [ms]
t_On = 0;  %time stimulus turns on [ms]
t_Move = 500; %time stimulus begins moving [ms]
t_Off = 2500; %time stimulus turns off [ms]
NumAngles = size(spikes,1) - NumControls  %number of angles tested, equally spaced;
                                          %last 2 sets of recordings are controls
NumTimePoints = size(spikes,2)  %number of time points; time was sampled every 1 ms
NumTrials = size(spikes,3)      %number of trials performed at each angle
t_vect = t_On:dt:(NumTimePoints-1)*dt; %time vector for each trial

%PLOT RASTERS FOR ONE PARTICULAR ANGLE
figure(1)
for trial=1:NumTrials
   plot(t_vect,trial*spikes(2,:,trial),'+')
   hold on
end
xlabel('time (ms)')
ylabel('trial number')
axis([0 3500 0.5 NumTrials])
hold off
```

IV. Calculating the post/peri-stimulus time histogram (PSTH) for this trial

As you may have noticed, the trial-to-trial variability in the response of this neuron is huge (e.g. compare trial 20 to trial 1, especially when the stimulus is off. This activity in the absence of a stimulus is called ‘background activity’). This is common for neocortical neuron spike trains.

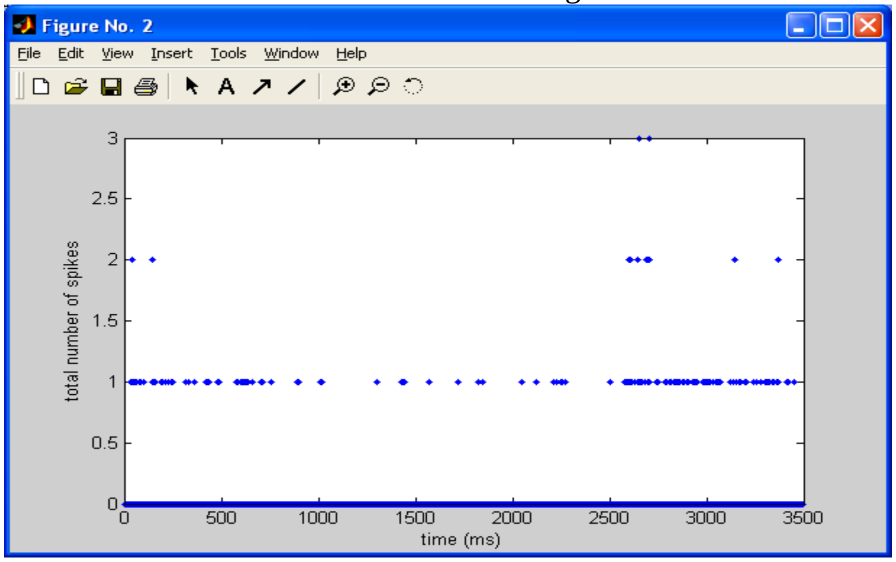
To get a better understanding of this neuron’s typical response, it is helpful to average the data across trials and plot the trial-averaged firing rate of a neuron in response to a stimulus. When this trial-averaged firing rate is plotted as a histogram in time bins near (“peri”) or following (“post”) the presentation of the stimulus, it is called a *PSTH* (which is short for “*peri-*” or “*post-*” *stimulus time histogram*). Below we will learn how to create a PSTH for the set of trials of our data set corresponding to presentation of a particular angle, and thereby characterize the average firing rate response of the neuron to presentation of a stimulus at this angle.

Since we will want to create an average firing rate across trials, let’s as a first step just sum the data across trials by typing:

```
%PLOT AVERAGED (PSTH) DATA FOR ONE PARTICULAR ANGLE
TrialSum_vect = sum(spikes(2,:,:),3);  %adds up spike trains across trials
figure(2)
plot(t_vect,TrialSum_vect,'.')
xlabel('time (ms)')
ylabel('total number of spikes')
```

The array spikes(2,:,:) is a 2-dimensional matrix containing all of the raster data for the 2nd orientation (22.5^0), i.e. all time points (the first colon) for all trials (the second colon). The sum command **sum(*array, dimension*)** sums the data in the array across the dimension specified by dimension. In our case, it sums over the 3rd dimension, i.e. over all trials.

Run this code. You should see the following:



The plot gives the total number of spikes that occurred at a given time point over all of the trials. Since it is highly unlikely that more than one spike appeared in a given millisecond-sized time bin, most of the values are zero or 1.

A more useful quantity to plot along the y-axis is the *average spike rate* over trials

$$\langle r \rangle_{trials} = \frac{(\#\text{ of spikes in time window for all trials})}{(\#\text{ of trials})*(\text{time window duration})} \qquad (1)$$

where the "time window" duration over which we will compute this average is a single time bin (for now...). Let's do this, and also multiply by 1000 to convert to Hz. Replace the final 3 lines by:

```
plot(t_vect,1000*TrialSum_vect/(NumTrials*dt),'.')
xlabel('time (ms)')
ylabel('Average firing rate (Hz)')
```

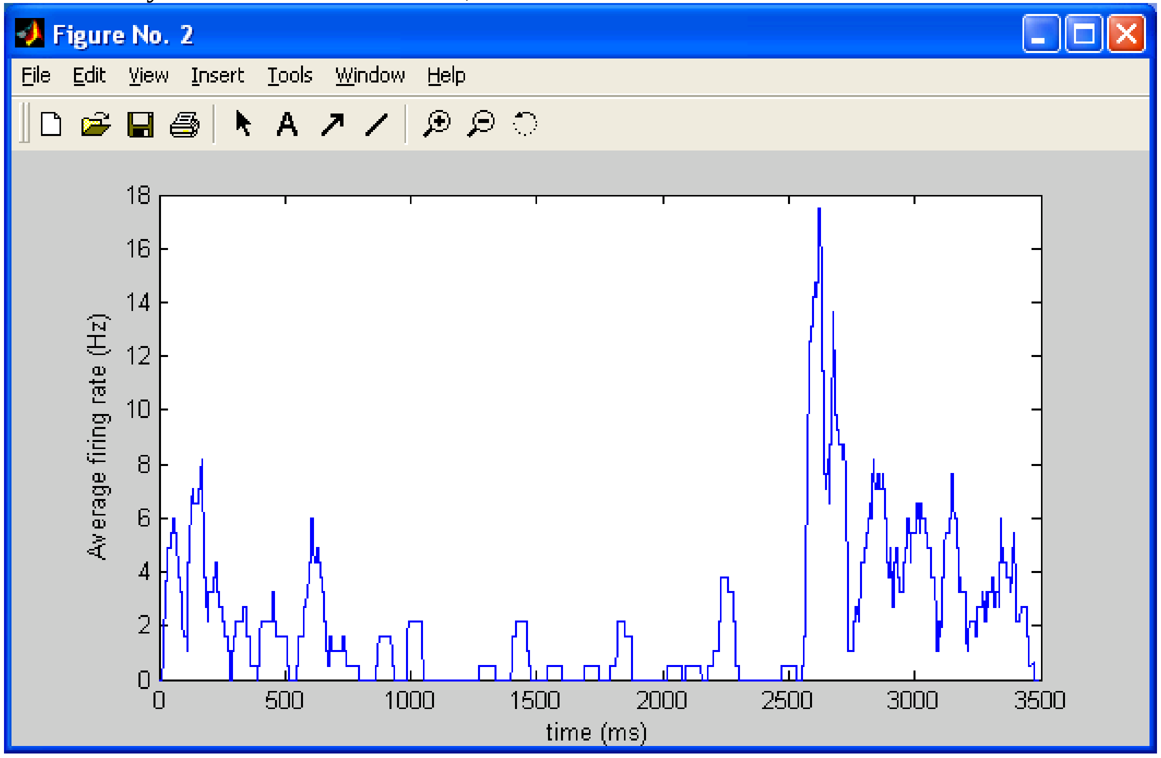
Run this and you should now see that the rate varies from 0 Hz to 100 Hz. These numbers make sense, 1 spike on a single trial in a time dt=1 ms would correspond to a rate of 1000 Hz; 3 spikes on a single trial to 3000 Hz; and 3 spikes in 30 trials to 100 Hz.

Although our plot is technically correct, it's not very useful because it's very discontinuous. What we really need to do is to smooth this data by instead calculating the averages over a wider time bin such as 60 ms. To do this, we can produce a smoothed set of data by using the command **smooth(*vector, # of points to smooth over*)**, which at each time point produces the average of the data centered at the given time point and averaged over the nearest *(# of points)*, including the point itself (for more information, see smooth in the help menu). [*Two technical notes: 1) the # of points must be an odd number so that the average includes the point itself and equal number of points on each side. If you instead put an even number as this argument,* *MATLAB will use the number you gave minus one. 2) For the beginning and end of the vector, MATLAB only averages over numbers reaching to the end of the vector (e.g. the first point of the smoothed vector doesn't have a point to its left, so there is no smoothing done; the second point only has a single point to its left, so the smoothing is done over just 3 points: the point itself and the neighbors to the right and left. Likewise for assigning the end of the smoothed vector.)*]

To do this, let's modify our code so that it smooths over +/- 30 bins (= +/- 30 ms for our data), as follows:

```
%PLOT AVERAGED (PSTH) DATA FOR ONE PARTICULAR ANGLE
TrialSum_vect = sum(spikes(2,:,:),3);  %adds up spike trains across trials
SmoothingWidth = 61; %smooths data over +/- [(this # - 1)/2] bins
TrialSum_smooth_vect = smooth(TrialSum_vect,SmoothingWidth);
figure(2)
plot(t_vect,1000*TrialSum_smooth_vect/(NumTrials*dt))
xlabel('time (ms)')
ylabel('Average firing rate (Hz)')
```

Make sure you understand this code, then run it. You should see:



We notice that the stimulus appears to suppress the firing of this neuron (compare to when the stimulus is off at the end of the trial), especially during the period when the stimulus is moving (starting at 500 ms, although note that there is some delay or 'latency' between the stimulus turning on and the signals reaching these neurons). The firing rate then jumps up when the stimulus turns back off (at 2500 ms), before settling back down to its background firing rate. This is typical behavior: sensory cells typically respond most (either with excitation or inhibition) when stimuli change. Try this code again for a different orientation and for different smoothing widths (e.g. orientation condition 13 is a stimulus that highly excites the cell and you will again notice that the cell responds most when either the stimulus first turns on, or when it first starts moving). You should notice a latency (delay) in the response, which gives you an idea how long it takes for information in the visual world to reach the visual cortex.

V. Calculating the Fano factor

The PSTH gives a measure of the trial-averaged firing rate of a neuron. To get a measure of the variability of the neuron's response we might try something like counting the number of spikes in a given time window and seeing how this varies across trials.

Let's do this for orientation condition 13 (angle of 270^0) for the period of time during which the stimulus is moving (t=500 to t=2500). Because there seems to be a latency in the response of the neuron, let's actually start our average at t=600 ms. In addition, so that we can make sure to keep all parts of our program consistent, let's add a variable called "ThisOrientation," set it equal to 13, and change all references to the orientation condition to "ThisOrientation". The last parts of your code should now read (changes are noted in **boldface**):

```
t_vect = t_On:dt:(NumTimePoints-1)*dt; %time vector for each trial
ThisOrientation = 13;  %element index of orientation we are currently analyzing

%PLOT RASTERS FOR ONE PARTICULAR ANGLE
figure(1)
for trial=1:NumTrials
   plot(t_vect,trial*spikes(ThisOrientation,:,trial),'+')
   hold on
end
xlabel('time (ms)')
ylabel('trial number')
axis([0 3500 0.5 NumTrials])
hold off

%PLOT AVERAGED (PSTH) DATA FOR ONE PARTICULAR ANGLE
TrialSum_vect = sum(spikes(ThisOrientation,:,:),3);  %adds up spike trains across trials
SmoothingWidth = 61; %smooths data over +/- [(this # - 1)/2] bins
TrialSum_smooth_vect = smooth(TrialSum_vect,SmoothingWidth);
figure(2)
plot(t_vect,1000*TrialSum_smooth_vect/(NumTrials*dt))
xlabel('time (ms)')
ylabel('Average firing rate (Hz)')

%COMPUTE FANO FACTOR
TStartCount = 600; %time to start computing average
TEndCount = 2500; %time to end computing average
%next line gives number of counts for each trial
NumCounts_vect = sum(spikes(ThisOrientation,((TStartCount+1)/dt):(TEndCount/dt),:),2)
```

Note that spikes(13,(TStartCount+1)/dt):(TEndCount/dt),:), gives the time points ranging from TStartCount to TEndCount for all (indicated by ":") trials of condition 13 (the divided by dt converts times into element indexes; and the +1 is necessary because t=0 is the first element of the spikes array). To get the total number of spikes occurring over this time interval in each trial, we sum over the 2nd dimension of the spikes array.

Run this and verify that the resulting vector is correct by comparing to Figure 1.
The *Fano factor* gives the ratio of the variance (the square of the standard deviation) in the number of counts N across trials σ_N^2 to the mean number of counts μ_N :

$$\text{Fano factor} = \frac{\sigma_N^2}{\mu_N}$$

To compute this, add the line:

```
FanoFactor = (std(NumCounts_vect)^2)/mean(NumCounts_vect)
```

where **std** gives the standard deviation of a vector and **mean** gives the average of the elements.

You should get a Fano factor of 14.4, which is quite high (the typical model of such cells is as a Poisson process, which has a Fano factor of 1 (which is still quite variable)). I think some of the extra variability here may relate to the fact that the earlier trials appear to have systematically higher firing rates than the second half of the trials. If we were performing further experiments, we probably would want to look into this issue more carefully.

VI. Calculating the inter-spike interval (ISI) histogram and corresponding coefficient of variance CV_{isi}

The Fano factor is a nice way to capture the variability across trials. The *coefficient of variance* (or "*CV*") of the interspike intervals (defined below) gives a complementary measure that reflects the regularity of an individual trial's spike train.

The interspike intervals for a given spike train are the durations of the intervals t_{isi} between spikes (for example, in the Integrate-and-Fire Model tutorial, these are calculated analytically for the integrate-and-fire neuron receiving constant input). These intervals are easily obtained from spike data by the following strategy:

1) Calculate the spike times
2) Take the differences between spike times

Let's do this next. To calculate the spike times, we'll use the find command to locate the 1's in a given spike train. Let's do this for the first trial of the orientation 13 data, for the section of data when the stimulus is on:

```
%COMPUTE CV_isi
SpikeTimes_vect = dt*find(abs(spikes(ThisOrientation,TStartCount:TEndCount,1)-1) < 0.00000001)
```

The find command has as its argument a vector of conditions that can be true or false – in this case, whether or not each element of the spikes vector is within 0.00000001 of 1 – and returns the indices of the elements of the vector for which the condition is true. To get the hang of this command, I recommend that you try making up some simple vectors and test conditions and play with the find command in your Command Window. For the task at hand, the reason we use the test condition that the absolute value ('abs') of the difference from 1 is less than a very small number (instead of simply looking for spikes values equal to 1) is that, when numbers are assigned as doubles, MATLAB will sometimes consider 1.00000 to be different from 1 (I don't know why). So, in short, this code is safer. The dt converts from index elements (which is what the find command returns) to time. If you run this code, you should find that there are 34 spikes during this portion of trial 1.

Now to compute the ISI distribution, we just need to take the difference between neighboring spike times. This is done with the diff command (type this):

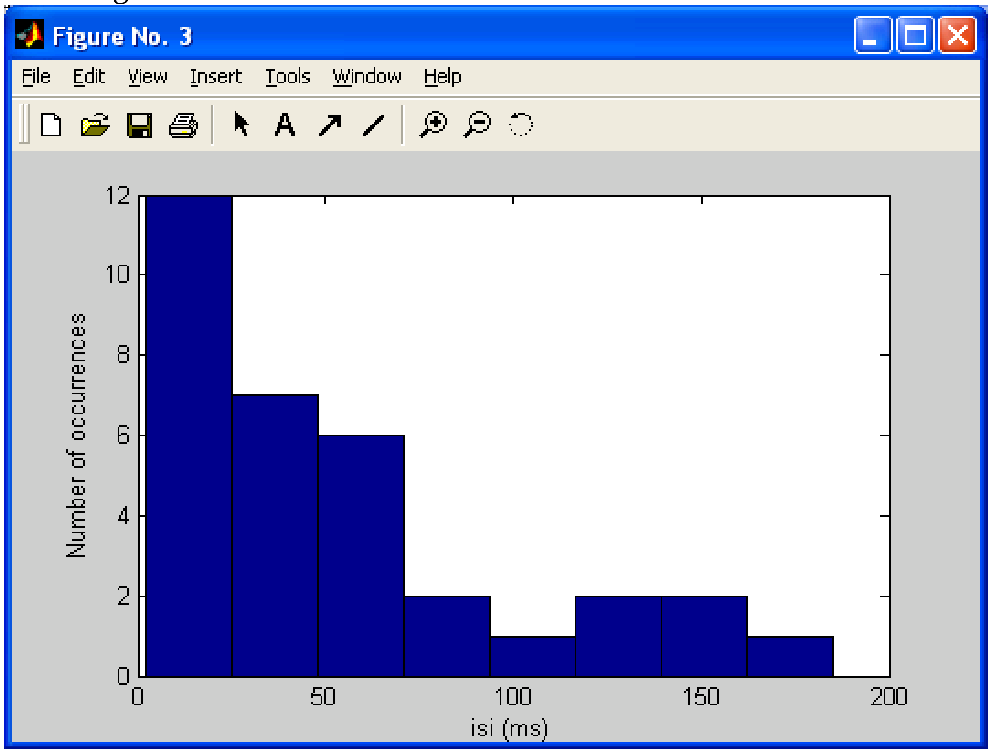
```
isi_vect = diff(SpikeTimes_vect)
```

Compare your isi_vect and SpikeTimes_vect to verify that this worked correctly.

Finally, let's make a histogram of these values using the "**hist**" command. Add:

```
%plot isi histogram
figure(3)
hist(isi_vect,8)
xlabel('isi (ms)')
ylabel('Number of occurrences')
```

This will produce a histogram of your ISI data, binned into 8 bins. It should look like the following:



The y-axis gives the number of times an interspike interval with values in the ranges specified on the x-axis occurred. We see that the most commonly occurring ISI's are small intervals. This is quite typical of cortical neurons.

Finally, the coefficient of variation of this distribution or 'CV_{isi}' is defined as the ratio of the standard deviation σ_{isi} of this distribution to its mean μ_{isi}:

$$CV_{isi} = \sigma_{\mathrm{isi}} / \mu_{\mathrm{isi}}$$

i.e. CV_{isi} represents how many standard deviations wide the distribution we just plotted is in units of its mean.

Let's compute this. Append to your code:

```
%compute CV_isi
mean_isi = mean(isi_vect)
std_isi = std(isi_vect)
CV_isi = mean_isi/std_isi
```

You should get that μ_{isi} = 52.39 ms, σ_{isi} = 50.46 ms, and $\mathrm{CV}_{\mathrm{isi}}$ = 1.04. If you wanted to get the average firing rate over this time period, you could take the inverse of μ_{isi} (times 1000 to convert to Hz) which gives an average rate of approximately 19 Hz. CV's close to 1 in value are very typical of neocortical neuron data.

Note that if the spike train does not contain 2 spikes, then you will get a divide by zero error. You could write an if statement to catch this possibility before it caused the error.

To get a more accurate measure of the $\mathrm{CV}_{\mathrm{isi}}$ of this neuron, one could calculate $\mathrm{CV}_{\mathrm{isi}}$ for all 30 trials and average these values to get a trial-averaged $\mathrm{CV}_{\mathrm{isi}}$. To save time, we will omit this calculation.

Your code at this point should read (I have added a few optional semicolons to suppress output of vectors):

```
%Analyze data from recordings of V1 cells responding to oriented
%gratings presented at various angles

clear all
close all

%LOAD IN DATA
load Sur_Orientation_SpikeData
spikes = double(spikes);

%SET UP USEFUL VARIABLES
NumControls = 2;  %number of control experiments with no grating
dt = 1;  %spacing between sampled time points [ms]
t_On = 0;  %time stimulus turns on [ms]
t_Move = 500; %time stimulus begins moving [ms]
t_Off = 2500; %time stimulus turns off [ms]
NumAngles = size(spikes,1) - NumControls  %number of angles tested, equally spaced;
                            %last 2 sets of recordings are controls
NumTimePoints = size(spikes,2)  %number of time points; time was sampled every 1 ms
```

```
NumTrials = size(spikes,3)      %number of trials performed at each angle
t_vect = t_On:dt:(NumTimePoints-1)*dt; %time vector for each trial
ThisOrientation = 13;  %element index of orientation we are currently analyzing

%PLOT RASTERS FOR ONE PARTICULAR ANGLE
figure(1)
for trial=1:NumTrials
   plot(t_vect,trial*spikes(ThisOrientation,:,trial),'+')
   hold on
end
xlabel('time (ms)')
ylabel('trial number')
axis([0 3500 0.5 NumTrials])
hold off

%PLOT AVERAGED DATA (PSTH) FOR ONE PARTICULAR ANGLE
TrialSum_vect = sum(spikes(ThisOrientation,:,:),3);  %adds up spike trains across trials
SmoothingWidth = 61; %smooths data over +/- [(this # - 1)/2] bins
TrialSum_smooth_vect = smooth(TrialSum_vect,SmoothingWidth);
figure(2)
plot(t_vect,1000*TrialSum_smooth_vect/(NumTrials*dt))
xlabel('time (ms)')
ylabel('Average firing rate (Hz)')

%COMPUTE FANO FACTOR
TStartCount = 600; %time to start computing average
TEndCount = 2500; %time to end computing average
%next line gives number of counts for each trial
NumCounts_vect = sum(spikes(ThisOrientation,((TStartCount+1)/dt):(TEndCount/dt),:),2);
FanoFactor = (std(NumCounts_vect)^2)/mean(NumCounts_vect)

%COMPUTE CV_isi
SpikeTimes_vect = dt*find(abs(spikes(ThisOrientation,TStartCount:TEndCount,1)-1) < 0.00000001);
isi_vect = diff(SpikeTimes_vect);
%plot ISI histogram
figure(3)
hist(isi_vect,8)
xlabel('isi (ms)')
ylabel('Number of occurrences')
%compute CV_isi
mean_isi = mean(isi_vect)
std_isi = std(isi_vect)
CV_isi = mean_isi/std_isi
```

VII. Plotting the tuning curve of this neuron

Finally, let's plot the tuning curve of this neuron. Recall that the tuning curve gives the average firing rate of the neuron across trials and over the duration of a stimulus, i.e. the average is over *both* time within the trial and across trials. Let's plot this tuning curve for the period of time when the stimulus is moving, which we recall is from 600 ms to 2500 ms.

To calculate this average we can again use the formula for the trial-averaged rate $\langle r \rangle_{trial}$ of equation (1):

$$\langle r \rangle_{trials} = \frac{\text{(\# of spikes in time window for all trials)}}{\text{(\# of trials)*(time window duration)}}$$

This time, however, the duration of the time window over which we are calculating the average rate is 1900 ms (= (2500 – 600) ms).

In calculating the Fano factor, we already calculated the number of counts in each trial, which we assigned to the NumCounts variable. Now let's sum this vector across trials to get the number of counts in all trials. Then we'll divide this by the # of trials and by the time window duration to get the average firing rate (and multiply by 1000 to convert to Hz):

