Overview

Calcium Imaging Analysis of Visually-Evoked Responses in the Zebrafish Tectum

Ruthazer Lab

By: Niklas Brake

Contents

Introduction

In 2016-2017, the following code was written for the purposes of showing visual stimuli to larval zebrafish (6 dpf) with nuclear-targeted GCaMP and analysing cellular responses by processing images collected by 2-photon microscopy.

https://github.com/RuthazerLab/Stimulus-Presentation-and-Analysis

If you need to know what a function does, most of them have a help function. Type help *functionName* and it will give you a brief description of what it does and how to use it. I've also included a rudimentary table of the function you may want to use in the second Code Overview.

This document outlines the analysis pipeline including a description of the output files, and examples of how to work with the analysed data. It also includes a theory section at the beginning to provide a basis for some of the analyses.

There are different scales of analysis here.

1) Pixels
   a) Measurements done by the microscope
2) Cell
   a) Cells are drawn and pixel values conflated
   b) $\Delta F/F_0$ and Z-Scores are calculated
   c) Regression lines, a selectivity index, and preferred stimulus are calculated
3) Tectal
   a) Distribution of different cell properties are created
4) Fish
   a) Distribution of tectal properties are compared
   b) Distribution of different cell properties can also be created, however due to the relatively large variation between fish, grouping individual fish's cells together first is probably preferable.
5) Experimental Groups
   a) Fish distributions of LPS and Control groups are compared

We're still trying to define exactly what properties we want to look at for each cell. So far we've defined things such as the average Selectivity Index, X-Intercept and Slope of regression lines for Direction, Spatial Frequency, and Brightness respectively. We also look at histograms of preferred stimulus, as well as average ZScore for each stimulus.

Theoretical Overview

Basic Probability

An “experiment” in probability theory is a procedure that has a well-defined set of outcomes. Each of these outcomes have a certain probability of happening, and the probability of all possible outcomes sum to one. A probability of 1 is equivalent to something having a 100% chance of happening.

For example, an experiment could be flipping a coin. The possible outcomes are that it comes up heads or tails, each being assigned a probability of 0.5. The probability of the two outcomes sum to 1 (obviously because the coin has to land on something.)

To extend this idea, an experiment could be flipping a coin 4 times. Now there are many more than two possible outcomes; in fact, there are 16.

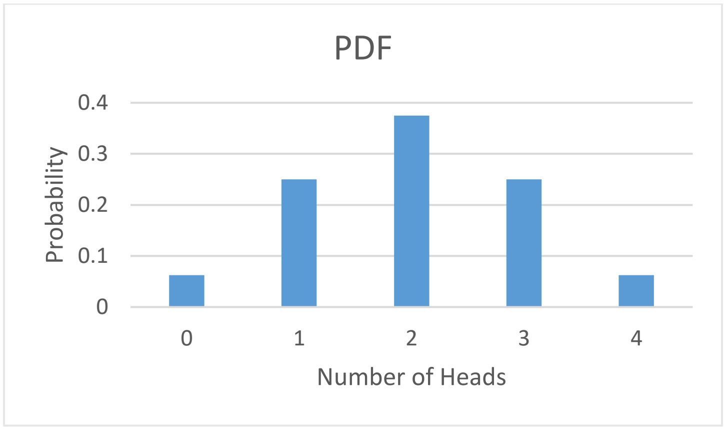
Flip 1	Flip 2	Flip 3	Flip 4
H	H	H	H
H	H	H	T
H	H	T	H
H	H	T	T
H	T	H	H
H	T	H	T
H	T	T	H
H	T	T	T
T	H	H	H
T	H	H	T
T	H	T	H
T	H	T	T
T	T	H	H
T	T	H	T
T	T	T	H
T	T	T	T

Each of these outcomes have the same probability of occurring, hence 1/16. We can also see this another way. Each flip’s outcome as a 0.5 probability of happening (either heads or tails) so each four-flip outcome has a probability 0.5 x 0.5 x 0.5 x 0.5 = 0.0625 = 1/16.

Since every outcome has the same probability of happening, this is called a Uniform Distribution.

Instead of looking at the exact ordering of heads or tails, another experiment could be to flip a coin 4 times and count the number of heads that turn up.

Since we know every possible outcome (and each outcome has the same probability,) calculating the probability of 2 heads occurring is as simple as counting the number of outcomes in the table that contains 2 heads, and dividing by 16. We can see this is equal to 6/16: HHTT, HTHT, HTTH, THTH, TTHH, THHT are all combinations that lead to 2 heads. We can do this 0, 1, 3, and 4 heads appearing. We get the following probabilities (shown in a bar graph)



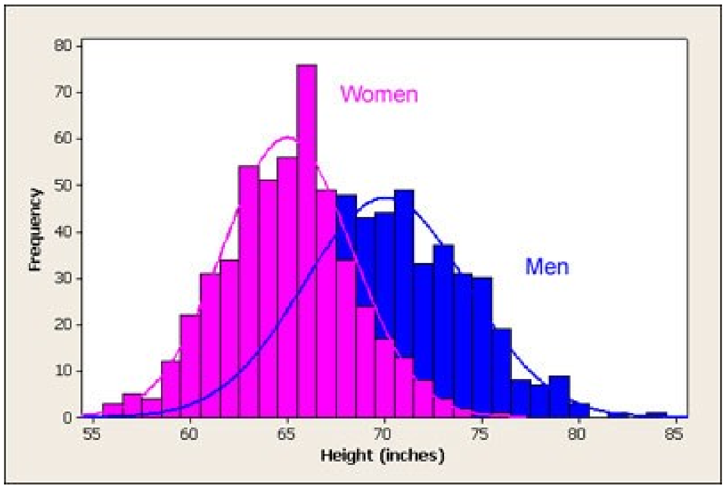
This is called a Binomial Distribution. It has a fancy equation so you don't have to count things:

$$\mathbb{P}(\{x = k\}) = \binom{n}{k}(1-p)^{n-k} \cdot p^k$$

If you were to plug in 0, 1, 2, 3, 4 in for k and set $p = \frac{1}{2}$ and $n = 4$, you would get the values in the bar graph.

Histograms

So far, we've known the exact probability of each event occurring, because we know the probability of heads or tails. But most of the time we don't know how often to expect a certain event to happen. You don't know what the probability is of selecting a woman at random and finding they're 60 inches tall. This is why we collect data. We select a bunch of people, measure their height, and record the data. The goal is to get something like this:



http://www.usablestats.com/lessons/normal

According to the data, someone selected around 600 women. Turns out about 22 of them were 60 inches tall. So in your sample, 22/600 = 3.6% of women were 60 inches tall. Now the important question, is it reasonable to say having selected a woman at random, there is a 3.6% chance that they are 60 inches tall? Well, this depends on how representative of the entire population of women our sample was.

Continuous Distributions

Let's take a brief tangent. On the above chart you'll notice there are bars and a smooth line. It is clear from the bar graph that the measurements were rounded. There is a bar for 60-61 inches, 61-62 inches, etc. But of course people's actual heights aren't a conveniently round number. Indeed there are an infinite number of possible heights between 60 inches and 61 inches, you just need to measure with enough accuracy. This raises two problems.

Firstly, smooth means that for any two heights there is a height in between them, whether it's 61 being between 60 and 62 or 60.00000001 between 60.00000000 and 60.00000002. As such our histograms will only ever be an approximation to an "ideal" distribution, because our accuracy will always be restricted.

Secondly, recall our "Uniform Distribution" from before. If instead of discrete events (like coin tosses), we were randomly pulling a number between 0 and 1 (an interval we purchased from the magical infinite number line store), what does it mean to say the probability of pulling a 0.6? As we said, there are an infinite number of outcomes on a continuous scale, so each number has a probability of $1/\infty$, which, spoiler alert, is equal to 0. This gets into something called Measure Theory, and unless you want to do a degree in math (and who in their right minds would want to do that?) you're not going to learn it. So let's skip over all the nit and grit and jump to the conclusion: when talking about continuous distribution, you need to talk in terms of a range of numbers. Each number may have a probability of 0, but since all numbers have equal weight, the probability of your number being between 0 and 0.5 is ½, the probability of your number being between 0.55 and 0.65 is 1/10, and so on.[1]

In conclusion, continuous distributions don't really help us because we can't actually have the infinite accuracy that they promised us. However, as a mathematical tool for doing statistics, they are important abstractions, which we'll see in in the section after the next.

[1] 0.65-0.55 = 0.1, which is one tenth of our entire range. So there is 1/10 probability that our number will be in this range.

