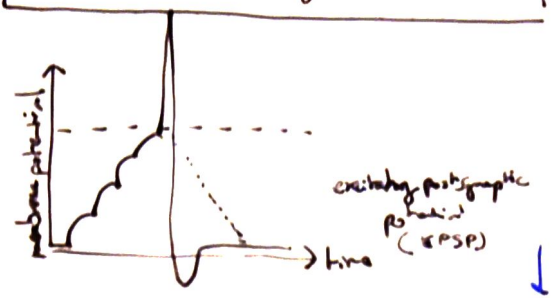
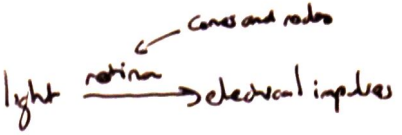
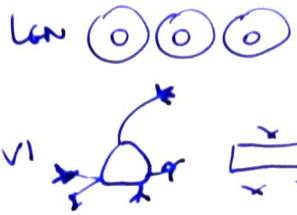
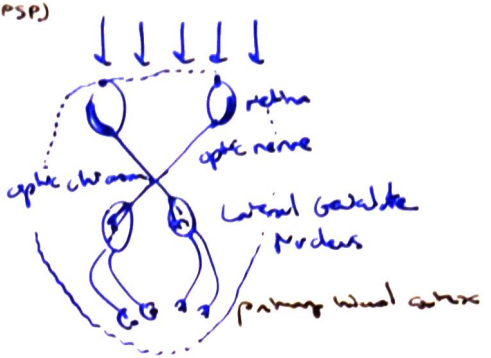
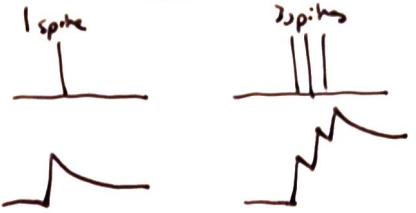
Neurosciences : ImageJ and MatLab











two-photon calcium imaging

Orientation selectivity of neurons in V1

ImageJ >> File >> Open... stack.tif ou video

Image >> Stacks >> Z Project...
- → Average Intensity
- → Standard Deviation

} detect neurons that had response/activity

Analyze >> Tools >> Roi Manager...

Draw Oval/Elliptical selection to each cell in the avg/std projection

→ Add [t] Combine with all ROIs

then "Show All" once back on the original window

Analyze >> Set Measurements : Mean gray value only

Select all ROIs [Ctrl+A] >> More >> Multi Measure >> OK

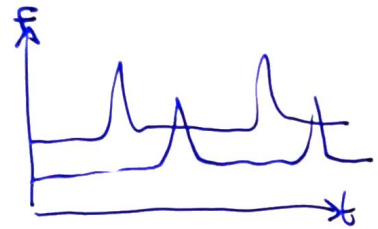
=> File >> Save as... data.txt

% import data with mixed structure (not a matrix) = table

```
rawF = readtable("RawF.txt");
head(rawF)
% T = table([1;4;3],[4;5;6])
% column → variable
% row → observation
% [rows,cols] = size(T)  or  rows = size(T,1)
[rows, cols] = size(rawF)
% rows: the total number of frames required to achieve rate of 5Hz
% columns: the total number of ROIs + four columns with information

% Subsetting data
% rows = [1 5];
% cols = 2:4;
% subdata = data(rows,cols);
% col1 = data(:,1);  col1 = data(1:6,1);

% A = [1,2,3; 4,5,6; 7,8,9]
% T = array2table(A, "VariableNames",["Var1","Var2","Var3"]);
% T(:,1)  > returns a 3x1 table
% T{:,1}  > returns the data in a vector
% T.Var1
% extract data for first cell called "Cell1"
rawF_vector = rawF{:,5},  % could also use rawF{:,"Cell1"};
figure
plot(rawF_vector)
% extract data for all cells
rawF_matrix = rawF{:,5:77};  % also rawF{:,5:end}   (end = last element)
figure
imagesc(rawF_matrix)
colorbar
```

```
% vector arithmetic
% x = -2:0.1:2
% y = 3*x.^2 +2*x-6;
% plot(x,y)

% Statistics
% testAvg = mean(scores)
% studentAvg = mean(scores,2)
% totalAvg = mean(scores(:)); turns the matrix into a single column
% testMax = max(scores)
% classMin = min(scores(:))
% testSTD = std(scores,0,2)
```

	test 1	2	3
student 1	.	.	.
student 2		.	.
student 3		.	.
student n	.	.	.

