Read the text file with the fluorescence data obtained from ImageJ into MATLAB as a matrix of data, then transpose the data.

```
rawData = (readmatrix('Results.txt'))';
```

Now we have a matrix that contains our fluorescence values. Check the number of rows and columns of this matrix using the size function.

```
rows = size(rawData,1)
```

```
rows = 74
```

```
cols = size(rawData,2)
```

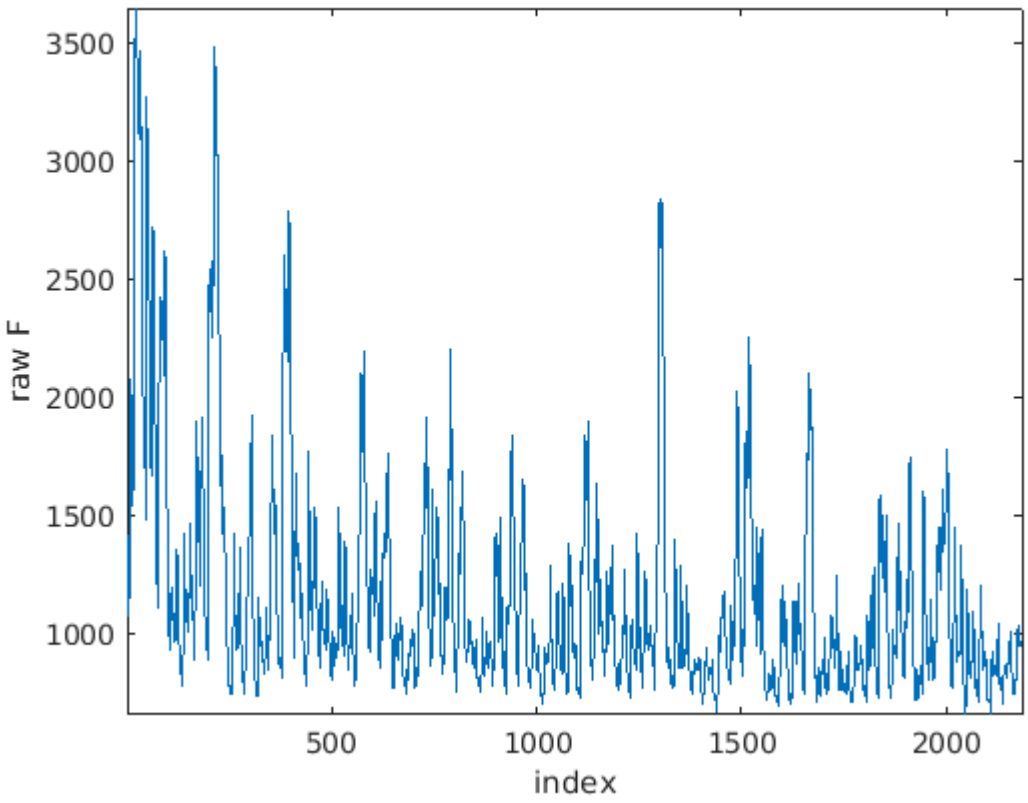
```
cols = 2184
```

The first row of our rawData matrix does not contain fluorescence data, so we will remove it. Write code that indexes the `rawData` matrix by selecting ALL other rows except the first one, and stores the values into a new matrix `rawF`.

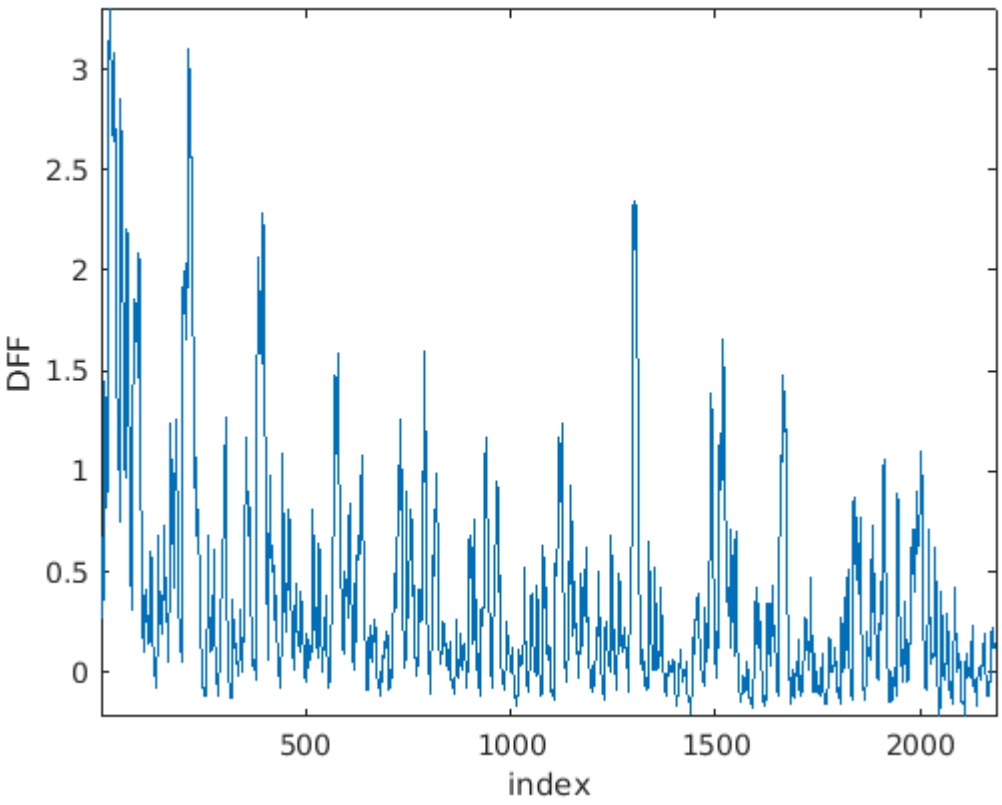
```
rawF = rawData(2:end,:);
```

Calculate DF/F for one cell ((F_raw - F_baseline)/F_baseline) and store it in a vector named `dff_cell`. To visualize your results, plot the raw fluorescence trace **and** the DF/F of your cell as two different figures.

```
cell = 1; % this is the cell we will extract
rawF_cell = rawF(cell,:);

% round to nearest multiple of 10
rawF_rounded = round(rawF_cell/10)*10;
baseline = mode(rawF_rounded);

% calculate df/f
dff_cell = (rawF_cell-baseline)/baseline;


figure()
plot(1:size(rawF_cell,2),rawF_cell)
xlabel('index')
ylabel('raw F')
axis tight
```




```
figure()
plot(1:size(rawF_cell,2),dff_cell)
xlabel('index')
ylabel('DFF')
axis tight
```