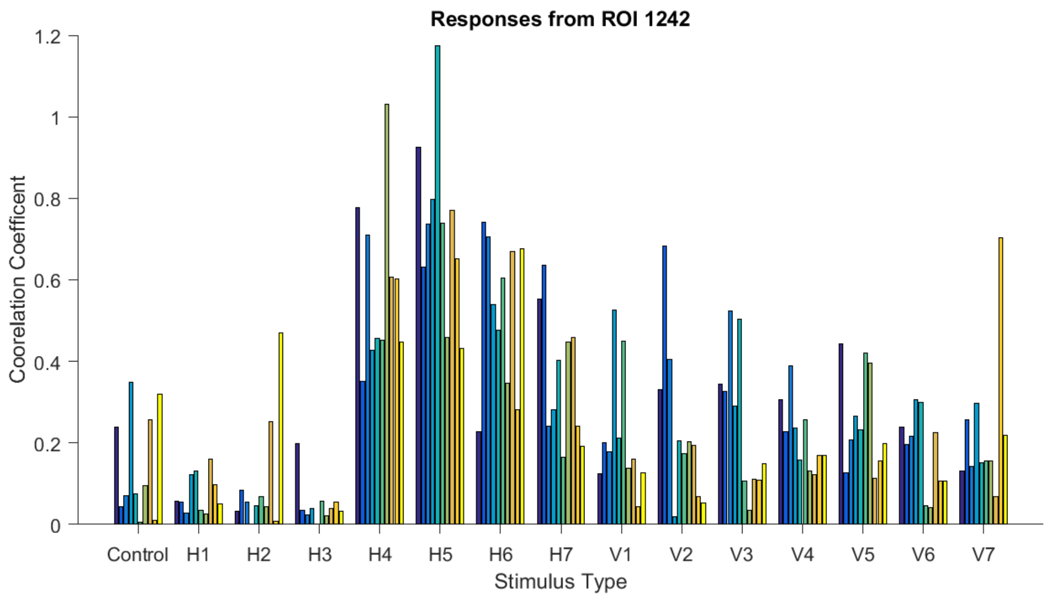
Biology

Sadly we don't care about human heights. If we did, our experiments would be a lot easier! We care about the activity of cells. We go in to this with the following:

Model: when a cell is active, more photons hit the photoreceptors and the pixels that represent the cell's nucleus are brighter

Question: how bright is this cell when we show it an image? (Indirectly, how active is the cell)

Our interpretation is that the brighter the cell, the more active the cell was, the more the cell was responsive to the image. To quote Pologruto, Yasuda, and Svoboda (2004), " interpreting [Genetically encoded Ca2+ indicator (CECI)] fluorescence in terms of neural activity and cytoplasmic-free Ca2+ concentration ([Ca2+]) is complicated by the nonlinear interactions between Ca2+ binding and GECI fluorescence." For now, our definition of cell activity as pixel brightness is probably the best we can do, but it is probably also sufficient for detecting significant differences between two different response types.

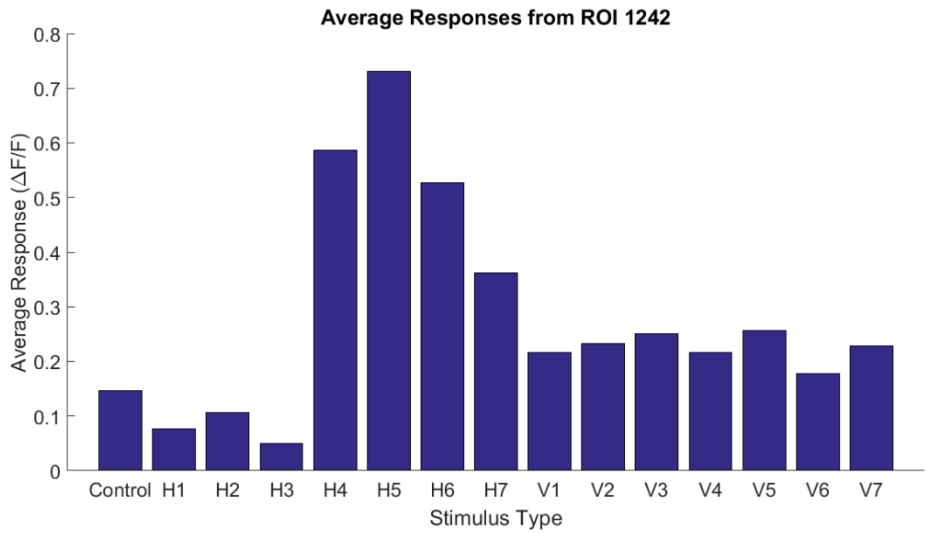


Above is some data. The first thing to acknowledge is that every "Stimulus Type" is a different experiment. We have a sample size of 10 for each stimulus type to try and answer the question "how responsive was this cell to this stimulus." The "Correlation Coefficient" is just the average $\Delta F/F_0$ post stimulus (our measure of how bright the cell was when we show it an image.)

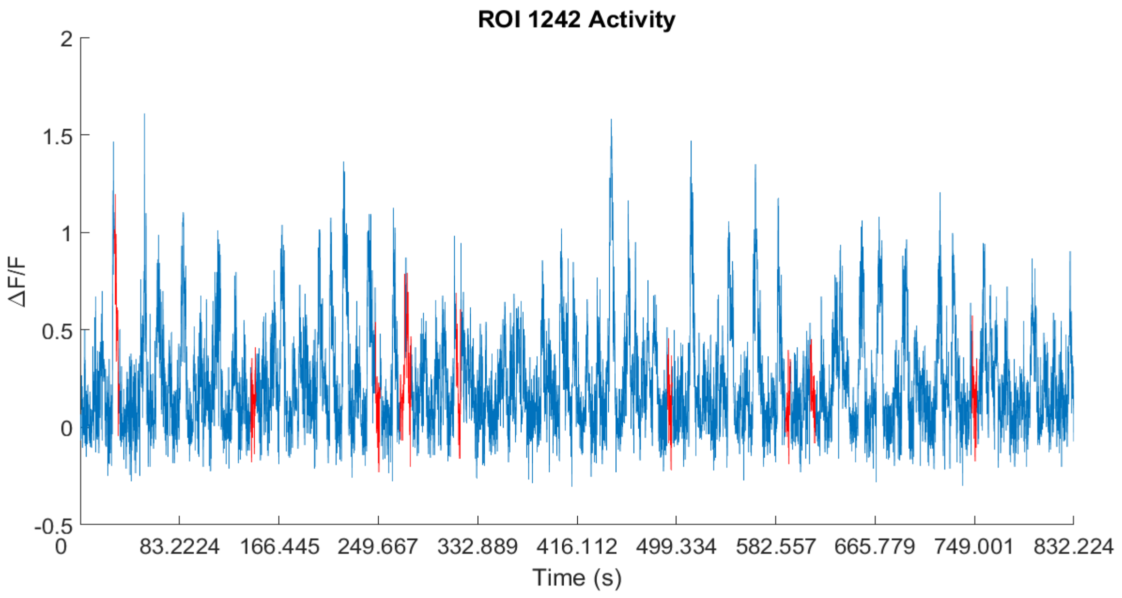
Clearly, the measurements are different, even within a single experiment. Our data is noisy unfortunately. Presumably from two sources.

The actual photodetectors are going to pick up random photons, which is the speckling in our images. This means we have an underlying signal which is altered in a random manner by these random photons.

The number of random photons that hit during our measurement follows its own distribution called a Poisson Distribution. There is some unknown average number of photons that increases the brightness of the cell a certain amount. During one measurement, maybe two random photons hit our photodetector, a second time maybe it was 5. But over all our measurements, these average out to some average number of photons, so by average our measurements together we have our average signal + average added photon brightness.



The second source of noise is from the cell itself. Firstly, there can be activity independent of our stimuli, or even random firing. Secondly, the cells clearly don't respond the same way to the same stimulus. In red are the 5 seconds after the presentations of H4.



But we assume that on average, the cell responds an average way. That is, by averaging enough responses together we get the cell's "average response."

If we assume there is the same amount of noise in each experiment, and we assume there is no signal during our blank stimulus presentation, subtracting the average activity during our control from the average activity during stimulation reveals the actual response of the cell.

But what does this number truly mean? If a cell responds to a control with an average response of 0.15 and to H1 with an average response of 0.1, do we really believe the cell responded 0.05 less than control during visual stimulation? This brings us to confidences, and statistical tests.

Statistical Tests

Given two distributions, are they different? This is the fundamental quest of science! Okay enough drama.