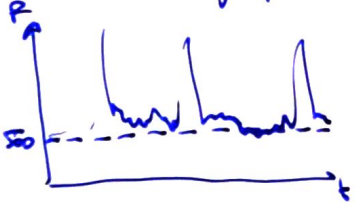


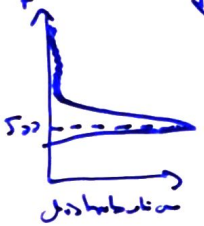
dye in fluorsance to baseline

$\Delta F/F = DFF = \frac{F_{raw} - F_{baseline}}{F_{baseline}}$

baseline: highpass filter → mode

moving average is another option






```
> histogram(rawF_vector,...
    NumBins = 1000,...
    DisplayStyle = "stairs")
```

```
rawF_rounded = round(rawF_vector/10)*10; % round to nearest multiple of
baseline = mode(rawF_rounded)
DFF_vector = (rawF_vector - baseline)/baseline
% rawF_rounded should only be used for calculate the baseline

plot(DFF_vector)
xlabel('Row')
ylabel('\DeltaF/F')
title('DF/F values of Cell 1')
```

% Save with a matrix with each column a different cell

raw_rounded = round(raw_matrix/10)*10;

baseline = mode(raw_rounded, 1); % for each column/cell

DFF_matrix = (raw_matrix - baseline)./baseline; % determine

% Create a copy of the table

DFF = raw;

% overwrite raw fluorescence with DFF values

DFF{:, 5:end} = DFF_matrix;

% slicing data on condition

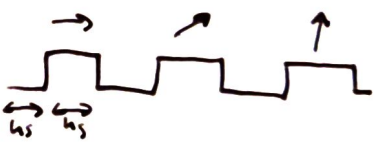
ind = data <= 0.1;

inactivity = data(ind)

% assign new values, e.g., give 0 to values <=0.1

data(ind) = 0;

Trial	Time	Orientation	Cycle	Cell1	Cell2
1	0.0	0	OFF	...	...
1	0.2	0	ON	...	..
1	0.4	30	OFF	...	...
1	0.6	30	ON	...	...
2	0.0	0	OFF	...	.
..	...	...	...	...	...



orientations = 0:30:330; % degrees

trials = 1:6;

cycles = ["", "OFF", "ON", "ON"]

- 12 orientations * 8 seconds = 96 seconds for one trial
- 12 orientations * 8 seconds * 6 repeats = 576 seconds total
- rad(30/8) = 4. → 4th stimulus is 90° seen at t=30s

Grouping

isOn = DFF.Orientation == 180 & DFF.Cycle == "ON"; % string array

cellOn = DFF{isOn, "Cell1"}; % returns a vector

meanOn = mean(cellOn)

```
orientations = 0:30:330;
tuningCurve = [];
for i = 1:length(orientations)
    isOn = DF.Orientation == orientations(i) & DF.Cycle == "ON";
    tuningCurve(i) = mean(DF.Cell1(isOn));
end
```

results = groupsummary(dataTable, groupVariables, summaryStats, dataVariables)

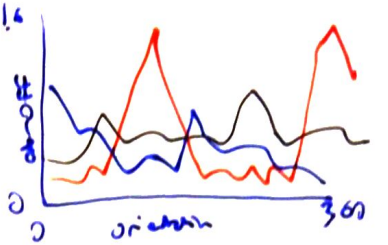
results = groupsummary(DF, "Trial", "mean", "Cell1")