Generate a matrix that contains the DF/F values for all cells. This matrix `DFF` should be the same size as the `rawF` matrix.

```
rawF_rounded = round(rawF/10)*10;
baseline = mode(rawF_rounded,2);
DFF = (rawF-baseline)./baseline;
```

Now let's begin analyzing the responses of a single cell. First we will plot the average timecourse across all six trials. After interpolation, our frame rate is 5fps. Each cell therefore has a DFF trace that is 2880 samples (or frames) long (5x576).

```
load('DFF.mat')

FrameRate = 5;
SamplesPerTrial = 480;
NumTrials = 6;
cell = 1; % this is the cell that we will analyze
```

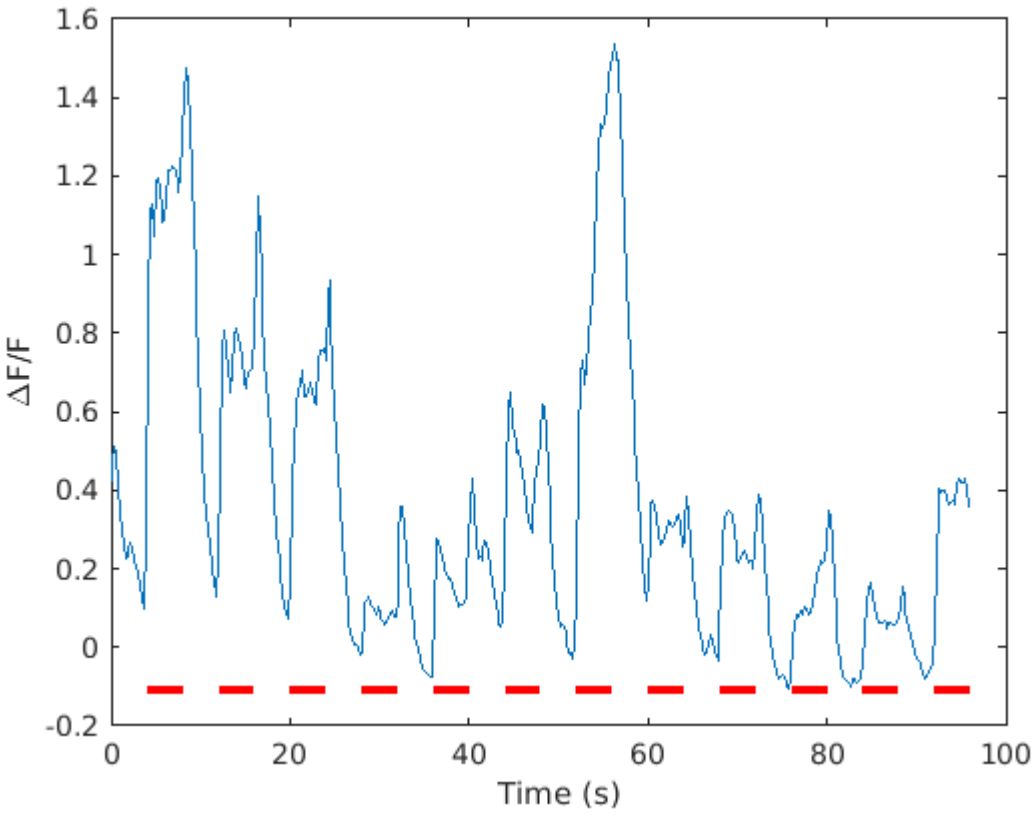
After extracting the row vector corresponding to cell #1, we need to rearrange our data into a 2-D matrix with size **SamplesPerTrial x NumTrials**.

Use `reshape`, and then `mean(dff_reshaped, dim)` to get the trial-averaged data over your chosen dimension `dim`.

```
dff_cell = DFF(cell,:); % extract dff trace for our cell
dff_reshaped = reshape(dff_cell,480,6);
dff_avg = mean(dff_reshaped,2);
```

Plot the trial-averaged response of cell #1.

```
% plot dff_avg
t = (0:SamplesPerTrial-1)/FrameRate;
plot(t,dff_avg)
hold on;

% plot bars indicating ON period
for i = 1:12
 plot([8*i-4,8*i],min(dff_avg)*ones(1,2),'r','LineWidth',3)
end
xlabel('Time (s)')
ylabel('\DeltaF/F')
```


1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
-0.2
ΔF/F
0
20
40
60
80
100
Time (s)

Now that we have looked at the trial-averaged response, we have an idea of which orientations the neuron is most responsive to. To quantify the response of the cell to each orientation, we need to calculate the **orientation tuning curve**. This curve is generated by calculating the mean response of the neuron during the ON period, and comparing with its response during the OFF period.

To do this we will again start by extracting a 1x2880 DF/F vector for our one cell, and reshape this vector into a 3D matrix of size **SamplesPerOri x NumOrientations x NumTrials**.

```
load('DFF.mat');

SamplesPerOri = 40;
NumOrientations = 12;
NumTrials = 6;
cell = 1; % this is the cell that we will analyze

dff_cell = DFF(cell,:); % extract dff trace for our cell
dff_reshaped = reshape(dff_cell,SamplesPerOri,NumOrientations,NumTrials);
```

Define ON and OFF periods as vectors extracted with appropriate indices. Extract the time-averaged response for each orientation and trial, and store them into `AveON` and `AveOFF`. Do NOT average over trials (`AveON` and `AveOFF`) will be 2D.

```
ON_period = 21:40;
OFF_period = 11:20;

AveON = mean(dff_reshaped(ON_period,:,:), 1);
AveOFF = mean(dff_reshaped(OFF_period,:,:), 1);

AveON = squeeze(AveON);
AveOFF = squeeze(AveOFF);
```

Calculate the **mean** and **standard error** (over trials) of the `AveON` matrix, assigned to `ON_mean` and `ON_sem` respectively. These values will be used to plot the tuning curve, with error bars.

```
Orientations = 0:30:330;

ON_mean = mean(AveON,2);
ON_sem = std(AveON,[],2)/sqrt(NumTrials);
```