Suppose our cell has an average response to control of 0.15 and an average response to H1 of 0.1. There difference is 0.05, but if we look at the variation in the cell's response, we see that the standard deviation of the cell's responses to the control is 0.13 and for H1 it's 0.08. Intuitively, given the deviation in the cells responses, a different of 0.05 doesn't seem like very much. But we have statistical methods to quantify this intuition.

Behold the Central Limit Theorem. Long story short, as $n \to \infty$, our distribution of cellular responses will approach a normal distribution. Well, more specifically,

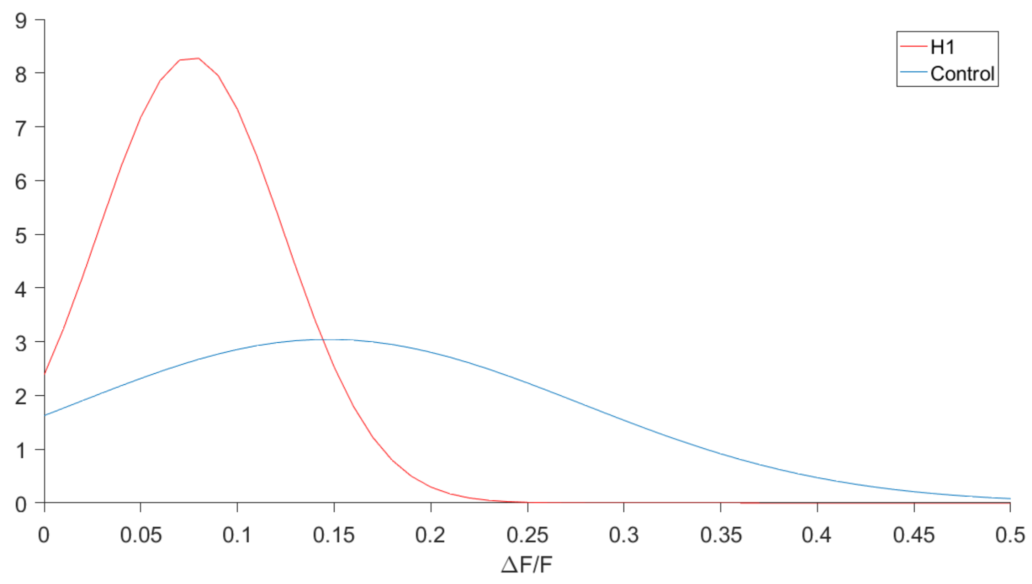
$$\sqrt{n}\left(\left(\frac{1}{n}\sum_{i=1}^{n} X_i\right) - \mu\right) \xrightarrow{d} N\left(0, \sigma^2\right).$$

Let's look back at responses to H1 and Control:

Control:	n = 10	$\bar{x} = 0.1468$	$s = 0.1310$
H1:	n = 10	$\bar{x} = 0.0759$	$s = 0.0481$

s is the sample standard deviation, which is the square root of the sample variance. The sample average is to the mean what the sample variance is to the variance (μ is the mean and σ^2 is the variance).

So according to the CLT, these two samples can be approximated by two different normal distributions: $\mathcal{N}(0.1468, 0.1310)$ and $\mathcal{N}(0.0759, 0.0481)$ respectively. These are what the graphs look like:



So how different are these distributions? Well the centers are 0.0709 away from each other, then to account of the variance of these distributions, we normalize by

$$\sqrt{\frac{\sigma_1^2}{n_1}+\frac{\sigma_2^2}{n_2}}$$

This would all be fine, except (there are a lot of exceptions as we violate more and more assumptions) we don't know σ_1 and σ_2, which are the population errors. We are estimating these with s_1^2 and s_2^2, that is the sample variances. So I wasn't really telling the truth when I said they could both be approximated by normal distributions. They can be approximated, however, by the Student's t-distribution. Our statistic is the same:

$$\frac{\bar{x}_1-\bar{x}_2}{\sqrt{\frac{s_1^2}{n_1}+\frac{s_2^2}{n_2}}}$$

Except for some minor pedantic letter changes. Calculating this for our cellular responses to Control and H, we get:

1.60610036928039

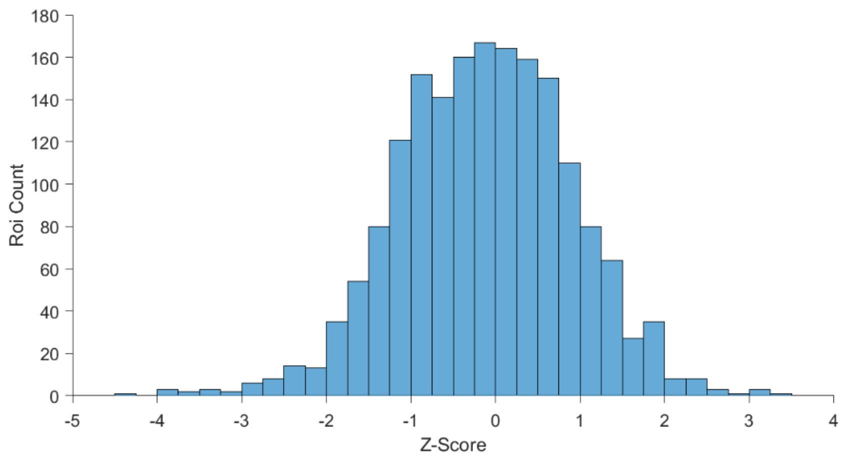
The big difference between Z-Test and T-Test is the table you look at, at this point. Since this is a t-distribution, we look at a t-table. The first thing you'll notice is "Degrees of Freedom." (That's because as n gets larger, our sample variance gets closer to the actual population variance, and so the t-distribution gets closer to a normal distribution). Degrees of freedom is equal to 2*(n-1), so for us 18. We see 1.6 falls somewhere between 0.9 and 0.95. So $p > 0.05$.

All this to say: when you see a Z-Score in the data, we should actually interpret it as a T-Score, and the response is significantly different from the control response (assuming 10 repetitions) if the Z-Score is greater than 1.74 (as per the t-table.)

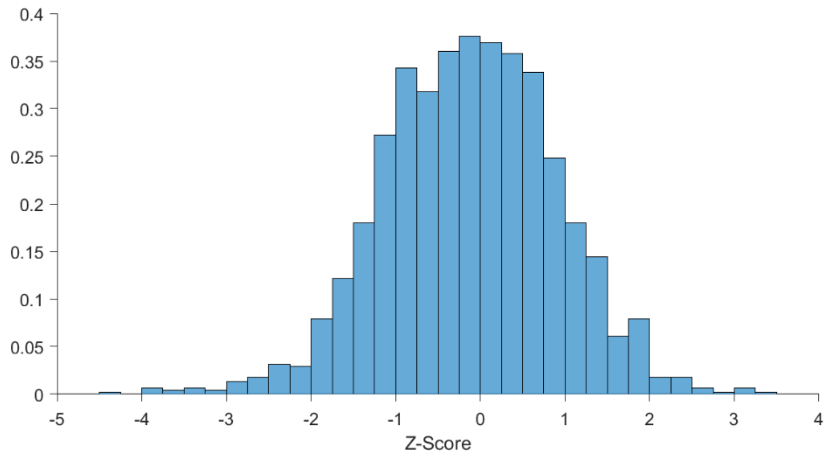
A t-test is only one kind of statistical test. Given two distributions, you can devise a test to determine if the distributions are the same or not. A z-test is used when you have two normally distributed samples. A t-test is used when you have two Student's t distributed samples. There are many more tests, and each one has its own assumptions and usefulness.

Population Distributions

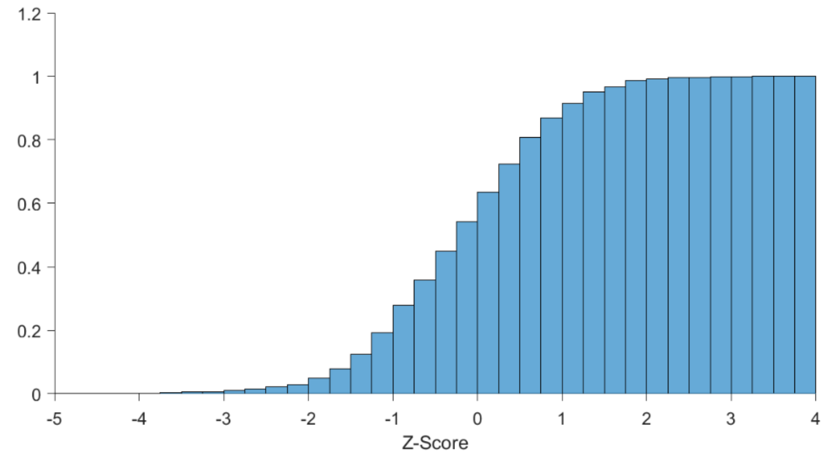
Histograms are an attempt to uncover what the population distribution looks like. When we are collecting data from almost all the cells in the tectum, we can be fairly sure our sample distribution is very close to the population distribution.