```
%COMPUTE AVE RATE FOR 1 TRIAL
TotalCounts = sum(NumCounts_vect) %total spikes across time and trials
AveRate = 1000*TotalCounts/((TEndCount-TStartCount)*NumTrials) %average across time and trials
```

You should get an answer of 12.5 Hz for this orientation.

Now we need to get the average rate for all 16 orientations. We could do this with a for loop, looping over all orientations. However, let's see if we can be more efficient than that by using vectors. Previously, in our PSTH code, we counted the number of spikes *across trials* for various time points and a particular orientation. Then, in our Fano factor code, we counted the number of spikes *over time* for various trials and a particular orientation. Seemingly we should be able to combine these two types of averages to get a vector count of the spikes *across trials and over time* for various orientations.

We do this in two steps. First, make a *matrix* corresponding to the average across time (like in the PSTH) but for all orientations:

```
%COMPUTE TUNING CURVES
%next line gives matrix of counts across trials for various orientations and times
TuningCurveCounts_matrix = sum(spikes(:,((TStartCount+1)/dt):(TEndCount/dt),:),3);
```

The only difference between the above line and the TrialSum_vect calculation done in calculating the PSTH is that we now add an extra dimension (hence going from a vector to a matrix) corresponding to all orientations.

Run your code. Then, at the command prompt, let's check the size of the resulting matrix:

```
>> size(TuningCurveCounts_matrix)

ans =

        18       1900
```

As expected, we are left with a 2-dimensional matrix with the size of the 1st dimension equal to the number of orientation angles (+ 2 for the controls) and the size of the 2nd dimension equal to the number of time points.

Now let's sum this matrix across the time dimension to get the total number of spikes for each orientation across all trials and then convert this counts number to an average rate by dividing by both the number of trials and the time per trial:

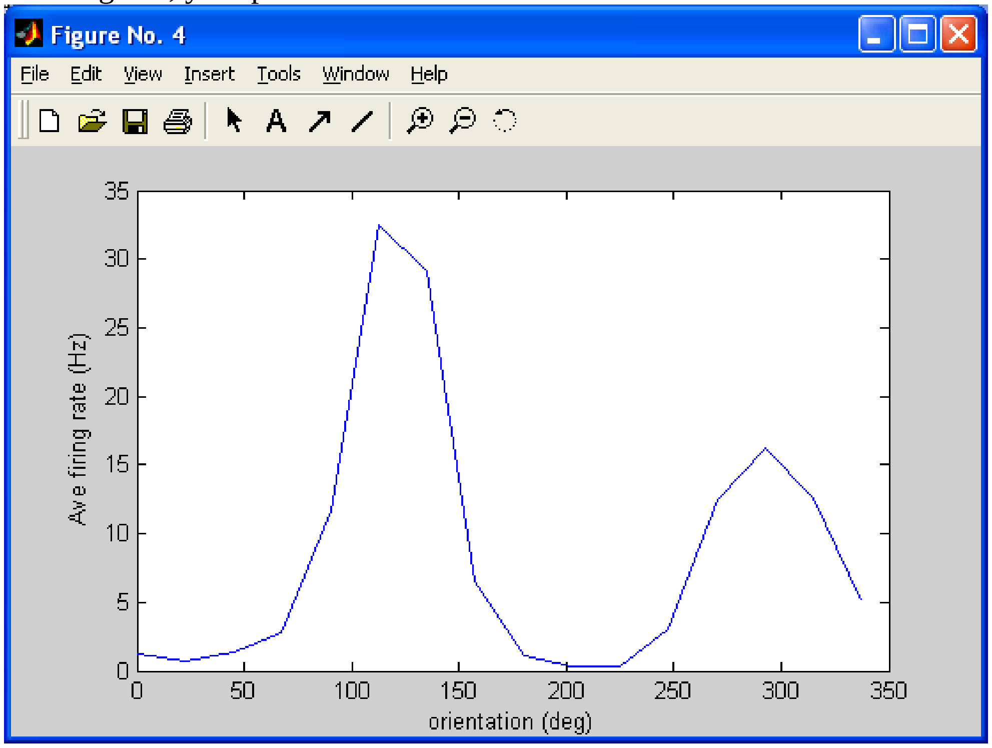
```
TuningCurveCounts_vect = sum(TuningCurveCounts_matrix,2) %sum over time dimension
TuningCurve_AveRate_vect = 1000*TuningCurveCounts_vect/((TEndCount-TStartCount)*NumTrials)
```

From the output, you should be able to see that the average rate varies considerably with the angle.

Finally, let's plot the data. To do this, we need to set up an orientation vector to hold all of the different angles tested (with spacing between angles $\Delta\theta = 360^0$/(Number of angles measured) = 22.5^0). Do this and then make the plot (note: the plot should only contain the first 16 points, as the final two are controls). Add to this section of code:

```
%plot tuning curve
DeltaTheta = 360/NumAngles
Orientation_vect = 0:DeltaTheta:(NumAngles-1)*DeltaTheta;
figure(4)
plot(Orientation_vect,TuningCurve_AveRate_vect(1:NumAngles))
xlabel('orientation (deg)')
ylabel('Ave firing rate (Hz)')
```

Running this, your plot should look like:



We see that this cell is driven best by the grating moving at an angle near 110 degrees, with a secondary preference for angles near 290 degrees. Why should it not be surprising that there are these two peaks? (How many degrees apart are they?). The two peak locations correspond to a grating with the same spatial orientation but being moved in opposite directions. The cell is driven strongly by this orientation, but responds better to motion in one direction than motion in the other direction.

Finally, we might want to consider our "control" experiment. In the control (conditions 17 and 18) there was a solid uniform light presented over the full visual field of the animal with the same average brightness as the gratings. Let's calculate the average firing rate in response to these controls.

```
Control_AveRate = mean(TuningCurve_AveRate_vect(17:18))
```

You should find that this rate is approximately 3.23 Hz. Looking at the average firing rates across all orientations in the figure above, we see that for angles around 90 degrees away from the peaks, the rate actually goes below this control rate. It seems sensible that we should really plot the *difference* between the average rates for the various angles and the control rate. Let's do this:

```
%plot difference from control average rate
TuningCurve_RateDiff_vect = TuningCurve_AveRate_vect - Control_AveRate;
figure(5)
plot(Orientation_vect,TuningCurve_RateDiff_vect(1:NumAngles))
xlabel('orientation (deg)')
ylabel('Ave firing rate relative to control(Hz)')
grid on
```

The last line adds a grid to make the plot pretty. Your final plot should look like:

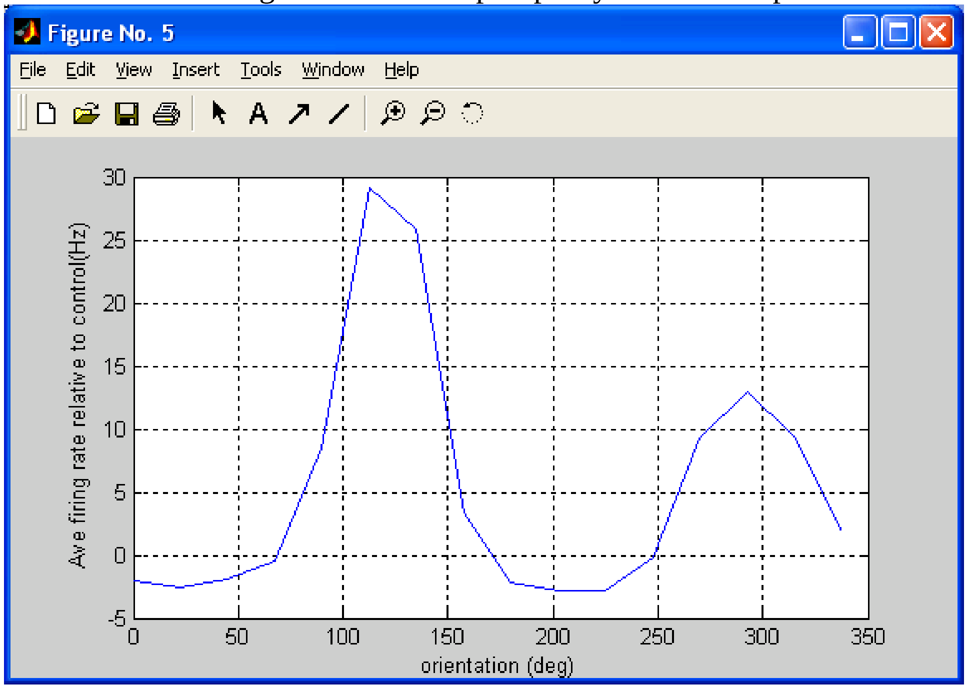


Figure No. 5
File Edit View Insert Tools Window Help
30
25
20
15
10
5
0
-5
0
50
100
150
200
250
300
350
Ave firing rate relative to control(Hz)
orientation (deg)

The areas around each peak have a classic "center-surround" arrangement: There is a peak in the response near the angle 110^0 (and near 290^0 for motion in the opposite direction), with an inhibitory "surrounding" region that in this case dips lowest around 90 degrees from the peaks. People often refer to 110^0 (and sometimes 290^0) as the "preferred" orientations of such a cell, and angles around 20^0 and 200^0 as the “cross-oriented” or “orthogonal” directions to indicate that these angles are 90 degrees away from the preferred orientations.

Note finally that, if we wanted to, we could add error bars to this graph based on the standard deviations we calculated during the Fano Factor calculation.

CONGRATULATIONS! You have written the code to perform many of the analyses that real laboratories do every day on their data!

Your final code for this exercise (please save!) should be:

```
%Analyze data from recordings of V1 cells responding to oriented
%gratings presented at various angles

clear all
close all

%LOAD IN DATA
load Sur_Orientation_SpikeData
spikes = double(spikes);

%SET UP USEFUL VARIABLES
NumControls = 2;  %number of control experiments with no grating
dt = 1;  %spacing between sampled time points [ms]
t_On = 0;  %time stimulus turns on [ms]
t_Move = 500; %time stimulus begins moving [ms]
t_Off = 2500; %time stimulus turns off [ms]
NumAngles = size(spikes,1) - NumControls  %number of angles tested, equally spaced;
                                          %last 2 sets of recordings are controls
NumTimePoints = size(spikes,2)  %number of time points; time was sampled every 1 ms
NumTrials = size(spikes,3)      %number of trials performed at each angle
t_vect = t_On:dt:(NumTimePoints-1)*dt; %time vector for each trial
ThisOrientation = 13;  %element index of orientation we are currently analyzing

%PLOT RASTERS FOR ONE PARTICULAR ANGLE
figure(1)
for trial=1:NumTrials
   plot(t_vect,trial*spikes(ThisOrientation,:,trial),'+')
   hold on
end
xlabel('time (ms)')
ylabel('trial number')
axis([0 3500 0.5 NumTrials])
hold off

%PLOT AVERAGED DATA (PSTH) FOR ONE PARTICULAR ANGLE
TrialSum_vect = sum(spikes(ThisOrientation,:,:),3);  %adds up spike trains across trials
SmoothingWidth = 61; %smooths data over +/- [(this # - 1)/2] bins
TrialSum_smooth_vect = smooth(TrialSum_vect,SmoothingWidth);
figure(2)
plot(t_vect,1000*TrialSum_smooth_vect/(NumTrials*dt))
xlabel('time (ms)')
ylabel('Average firing rate (Hz)')
```

```
%COMPUTE FANO FACTOR
TStartCount = 600; %time to start computing average
TEndCount = 2500; %time to end computing average
%next line gives number of counts for each trial
NumCounts_vect = sum(spikes(ThisOrientation,((TStartCount+1)/dt):(TEndCount/dt),:),2);
FanoFactor = (std(NumCounts_vect)^2)/mean(NumCounts_vect)

%COMPUTE CV_isi
SpikeTimes_vect = dt*find(abs(spikes(ThisOrientation,TStartCount:TEndCount,1)-1) < 0.00000001);
isi_vect = diff(SpikeTimes_vect);
%plot ISI histogram
figure(3)
hist(isi_vect,8)
xlabel('isi (ms)')
ylabel('Number of occurrences')
%compute CV_isi
mean_isi = mean(isi_vect)
std_isi = std(isi_vect)
CV_isi = mean_isi/std_isi

%COMPUTE AVE RATE FOR 1 TRIAL
TotalCounts = sum(NumCounts_vect) %total spikes across time and trials
AveRate = 1000*TotalCounts/((TEndCount-TStartCount)*NumTrials) %average across time and trials

%COMPUTE TUNING CURVES
%next line gives matrix of counts across trials for various orientations and times
TuningCurveCounts_matrix = sum(spikes(:,((TStartCount+1)/dt):(TEndCount/dt),:),3);
TuningCurveCounts_vect = sum(TuningCurveCounts_matrix,2) %sum over time dimension
TuningCurve_AveRate_vect = 1000*TuningCurveCounts_vect/((TEndCount-TStartCount)*NumTrials)
%plot tuning curve
DeltaTheta = 360/NumAngles
Orientation_vect = 0:DeltaTheta:(NumAngles-1)*DeltaTheta;
figure(4)
plot(Orientation_vect,TuningCurve_AveRate_vect(1:NumAngles))
xlabel('orientation (deg)')
ylabel('Ave firing rate (Hz)')
Control_AveRate = mean(TuningCurve_AveRate_vect(17:18))
%plot difference from control average rate
TuningCurve_RateDiff_vect = TuningCurve_AveRate_vect - Control_AveRate;
figure(5)
plot(Orientation_vect,TuningCurve_RateDiff_vect(1:NumAngles))
xlabel('orientation (deg)')
ylabel('Ave firing rate relative to control(Hz)')
grid on
```