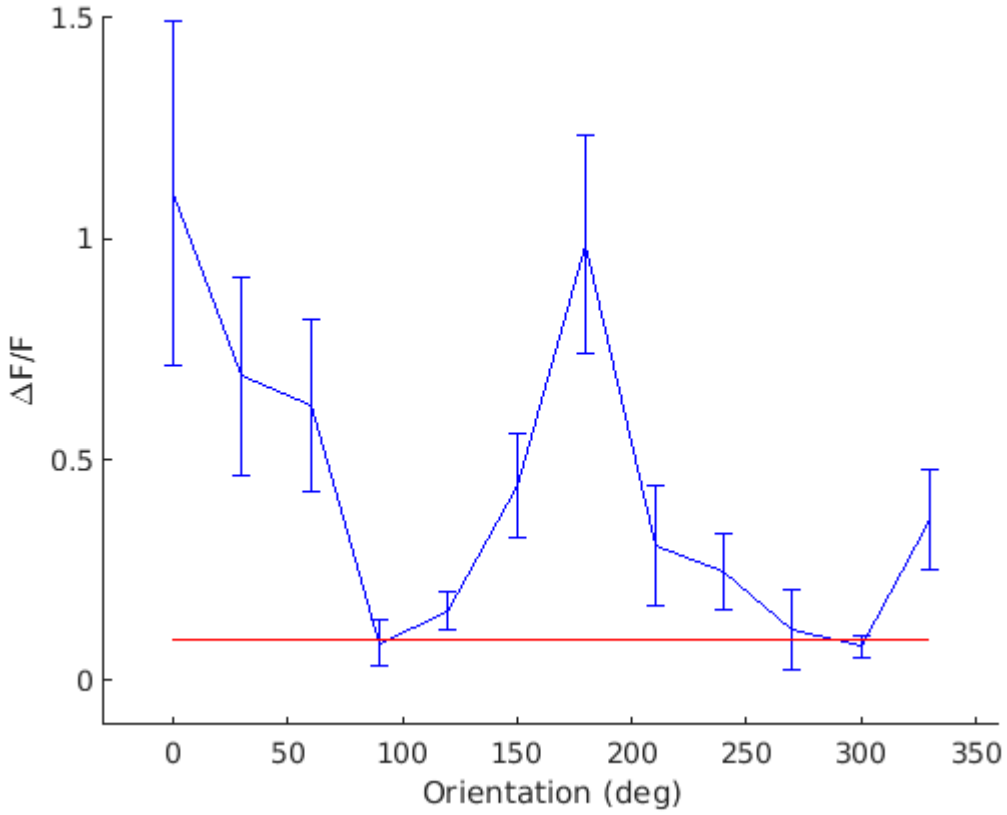
The `AveOFF` matrix will be used to estimate the baseline (i.e. spontaneous) activity. Calculate a single value by averaging over **both** dimensions (orientations and trials), assigning this value to `OFF_mean`. This baseline will then be plotted as a horizontal line, given by the assignment of `OFF_line`.

```
OFF_mean = mean(AveOFF(:));
OFF_line = OFF_mean*ones(size(Orientations)); % this generates a baseline you can plot
```

The tuning curve is now plotted.

```
figure
```

```
hold on
errorbar(Orientations,ON_mean,ON_sem,'b') % plot ON tuning curve
plot(Orientations,OFF_line,'r') % plot OFF spontaneous activity
xlim([-30 360])
ylim([min(ylim)-0.1, max(ylim)])
xlabel('Orientation (deg)')
ylabel('\DeltaF/F')
```



Now that we have calculated and plotted the tuning curve for one cell, it should be relatively straightforward to replicate our code from Step 4 for all cells. The key difference is that `AveON` and `AveOFF` need to be 3-dimensional matrices, with size **NumCells x NumOrientations x NumTrials.**

```
load('DFF.mat');

NumCells = size(DFF,1);

SamplesPerOri = 40;
NumOrientations = 12;
NumTrials = 6;

% initialize matrices
AveON = zeros(NumCells,NumOrientations,NumTrials);
AveOFF = zeros(NumCells,NumOrientations,NumTrials);
```

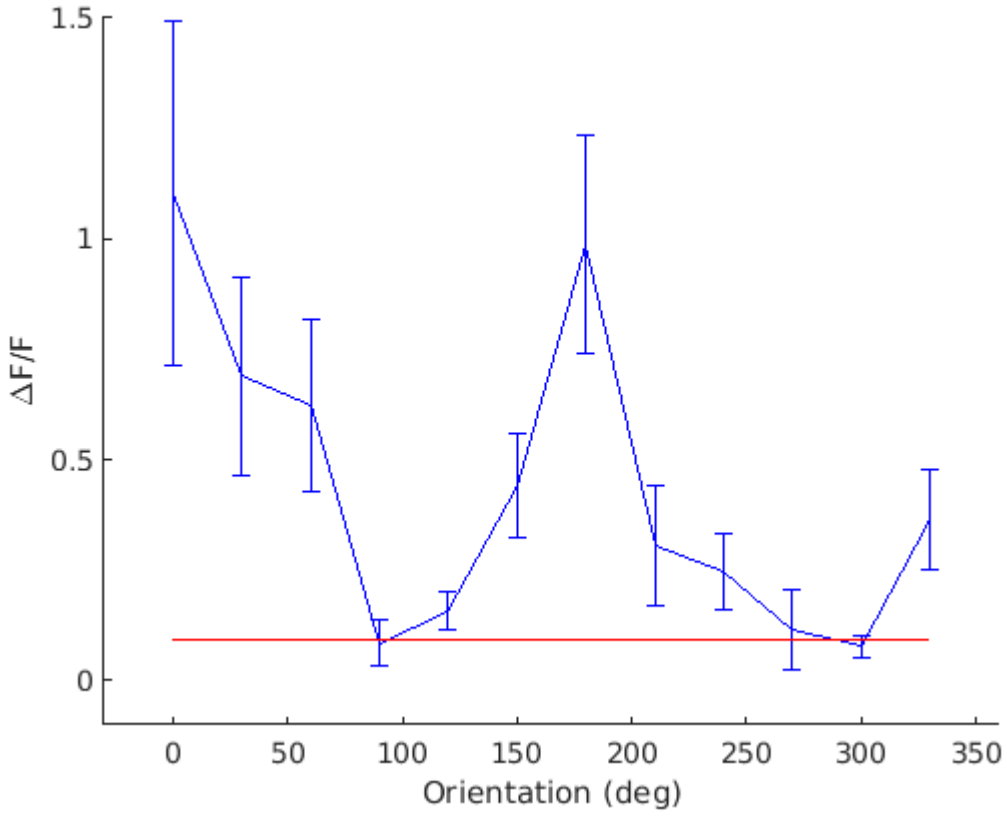
Write a `for` loop that extracts the time-avergaed response matrices for each cell.