This is a histogram showing the Z-Score for H1 in fish *Frodo*. Our previous example, ROI 1242, is adding 1 to the bar representing the range 1.5 – 1.75. The next thing we do is normalize the histogram, so that we can compare easily two different fish.



It looks the same, except the Roi Count is different. This is for Measure Theoretic reasons as explained (or more correctly not explained) in our section on continuous distributions. If you're curious, multiply the y axis numbers by the bin size, 0.25, and then multiply this number by the total Roi Count, and you would get the same numbers.

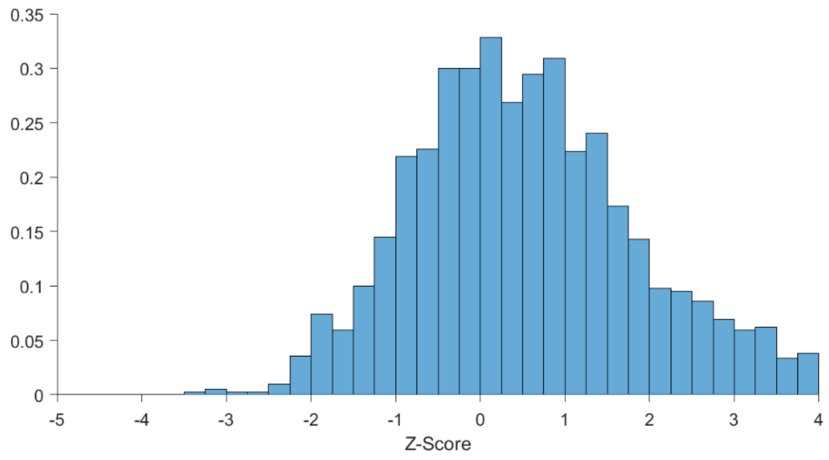


This is simply a different way of looking at the same data. Instead of plotting the Roi Count for each Z-Score, we're plotting the running sum. The first graph is called the PDF (probability density function) because it is graphing a "density" of probability at each spot along the x-axis. The second is called the CDF (cumulative distribution function) because it is the cumulative graph. If you remember calculus, the CDF is the integral of the PDF and the PDF is the derivative of the CDF.

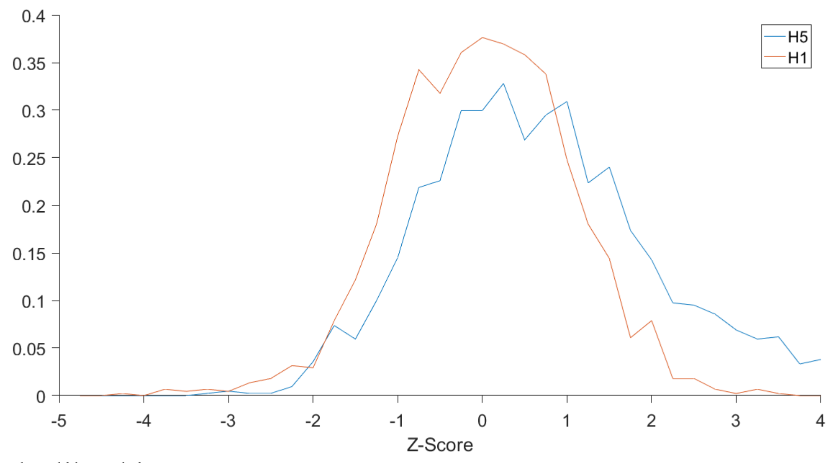
How to interpret these

One things you'll notice about the PDF for H1 is that it is pretty symmetric around zero. Remember that a Z-Score of 0 means than there is no difference whatsoever between the ROIs responses to this stimulus and the control stimulus.

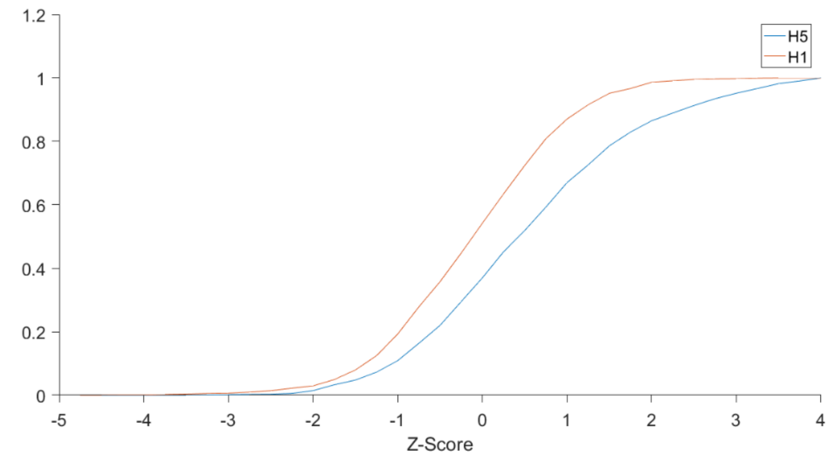
If we look at the same distribution, but for stimulus H5 we see it is skewed to the right:



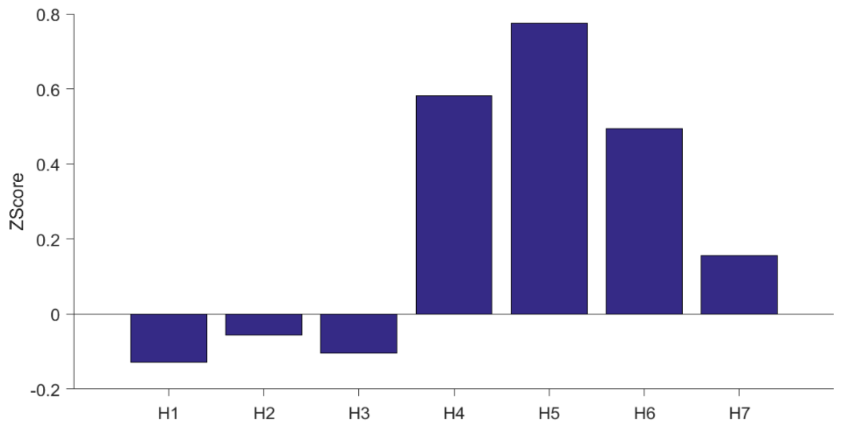
We can see this better if we plot them on top of each other as line graphs:



As a CDF, it looks like this:

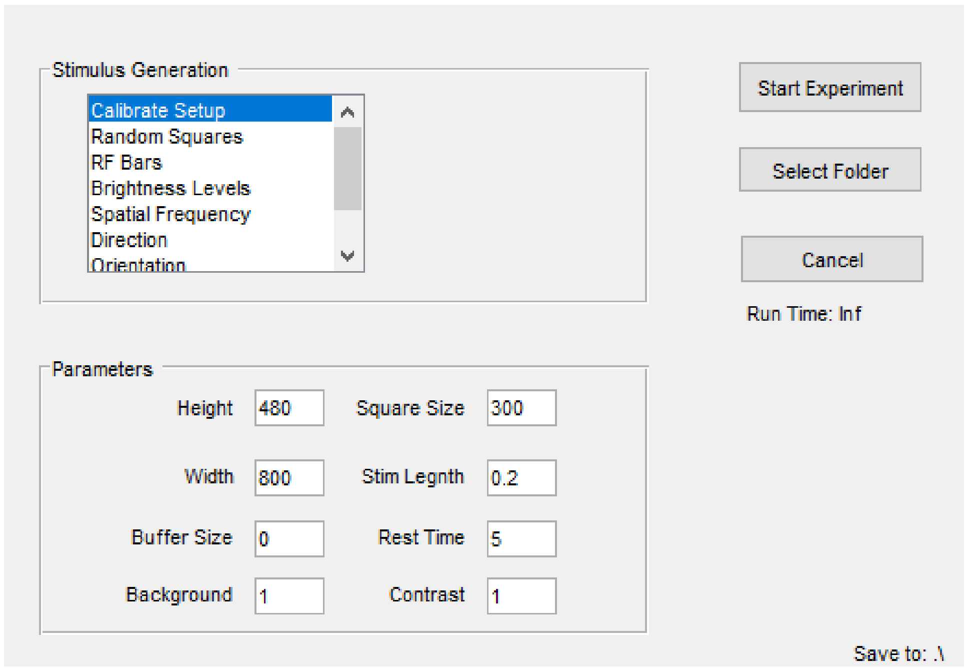


The fish seems to be more responsive to H5 than H1, because many more cells have a higher Z-Score to stimulus H5 when compared to H1. In fact, the entire distribution of H1 seems within ±2 Z-Scores. One may interpret this as noise, and that in fact the fish didn't really respond at all to H1. If we plot the mean Z-Score for each horizontal stimulus (the mean being a measure of the center of the distribution, and being affected by skew) we get this:



My interpretation of this is that the fish could not see the top three horizontal bars.