6x2 Table

Trial	GroupCount	mean_Cell1
1	480	0.82476
2	480	0.37564
3	480	⋮
4	480	⋮
5	480	
6	480	

```
% the orientation tuning curve is the average response of a neuron across
% trials and across times. we need to calculate values for each stimulus
% condition, such as the orientation and cycle.

% number of trials for std error statistics
numTrials = 6;

% get summary statistics
results = groupsummary(...
    DF, ...
    ["orientation", "Cycle"], ...
    ["mean", "std"], ...
    ["Cell1"]);
meanCellOn = results{results.Cycle == "ON", "mean_Cell1"};
StdCellOn = results{results.Cycle == "ON", "std_Cell1"};
stdErrorCellOn = stdCellOn / sqrt(numTrials);

% same with mean of Cell1 during the OFF cycle, i.e., averaged across all orientations
meanCellOff = mean(results{results.Cycle == "OFF", "mean_Cell1"});

figure
errorbar(orientations, meanCellOn, stdErrorCellOn)
yline(meanCellOff(1), "--")
    legend("ON", "OFF")
    xlabel("orientation (deg)"); xlim([-30 360])
```

```
% save 1st for all cells ; i.e, all columns
% Get away with the variable names
cellNames = "Cell" + (1:73);

% Get average response for all cells
results = grpsummary(...
    OPF, ...
    ["orientation", "Cycle"], ...
    ["mean"], ... % no std required here
    cellNames);

% extract mean ON responses → 12 x 73 matrix
rows = results.Cycle == "ON";
columns = "mean_" + cellNames;
meanOn = results{rows, columns};

% extract mean OFF → 1x73 vector, averaged across orientations (rows)
rows = results.Cycle == "OFF";
meanOff = mean(results{rows, columns});

% Create tuning curves matrix
% remove all ON responses that were smaller than avg OFF,
% substract ON responses by the OFF and replace negative values with zeros
tuningCurves = meanOn - meanOff
tuningCurves(tuningCurves < 0) = 0;

% plot some tuning curves
cellsToPlot = 1:6;
orientations = 0:30:330;
plot(orientations, tuningCurves(:, cellsToPlot), 'LineWidth', 1.5);
```



- preferred orientation (PO)
- orientation selectivity index (OSI)
  - → no preference (0)
  - → only one orientation (1)


```
load TC.mat tuningCurves

numCells = size(tuningCurves, 2);
orientations = 0:30:330;
OSI = zeros(1, numCells);
PO = zeros(1, numCells);

for i = 1:numCells
    [PO(i), OSI(i)] = preferredOrientation(orientations, ...
                                           tuningCurves(:,i));
end
```

custom function

```
% function [PO,OSI] = preferredOrientation(orientations, tuningCurves)
orientationsRad = orientations*pi/180;
Z = sum(tuningCurves.*exp(2*1i*orientationsRad));
OSI = abs(Z)./sum(tuningCurves);

PORad = 0.5*angle(Z);
PO = PORad*180/pi;
PO(PO<0) = PO(PO<0) + 180;
```

```
    arguments
        orientations (:,1) double
        tuningCurves double {mustMatchRows(orientations, tuningCurves)}
    end

function mustMatchRows(a,b)
    if size(a,1) ~= size(b,1)
        throwAsCaller(MException('',...
                '...'));
    end
end
```

```
%Generate Heatmap
    color ← neuron's preferred orientation ; zero = black
load RoiMap.mat img Rois OSI PO tuningCurves

% display average projection image
imagesc(img)
axis square
colors = hsv(180);  % colormap, 180x3 matrix, 1 row for each degree

for i=1:numCells  % loop for each cell
        roi = Rois{i}.mnCoordinates;  % get roi coordinates
        if all(tuningCurves(:,i)==0)  % also, if isnan(OSI(:))
            roiColor = zeros(1,3);  % color black unresponsive cells
        elseif OSI(i) < .25
            roiColor = ones(1,3);  % broadly-tuned neurons
        else
            POind = ceil(PO(i));  % need to round to get an integer
            roiColor = colors(POind,:);
        end
    patch(roi(:,1), roi(:,2), roiColor)  % draw a colored ROI
end
```