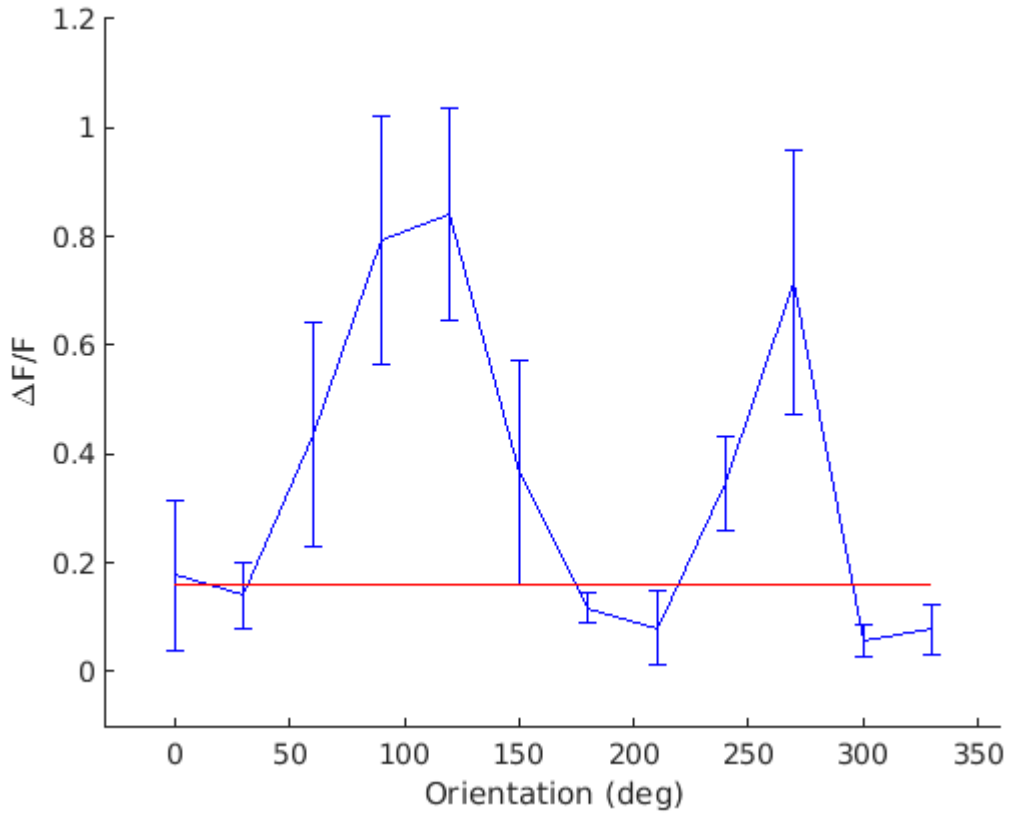
```
for cell = 1:NumCells

    dff_cell = DFF(cell,:); % extract dff trace for our cell
    dff_reshaped = reshape(dff_cell,SamplesPerOri,NumOrientations,NumTrials);

    ON_period = 21:40;
    OFF_period = 11:20;

    cell_AveON = mean(dff_reshaped(ON_period,:,:), 1);
    cell_AveOFF = mean(dff_reshaped(OFF_period,:,:), 1);

    AveON(cell,:,:) = cell_AveON;
    AveOFF(cell,:,:) = cell_AveOFF;
end
```

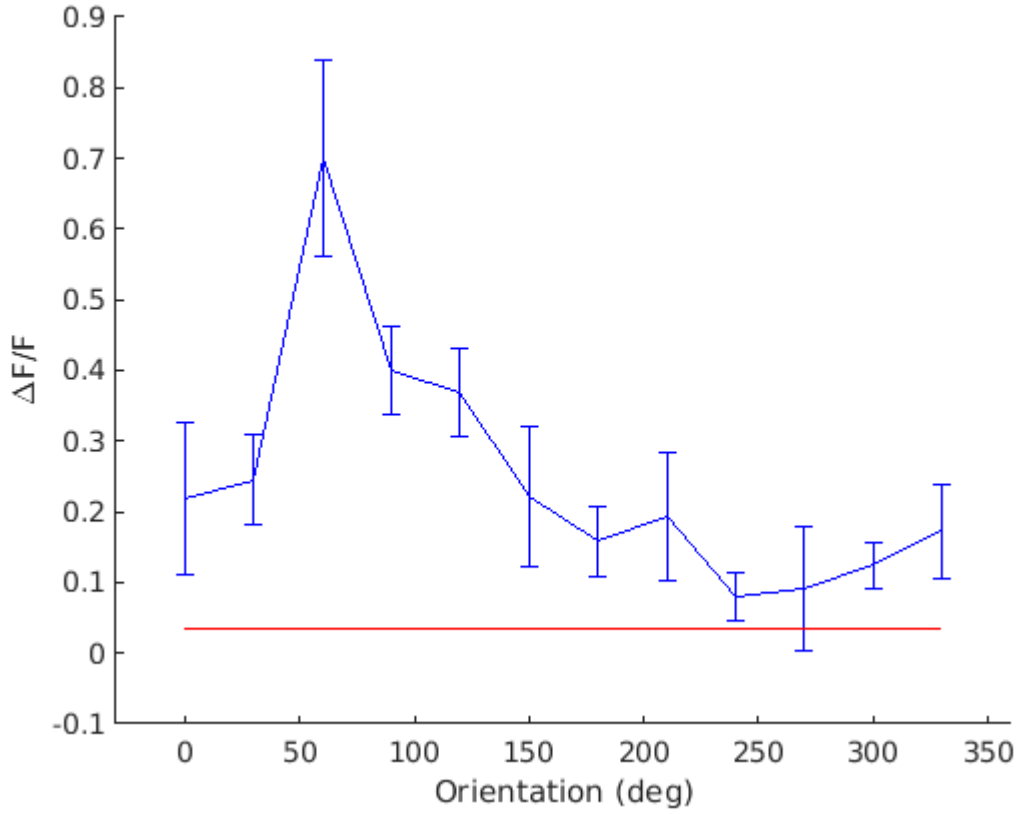
Using the new 3D `AveON` and `AveOFF` matrices, plot tuning curves for 8 of your cells. Assign `ON_mean` and `ON_sem` as 2D matrices (size NumCells x NumOrientations), and assign `OFF_mean` as a one-dimensional vector (size NumCells). Don't forget to assign your baseline as `OFF_line`.