Pipeline Overview


Stimulus Generation

- Calibrate Setup
- Random Squares
- RF Bars
- Brightness Levels
- Spatial Frequency
- Direction
- Orientation

Start Experiment

Select Folder

Cancel

Run Time: Inf

Parameters

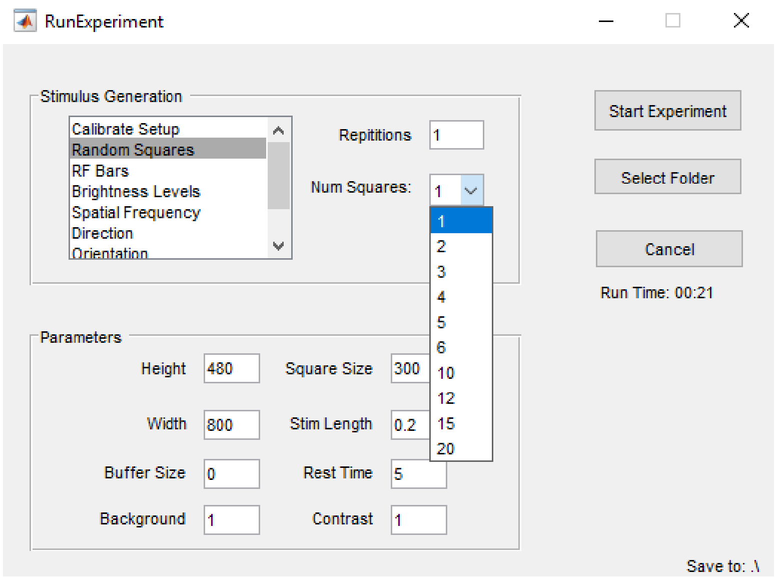
Height	480	Square Size	300
Width	800	Stim Legnth	0.2
Buffer Size	0	Rest Time	5
Background	1	Contrast	1

Save to: .\

This is the GUI interface for the function *RunExperiment.m*.

Stimulus Types Overview

- Calibrate Setup: This simply displays the display area defined by the parameters *Square Size* (see parameter overview). This is useful for making sure your screen is aligned properly.


RunExperiment

Stimulus Generation

- Calibrate Setup
- Random Squares
- RF Bars
- Brightness Levels
- Spatial Frequency
- Direction
- Orientation

Repititions 1

Num Squares: 1

1, 2, 3, 4, 5, 6, 10, 12, 15, 20

Start Experiment

Select Folder

Cancel

Run Time: 00:21

Parameters

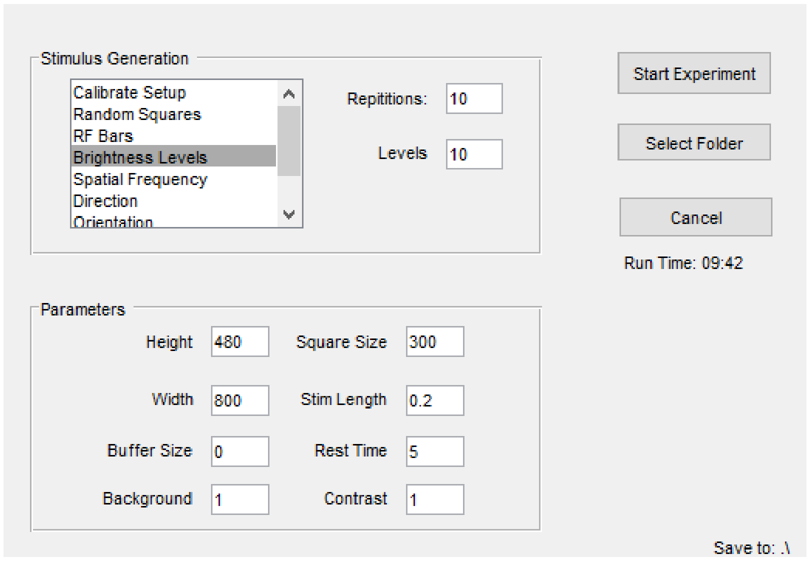
Height	480	Square Size	300
Width	800	Stim Length	0.2
Buffer Size	0	Rest Time	5
Background	1	Contrast	1

Save to: .\

- Random Squares: This stimulus type presents squares in different spatial locations. Used for receptive field mapping. The field *Num Squares* defines the number of squares that are shown, i.e. if 6 is selected, the display field is divided into a 6 x 6 grid. For each stimulus presentation, one of the grids is activated. The list for *Num Squares* is therefore made up of the factors of the parameter *Square Size* (see parameter overview). i.e. 1, 2, 3, 4, 5, 6, 10 , 12, 15, 20 are all divisors of 300. The control stimulus is a blank screen.

- RF Bars: This stands for receptive field bars. It is similar to *Random Squares*, except instead of squares it displays bars that are horizontal and vertical in orientation. This maps the azimuthal and elevation separately.

- Brightness Levels: This displays different levels of brightness, the number of levels being defined by the parameter *Levels*. In this example, 10 levels will be displayed, that is images with intensity levels {0.1, 0.2, …, 1}. The control stimulus is a blank screen, i.e. an intensity level of 0.

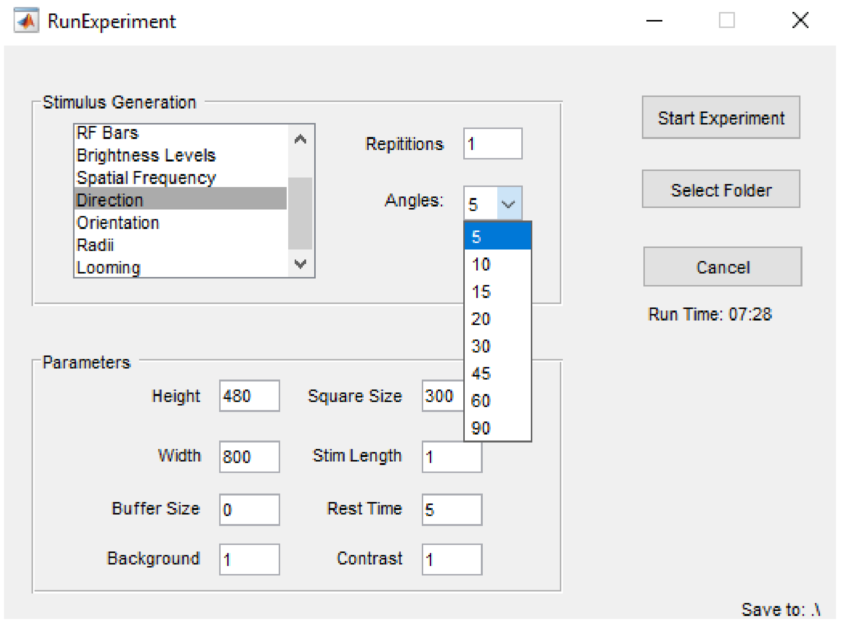


Repititions: 10
Levels 10
Run Time: 09:42
Height 480
Square Size 300
Width 800
Stim Length 0.2
Buffer Size 0
Rest Time 5
Background 1
Contrast 1
Save to: .\

- Spatial Frequency: This displays sinusoidal gratings with different frequencies. These frequencies, similar to *Num Squares* are all of the factors of the width of the screen. For example with a width of 800 pixels, there will be 16 frequencies; sinusoids with periods of {800,400,200,160,100,80,50,40,32,25,20,16,10,8,5,4}. Note that periods of 1 and 2 are not shown due to aliasing.[2] The sinusoid then counterphases with a frequency of 5 HZ (hardcoded.) The control stimulus is a static presentation of the previous spatial frequency to maintain average light levels coming from the screen.[3] For more information, see *Spatial Frequency Stimulus Information.txt.*

[2] Analysis indicates we shouldn't be showing a period of 4 either, but for now this frequency is still included
[3] Perhaps this should be changed to a uniformly grey screen? Depends on what exactly you want to control for.