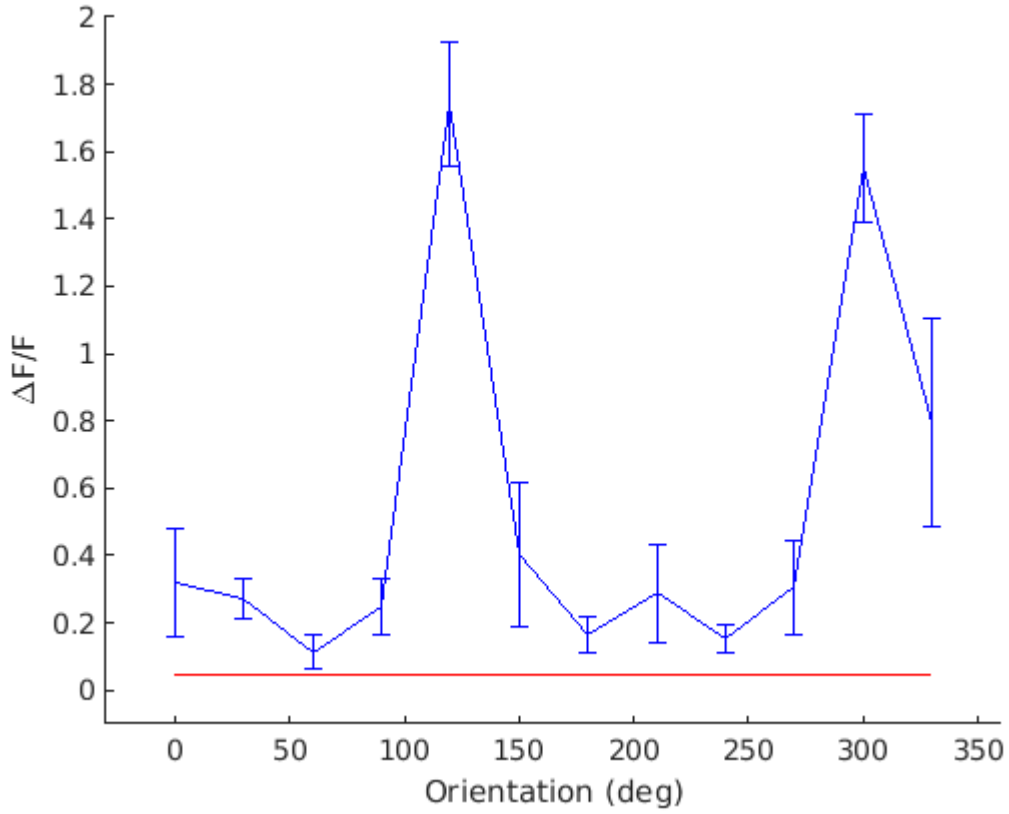
```
Orientations = 0:30:330;

ON_mean = mean(AveON,3);
ON_sem = std(AveON,[],3)/sqrt(NumTrials);

OFF_mean = mean(mean(AveOFF,3),2);

for cell = 1:8
    figure()

    OFF_line = OFF_mean(cell)*ones(size(Orientations)); % this generates a baseline you

 hold on
 errorbar(Orientations,ON_mean(cell,:),ON_sem(cell,:),'b') % plot ON tuning curve
```

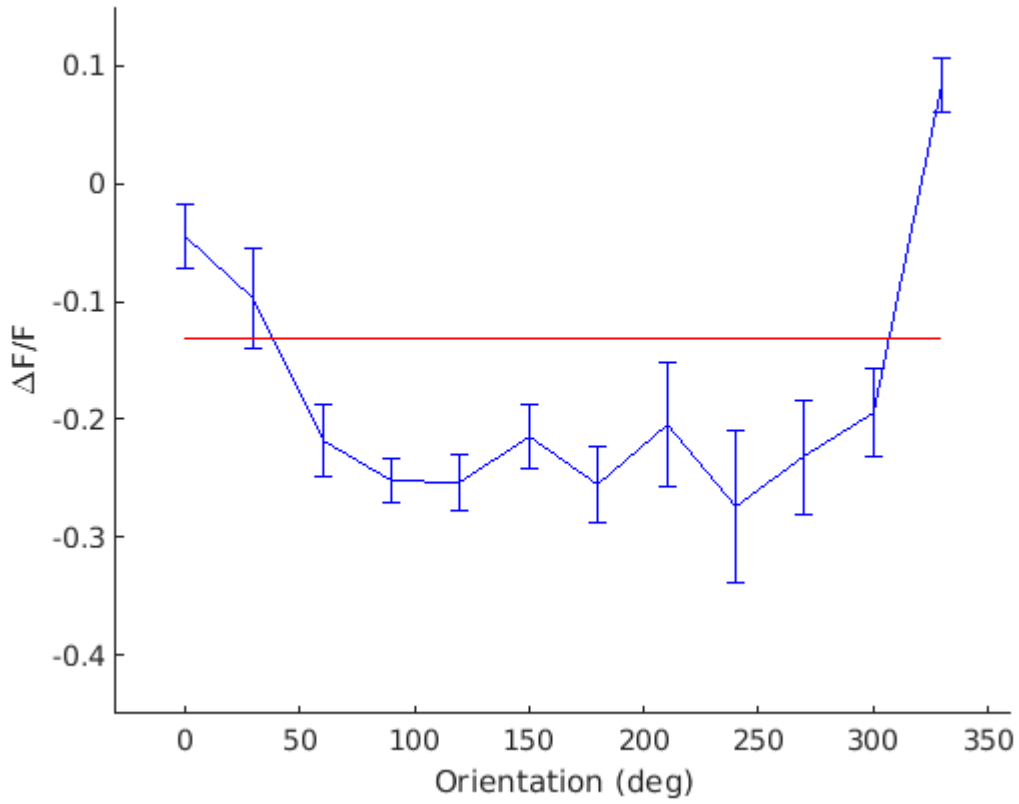
```
    plot(Orientations,OFF_line,'r') % plot OFF spontaneous activity
 xlim([-30 360])
 ylim([min(ylim)-0.1, max(ylim)])
        xlabel('Orientation (deg)')
        ylabel('\DeltaF/F')
end
```

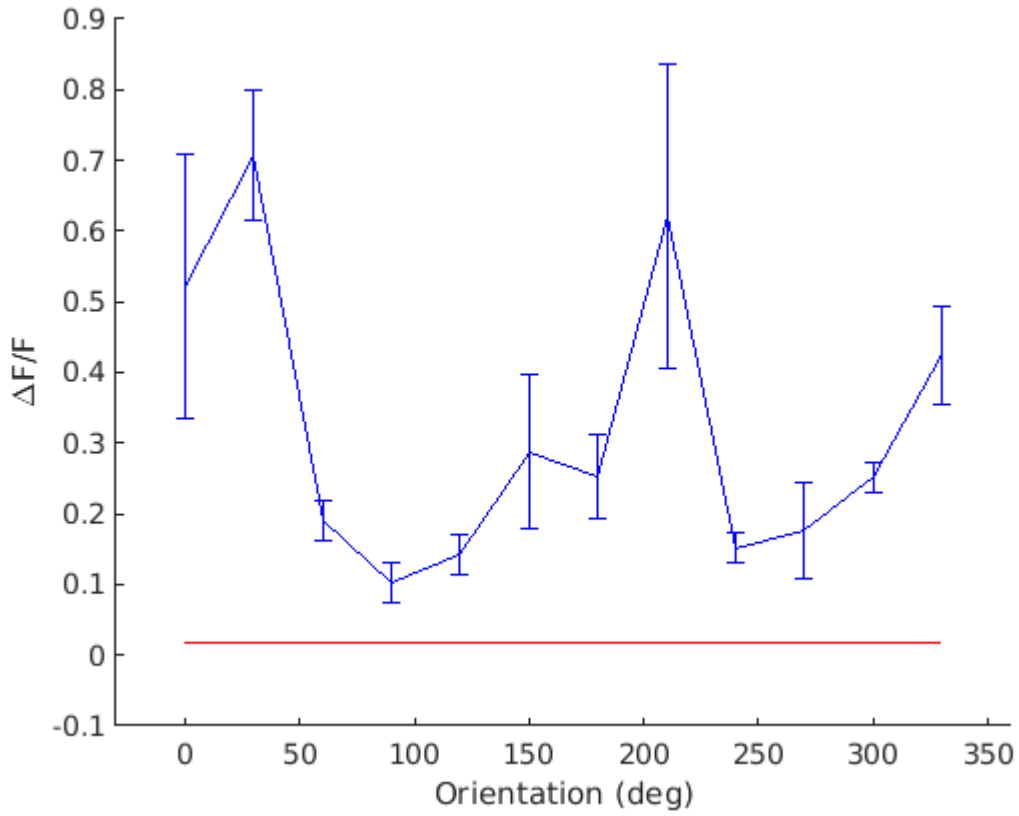


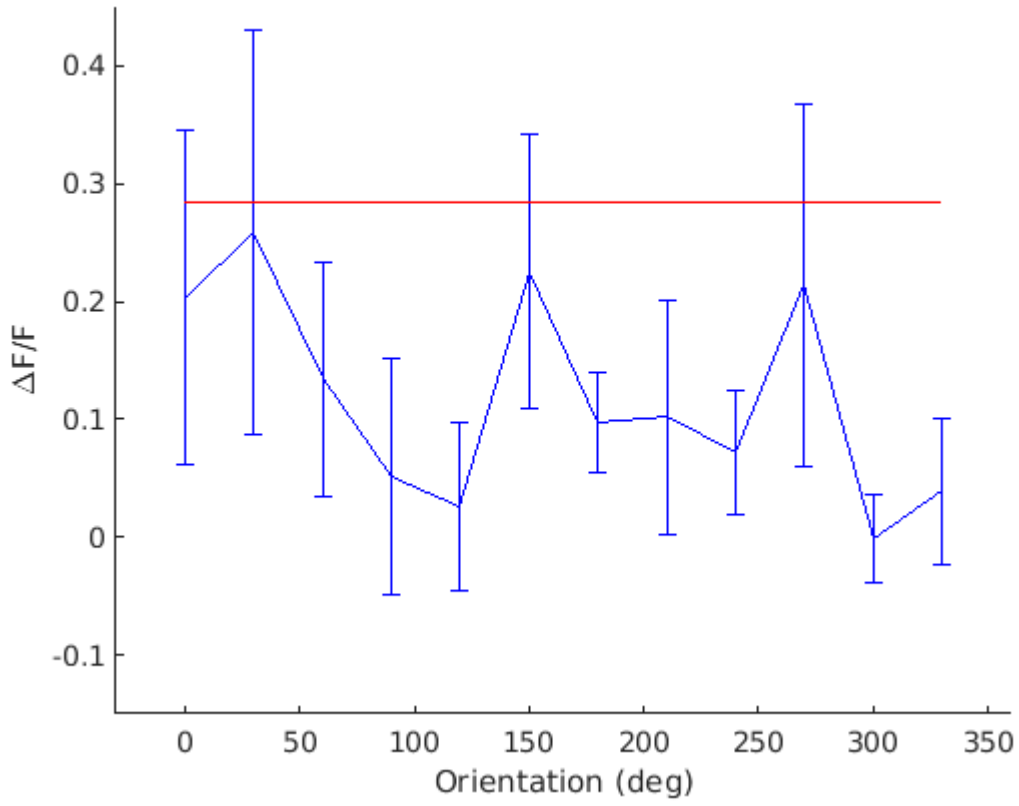

1.2
1
0.8
0.6
0.4
0.2
0
ΔF/F
0
50
100
150
200
250
300
350
Orientation (deg)

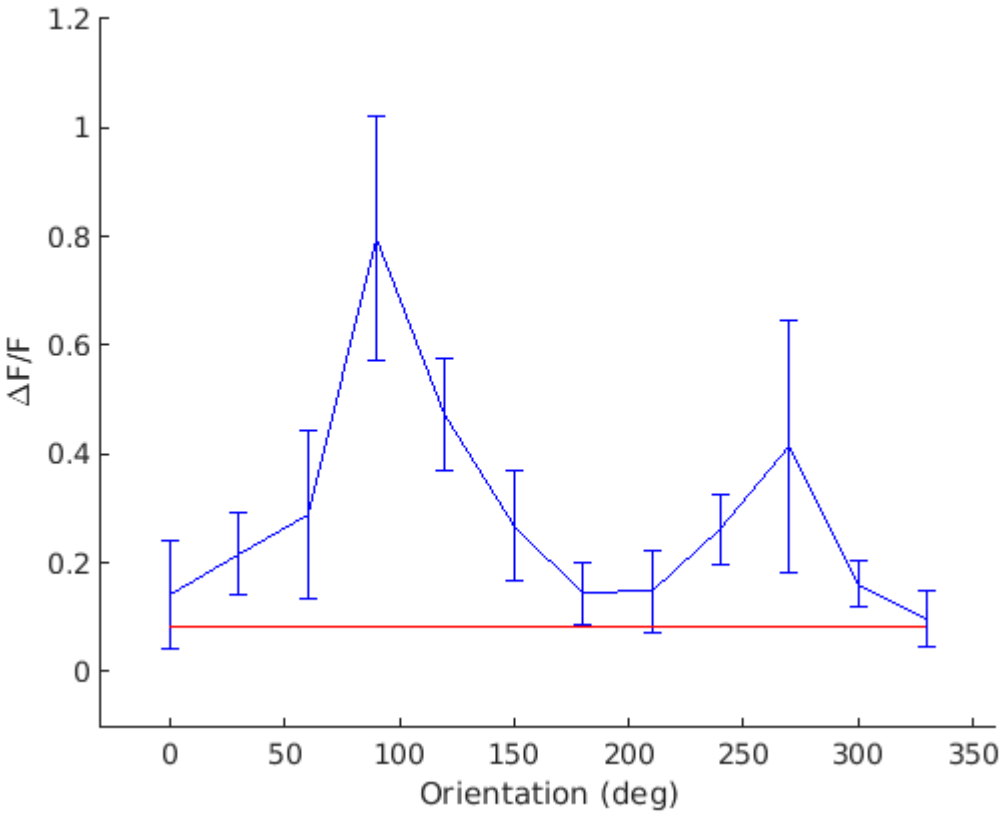

0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-0.1
ΔF/F
0
50
100
150
200
250
300
350
Orientation (deg)


2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
ΔF/F
0
50
100
150
200
250
300
350
Orientation (deg)


0.1
0
-0.1
-0.2
-0.3
-0.4
ΔF/F
0
50
100
150
200
250
300
350
Orientation (deg)


0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-0.1
ΔF/F
0
50
100
150
200
250
300
350
Orientation (deg)


0.4
0.3
0.2
0.1
0
-0.1
ΔF/F
0
50
100
150
200
250
300
350
Orientation (deg)


1.2
1
0.8
0.6
0.4
0.2
0
ΔF/F
0
50
100
150
200
250
300
350
Orientation (deg)

Now we have plotted the tuning curves for all cells in our population. Let's use some statistical measures to characterize the properties of the neurons in our population.