- Direction: This shows sinusoidal gratings moving in the defined directions. The parameter *Angles* denotes the interval between angles. E.g. if angles = 5, that means grating will be shown moving at 5 degrees, 10 degree, 15… etc. We tend to use angles = 30. This gives us 12 direction. For the cardinal directions, choose angles = 90. The control stimulus is a stationary grating or a random orientation.



RunExperiment

Stimulus Generation: RF Bars, Brightness Levels, Spatial Frequency, Direction (selected), Orientation, Radii, Looming

Repititions: 1

Angles: 5 (options: 5, 10, 15, 20, 30, 45, 60, 90)

Parameters

Height: 480

Square Size: 300

Width: 800

Stim Length: 1

Buffer Size: 0

Rest Time: 5

Background: 1

Contrast: 1

Run Time: 07:28

Save to: .\

- Orientation: Shows a bar intersecting the center of the display area at defined angles similar to *Direction*. Currently, the only way to change the width of the bar is by changing the code directly.

- Radii: This shows circles of different radii centered on the display area. Radii are linearly spaced from 0 to half the height of the screen, with the number of radii defined by the parameter *Levels* (similar to *Brightness*). The control stimulus is a blank screen (a radius of zero).

- Looming: Created by Michael Lynn, this presents a looming stimulus, with the speed of looing defined by *radius/velocity* and the starting size of the circle defined by *Min obj size.* The control is a blank screen.

Parameter Overview

- Height/Width: This is the height and width of the screen on which the stimulus will be presented. In this example a 480x800px LED screen is being used. This is important for determining the size of the stimulus being presented, especially for stimuli like Spatial Frequency and Radii.

- Square Size: This is the size of the part of the screen where you want to display your stimuli. This is important for stimuli such as Random Squares and RF Bars. Currently, the display area can only be a square, and is centered in the middle of the screen. You can adjust the height of the area with the field Buffer Size.

- Buffer Size: This is the number of pixels from the bottom of the screen to the bottom of your desired display area. For example, if your screen has a height of 480px, a display area of 300x300px and you want the display area to be perfectly centered on the screen, you would need to set the Buffer Size to 90.

- Stim Length: This is the length of time each stimulus will be displayed for in seconds. For the brightness level example the stimulus will be presented for 200 ms. For the direction stimulus, it will be presented for 1 s.

- Rest Time: This is the length of time between stimuli, during which the "Blank" or "Control" stimulus will be presented. In each example, each stimulus will be followed by a 5 second rest.

- Background: This is the brightness of the background. 1 = white, 0 = black. In between are different shades of grey.

- Contrast: This defines the contrast between the background (see *Background*) and the stimuli. If contrast is 1, and the background is 1, the background will be white and the stimuli will be black. Alternatively if the background is 0, the background will be black and the stimuli will be white. We find black stimuli on a white background elicit the strongest responses. If the background is less than 0.5, the stimulus will be *Contrast* brighter than the background. Otherwise the stimulus will be *Contrast* darker than the background. E.g. if the background is 0.5 and the contrast is 0.5, the stimuli will be white on a grey background. If the background is 0.49 and the contrast is 0.49, the stimuli will be black on a grey background.

Execution Overview

The parameter *Repetitions* defines the number of repetitions of each stimulus (I draw the distinction between different *stimuli*, e.g. circles of different radii, and different *stimulus types,* which are radii, random squares, spatial frequency, etc.)

On the computer connected to the microscope, open ThorSync and ThorImage. Make sure that ThorImage has trigger input enabled and is connected to ThorSync. You will also have connect ThorSync to ThorImage. There are boxes for this in each program respectively. Also ensure raw data capture is enabled. Under the cancel button on the stimulus generation GUI, there is the run time of the stimuli. Adjust the number of frames collected accordingly.

Once all the parameters are appropriately chosen and the stimuli are generated, a random permutation of the stimuli is generated.

Next a five volt, one second trigger is sent to ThorSync through the ao1 port of the National Instruments device USB 6009, initiating image capture.

The permutation of the stimuli is then presented. By the end of the experiment each stimulus, including a control stimulus, will have been presented *n* times, where *n* is the number of repetitions.

After all the stimuli are presented, two text files are created. *StimulusData.txt* and *StimulusConfig.txt*. The former contains the post-trigger time in seconds of each stimulus as well as an identifying stimulus number. The latter contains the parameters used in stimulus presentation.

Notes and Troubleshooting

Sometimes the NI device is not found, or the trigger can't be sent. We're still not sure why this happens sometime, but sometimes restarting MATLAB with the device already plugged in solves the problem. Otherwise, try restarting your computer.

If you click on the RunExperiment GUI while the stimuli are being presented, the stimuli may start being presented on the GUI. So don't click on the GUI while the experiment is running.

If your selected folder doesn't have write permission enabled, your stimulus data won't save. The data will still be printed on the screen, so you can copy and paste it into a text file.

You can edit *RunExperiment.fig* with MATLAB's GUIDE (GUI Development Environment,) which is helpful to change the default values for the stimulus parameters. You can also defined them in the code *RunExperiment.m.* Otherwise, update the stimulus parameters each time you launch RunExperiment.

You should save the stimulus files in a folder with the same name as the one in which you save the image data. This will allow the automatic merging of stimulus data and image data into a single folder later.

Data Analysis

You should have the following files in a single folder:

- ✓ *Episode001.h5*
- ✓ *Experiment.xml*
- ✓ *Image_0001_0001.raw*
- ✓ *StimulusConfig.txt*
- ✓ *StimulusTimes.txt*
- ✓ *ThorRealTimeDataSettings.xml*

Otherwise, the data analysis function will yell at you and refuse to do anything. The only exception is if you folder contains the word "darkness" in its name. In this case, the function will assume you were collecting data without any stimulus data, and so only requires *Image_0001_0001.raw* and *Experiment.xml*.

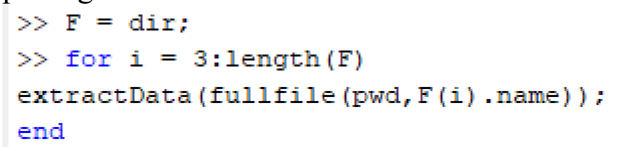
Run extractData(**Folder**), where **Folder** is a string containing the path to the folder in which all your data for the experiment is stored. This will generate two files:
"(**Folder**).mat"
"Analysed (**Folder**).mat"
The first contains the raw pixel brightnesses for each identified ROI, the second contains the analysed data, i.e. computes $\Delta F/F_0$ (Jia, H. et al, 2011) and calculates responses to each stimulus as average $\Delta F/F_0$ as well as Z-Score as compared to the ROI's response to the blank stimulus.

After running extractData, and you have all your experiments of the same stimulus type compiled in a folder, you can run the function *sampleData.m*. This will create a file *Sample.mat* in the folder which contains a sample for *n* cells from each fish, where *n* is the number of cells in the fish with the fewest number of cells. This ensures no one fish is weighted more than others in a pooled analysis.

Notes and Troubleshooting

"(**Folder**).mat" contains the structures **header** and **ImageData**. Extracting the data from the large image file takes up the majority of the analysis time. If you wish to change some things in the analysis part of the function, you can also run extractData(**header**,**ImageData**) and only the analysis part of the function will run.

The way the hard drive is set up, there is a folder for each stimulus type as well as a folder for new data. I recommend putting all data into the new data folder and running this script:



```
>> F = dir;
>> for i = 3:length(F)
extractData(fullfile(pwd,F(i).name));
end
```

pwd outputs your current folder. Navigate to the new data folder in MATLAB, run this script, and the function will run on every subfolder. Then you can move the folders, containing the analysed data, into the appropriate stimulus-type folder.

Analysed Data File Overview

Description of the variables contained in the data files generated by functions extractData and sampleData.

Analysed (**Folder**) .mat

header.

FileName	The name you gave the experiment
RoiCount	Number of ROI
TimeLapse	Total experiment run time
FPS	Capture speed
Frames	Number of frames recorded
Slices	Number of slices recorded
ImageWidth	Image width (pixels)
ImageHeight	Image height (pixels)
fieldSize	Length of image field in microns
zScale	Distance between Z Slices in microns
zStart	Depth of first Z Slice
FlyBackFrames	Number of fly-back frames
RoiMask	1 x (Slices) cell array. Each cell contains a p x 2 cell array where p is the number of ROIs in that slice. The cells in the first column contains the x coordinates and the second column contains the y coordinates for each ROI

AnalysedData.