Using the AveON and AveOFF matrices, we will measure two properties for each neuron: its **preferred orientation** and its **orientation selectivity index (OSI)**.

```
load('TC.mat');

NumOrientations = 12;
```

Before calculating these two properties, we need to remove all ON responses that were weaker than the average OFF response. To do this, we will subtract `OFF_mean` from the `ON_mean` matrix.

To replicate the 1D `OFF_mean` vector across NumOrientations columns so that the subtraction can be done, the `repmat` function is used.

```
OFF_mean = repmat(OFF_mean,1,NumOrientations);
TC = ON_mean - OFF_mean;
```

After subtracting the two matrices and storing the tuning curve in `TC`, remove all negative values using logical indexing, and replace them with zeroes.

```
TC(TC<0) = 0;
```

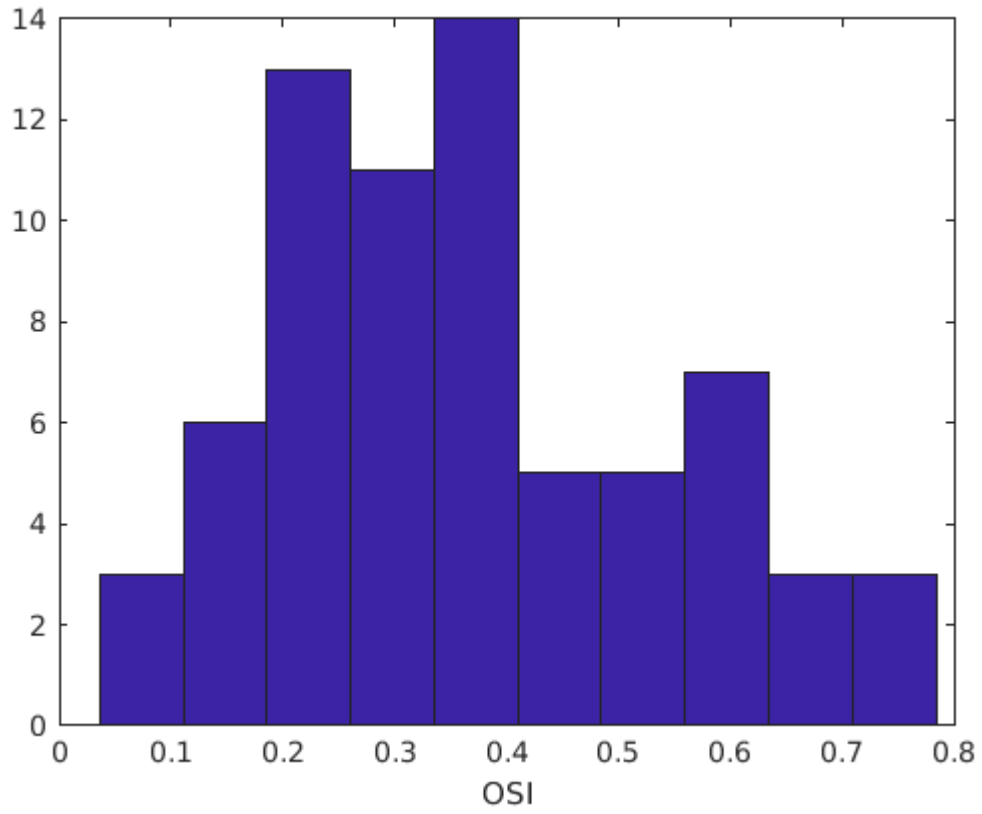
Write a `for` loop that calculates the PO and OSI for each neuron, , and stores the values for each property into a separate vector (`AllPO` and `AllOSI` respectively).

```
NumCells = size(TC,1);
Orientations = 0:30:330;
AngRad = Orientations*pi/180;

% initialize your vectors
AllOSI = zeros(1,NumCells);
AllPO = zeros(1,NumCells);

for cell = 1:NumCells

    TC_cell = TC(cell,:);

    OSI = abs(sum(TC_cell.*exp(2.*1i.*AngRad))./sum(TC_cell));

    PrefOri = 0.5.*(angle(sum(TC_cell.*exp(2.*1i.*AngRad))));
    PO = PrefOri.*(180/pi);
    if PO<0
        PO = PO + 180;
    end

    AllOSI(cell) = OSI;
    AllPO(cell) = PO;
```

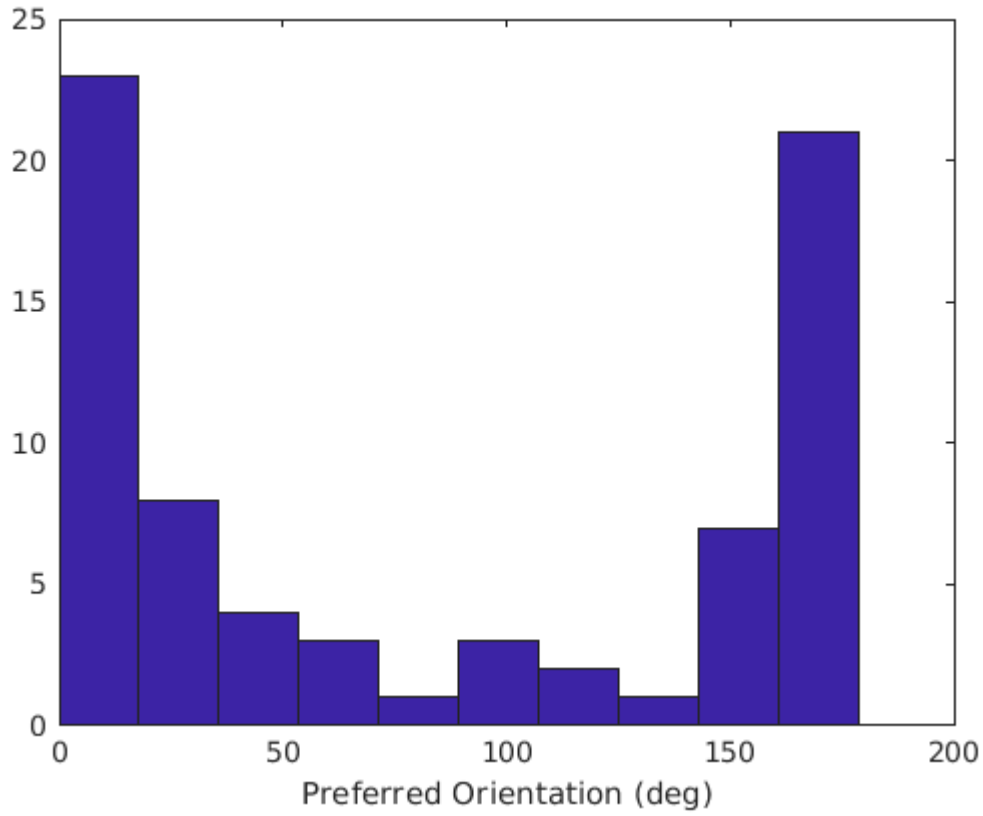
```
end
```

Plot two histograms: one for Preferred Orientation and one for OSI.

```
figure()
hist(AllOSI)
xlabel('OSI')
```