Times	Time of each frame for each ROI. N x T matrix
dFF0	$\Delta F/F_0$ each ROI at each frame. N x T matrix
RoiCoords	Coordinates of each ROI. 3 x N matrix
Responses	RoiData(n).XCor data (see below) for each ROI, averaged over each repetition of the stimulus. N x (StimuliCount) matrix
ZScore	ZScore for each ROI and each Stimulus. N x (StimuliCount – 1) matrix.

RoiData(n).

Brightness	Brightness profile of Roi n
Coordinates	Coordinates of Roi n
XCor	Stimulus Count by Repetition Count matrix containing the average $\Delta F/F_0$ for the 2 seconds following each stimulus presentation.

StimulusData.			
	Raw		3 x n matrix. First column is simply the count 1 through n. The second column is the time of that stimulus. The third column is a description of the stimulus, which is different for each stimulus type.
	Times		A copy of the second column of the raw data
	Configuration.		
		StimuluiCount	Number of different stimuli presented
		Repetitions	Number of repetitions
		Type	Stimulus type. Use stimType.m to decode
		DisplayLength	Length of stimulus
		RestLength	Pause between stimuli
		PlusMinus	Amplitude of up-down from grey background
		Number	Number of stimuli per repetition
		Height	Height of display window
		Width	Width of display window
		BottomPad	Verticle offset of display area
		Area	Size of display area
		Background	Shade of background between 0 and 1

Sampled.mat

RoiMin	Number of ROIs in fish with fewest ROIS
Responses	RoiMin x StimuliCount x FishCount matrix of $\Delta F/F_0$ data
SI	RoiMin x FishCount matrix of selectivity index $\left(1 - \frac{\min \Delta F/F_0}{\max \Delta F/F_0}\right)$
XCor	RoiMin x StimuliCount x Repitition x FishCount matrix of $\Delta F/F_0$ data
Zscore	RoiMin x StimuliCount-1 x FishCount matrix of Z-Score Data

Working with the Data (with Practical Examples)

Exporting Data to Excel

Most important is to be able to do something that works. So I start this section with how to export data to Excel. If ever you get stuck, you should be able to export the data and try to solve the problem in Excel, or any program in which you feel most comfortable.

runLinearModel(pwd) will run a linear regression for each cell. This will take a while…
For brightness, it solves the linear regression for the brightness levels 3 to 9. For spatial frequency, it solves the linear regression for spatial frequency stimuli 1 to 9.

The x - intercept, slope, and R-squared value are then saved under "LM" in the Sampled.mat file

```
>> whos
  Name                 Size                   Bytes  Class     Attributes

  LM                   1x1                   478896  struct
  Responses         1661x17x12              2710752  double
  RoiMin               1x1                        8  double
  SI                1661x12                  159456  double
  ZScore            1661x16x12              2551296  double

>> LM
LM =
    XIntercept: [1661x12 double]
         Slope: [1661x12 double]
      RSquared: [1661x12 double]
```

Suppose you want to save them all in an excel file, with different sheets for every fish.

```
>> for i = 1:size(ZScore,3)
Data = [LM.XIntercept(:,i) LM.Slope(:,i) LM.RSquared(:,i)];
xlswrite(['Spatial.LPS.LinearRegression.xlsx'],Data,['Fish ' int2str(i)]);
end
```

In xlswrite, the first parameter is the file name. The last parameter is the sheet name. The second parameter is the data you want to save (must be 2D).

This saves three columns of data on 12 different pages. The first, second, and third column are the XIntercept, Slope, RSquared respectively. If you wanted to put all the same properties on the same page, with different pages for each property, it is very similar.

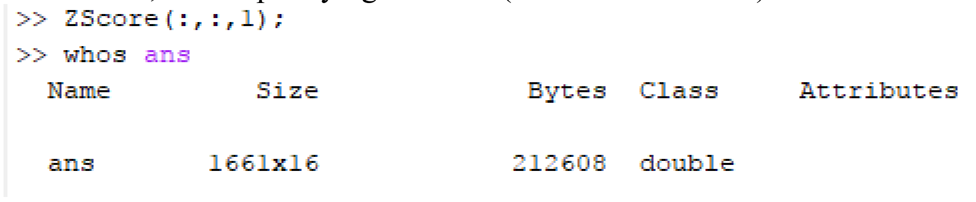
```
>> xlswrite(['Spatial.LPS.LinearRegression.xlsx'],LM.XIntercept,'XIntercept');
>> xlswrite(['Spatial.LPS.LinearRegression.xlsx'],LM.Slope,'Slope');
>> xlswrite(['Spatial.LPS.LinearRegression.xlsx'],LM.RSquared,'RSquared');
```

Working with the Data (with Practical Examples)

So far, we've created a ~2MB file with 15 sheets. 12 for each fish and its respective ROI's parameters, and another 3 with each parameter populated with the ROIs of each fish.

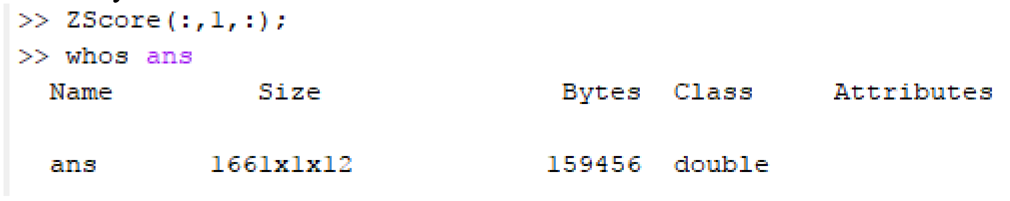
Using xlswrite should give you the power to handle the data in whatever way you wish, be it MATLAB, Excel, or PRISM. Just a few last reminders:

1. The first dimension is the vertical dimension, and the second is the horizontal.
2. You can only write a 2D variable
3. You must move any dimension of size 1 to the last dimension using permute
   Example: ZScore. ZScore is 3D, which means you have to specify at least one dimension before saving. For example: ZScore(:,:,1) will take all elements along the first two dimensions, while specifying the third (the fish dimension.)



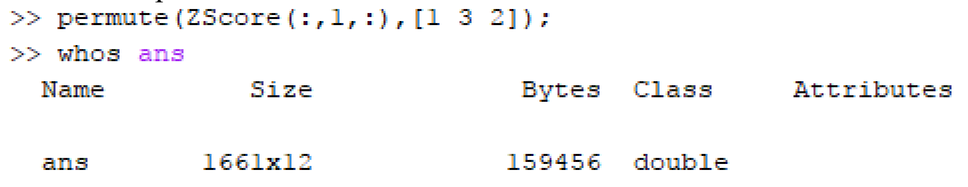
```
>> ZScore(:,:,1);
>> whos ans
  Name          Size                 Bytes  Class     Attributes

  ans        1661x16                212608  double
```

   As you can see, the output is two dimensional. On the other hand, ZScore(:,1,:) WON'T work as you can see:



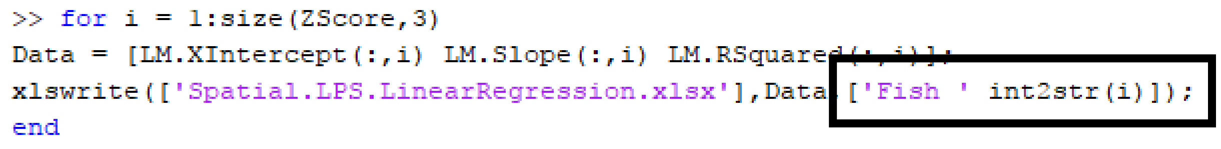
```
>> ZScore(:,1,:);
>> whos ans
  Name          Size                   Bytes  Class     Attributes

  ans        1661x1x12                159456  double
```

   Even though technically there are two dimensions, it's still a 3D variable in MATLAB, so we must permute it first:



```
>> permute(ZScore(:,1,:),[1 3 2]);
>> whos ans
  Name          Size                 Bytes  Class     Attributes

  ans        1661x12                159456  double
```

   As you can see, the second and third dimension were flipped, and any trailing size 1 dimensions are ignored, so the variable is now actually 2D.
4. In a for loop, you can use the index in your sheet and file names, but you have to use this syntax:



```
>> for i = 1:size(ZScore,3)
Data = [LM.XIntercept(:,i) LM.Slope(:,i) LM.RSquared(:,i)];
xlswrite(['Spatial.LPS.LinearRegression.xlsx'],Data,['Fish ' int2str(i)]);
end
```

Working with the Data (with Practical Examples)

Making Graphs

MATLAB does have its advantages. Most notably, computations that will take much clicking and dragging and scrolling can be executed in one line in MATLAB. Here I'll give an overview of how to make some graphs.