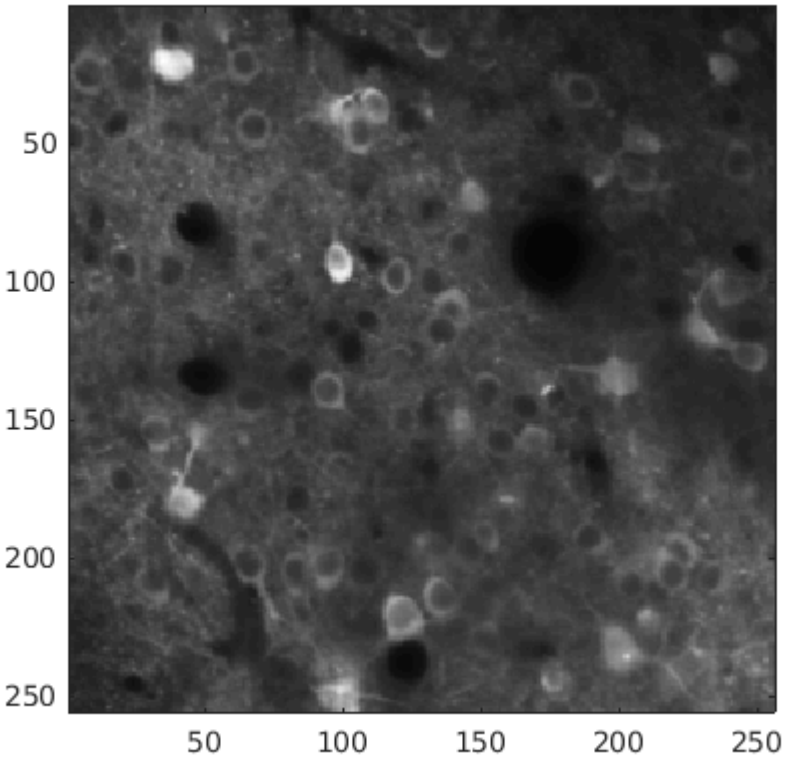

```
figure()
hist(AllPO)
xlabel('Preferred Orientation (deg)')
```


25
20
15
10
5
0
0
50
100
150
200
Preferred Orientation (deg)

In the final step of this exercise, we utilize the spatial information that is uniquely available to us using calcium imaging. Imaging, unlike electrophysiology, allows us to visualize the location of each neuron, and therefore we can make a image that demonstrates the relationship between orientation tuning and location in the brain.

```
load('PopMap.mat');

% Variables available:
% AllOSI
% AllPO
% TC
% img, the average projection image you made in Image J
% ROIs, a vector of region of interest data for each cell
```

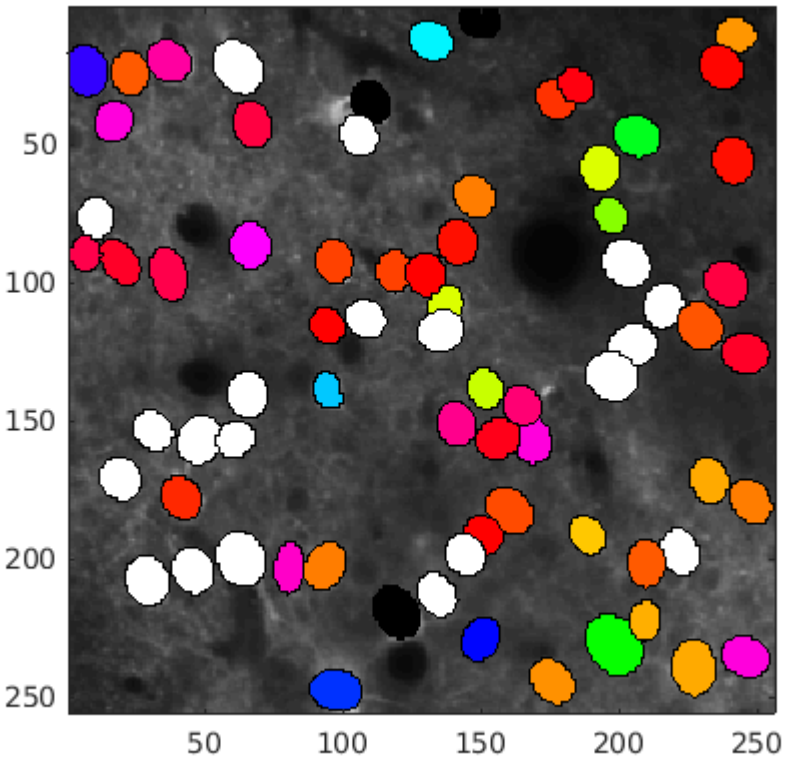
```
NumCells = size(TC,1);

% display average projection image
image(img)
axis square
```




```
% setup colors
color = hsv(180);
```

Write a for loop across all cells that determines the color based on the neuron's PO. We will plot cells with zero/ negative response in black, cells with OSI less than 0.25 in white, and all other cells color-coded to their PO.

```
% loop for each cell
for cell = 1:NumCells

    % get ROI coordinates
    roi = ROIs{cell}.mnCoordinates;

    % color based on preferred orientation
    if max(TC(cell,:)) == 0  % not visually responsive = black
        roicolor = zeros(1,3);

    elseif AllOSI(cell)<0.25 % broadly tuned = white
        roicolor = ones(1,3);

    else % not visually responsive
        roicolor = color(ceil(AllPO(cell)),:);

    end

    % draw colored ROI
    patch(roi(:,1),roi(:,2),ones(1,size(roi,1)),'FaceColor',roicolor)
end
```



Plot the final population map.

```
colormap(color)
colorbar('Ticks',[0:60:360]/180, 'TickLAbels',0:30:180)
axis off
title('Population Orientation Map')
```