Your folders should hopefully look something like this.

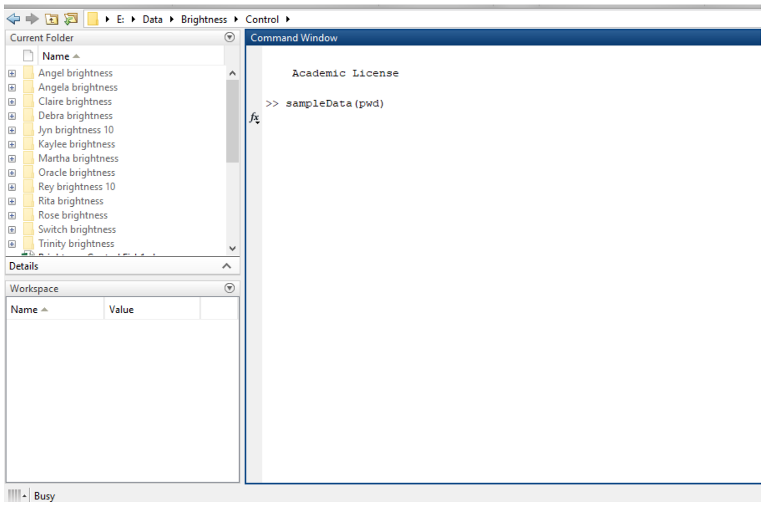
- Data
  - Stimulus Type 1
    - LPS
      - Fish 1
      - Fish 2
      - Etc.
    - Control
      - Fish 1
      - Fish 2
      - Etc.
  - Stimulus Type 2
    - LPS

      ...

    …

Now one thing to remember is there is no right way to do anything. All that matters is that it works. sampleData also saves its variables as an excel file for each fish. However, in this section, I'll cover how to do things in MATLAB.

You can see in the current folder panel, we have 13 Control fish. As explained before, the first thing sampleData(pwd) will do is loop through every fish in the current folder and find out what the smallest ROI count is. Then it samples randomly that many ROIs from each fish and compiles them in one location.

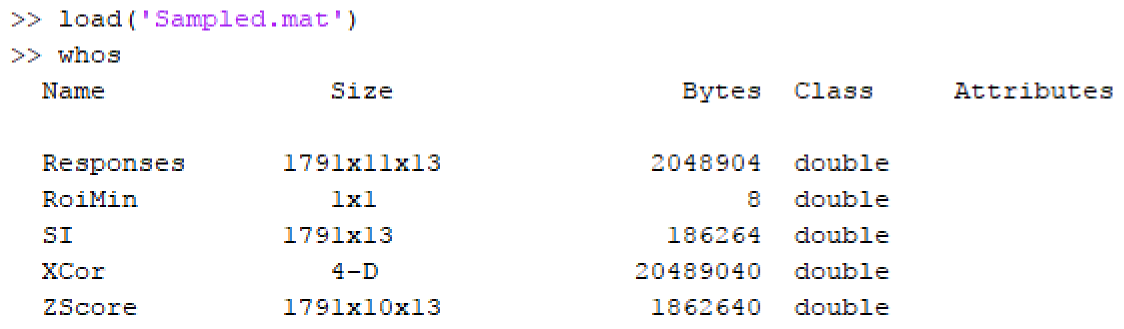


```
Angel brightness
Angela brightness
Claire brightness
Debra brightness
Jyn brightness 10
Kaylee brightness
Martha brightness
Oracle brightness
Rey brightness 10
Rita brightness
Rose brightness
Switch brightness
Trinity brightness
```

```
    Academic License

>> sampleData(pwd)
```

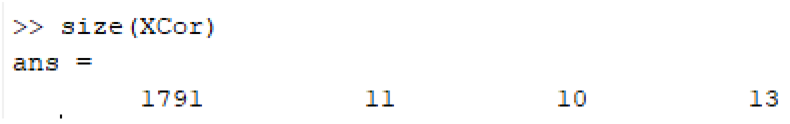
Working with the Data (with Practical Examples)

After this is done running, these are the variables that we have in the Sampled.mat file:



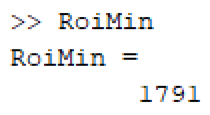
```
>> load('Sampled.mat')
>> whos
  Name                  Size                    Bytes  Class     Attributes

  Responses          1791x11x13               2048904  double
  RoiMin                1x1                         8  double
  SI                 1791x13                   186264  double
  XCor                  4-D                  20489040  double
  ZScore             1791x10x13               1862640  double
```

(MATLAB doesn't show the size of XCor, so here it is explicitly)



```
>> size(XCor)
ans =
        1791          11          10          13
```

As you can see, the length of the last dimension in each variable is the number of fish we have. (Think of the 3D variables as a cube of numbers. Each level is a plane of numbers associated with a single fish. Think of the 4D variable as…I guess 13 cubes. 4D is confusing)



```
>> RoiMin
RoiMin =
        1791
```

You can see the length of the first dimension in SI, Responses, and ZScore is the number of ROIs. So, for example, SI is a matrix of numbers containing the selectivity index for each ROI in each fish like so:

	Fish 1	Fish 2	…
ROI 1	0.546597353034439	0.401421952472869	
ROI 2	0.47712133124981	0.34727599450338	
…			

(Note: ROI 1 doesn't correspond to ROI 1 in Fish 1, it is just the first randomly selected ROI in each fish.)

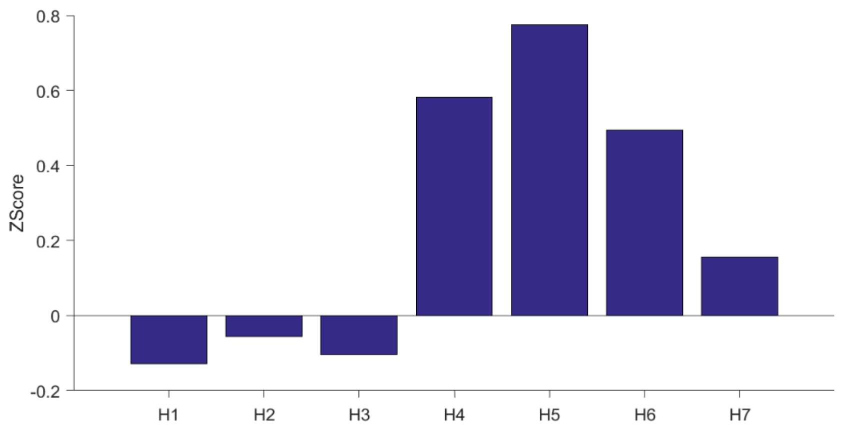
But we're looking at brightness data, so the more interesting variables here are ZScore and Responses, both of which are three dimensional. Let's ignore the third dimension, and just look at the first two (i.e. looking at a single slice of the data cube)

Each slice of Responses is a 1791 x 11 matrix of numbers. The second dimension represents the 10 brightness levels + 1 control stimulus. The ZScore compares the brightness responses to the control responses, so it doesn't have a ZScore for the control response; hence, its second dimension represents only the 10 brightness levels.

Now that we understand the data a bit better, let's do something with it.

```
>> whos ZScore
  Name               Size                    Bytes  Class     Attributes

  ZScore        1791x10x13                 1862640  double

>> Fish_Mean = mean(ZScore,1);
>> whos Fish_Mean
  Name               Size                 Bytes  Class     Attributes

  Fish_Mean        1x10x13                 1040  double

>> Fish_Mean = permute(Fish_Mean,[2 3 1]);
>> whos Fish_Mean
  Name                Size              Bytes  Class     Attributes

  Fish_Mean        10x13                 1040  double
```

Here we've take the mean of the variable ZScore along the first dimension, that is, along the ROI dimension. In the theory section we did this for a single fish and got this:



We've just gone and done this for 13 fish simultaneously.

We then permuted the variable, that is, we swapped the dimensions around. Now dimension 1 is the stimulus dimension, dimension 2 is the fish dimension, and the collapsed dimension of ROIs is pushed to the back so it can be ignored.

```
>> Average_ZScore = mean(Fish_Mean,2);
>> whos Average_ZScore
  Name                     Size               Bytes  Class     Attributes

  Average_ZScore         10x1                    80  double

>> Variation_ZScore = std(Fish_Mean,[],2);
>> whos Variation_ZScore
  Name                     Size               Bytes  Class     Attributes

  Variation_ZScore       10x1                    80  double
```

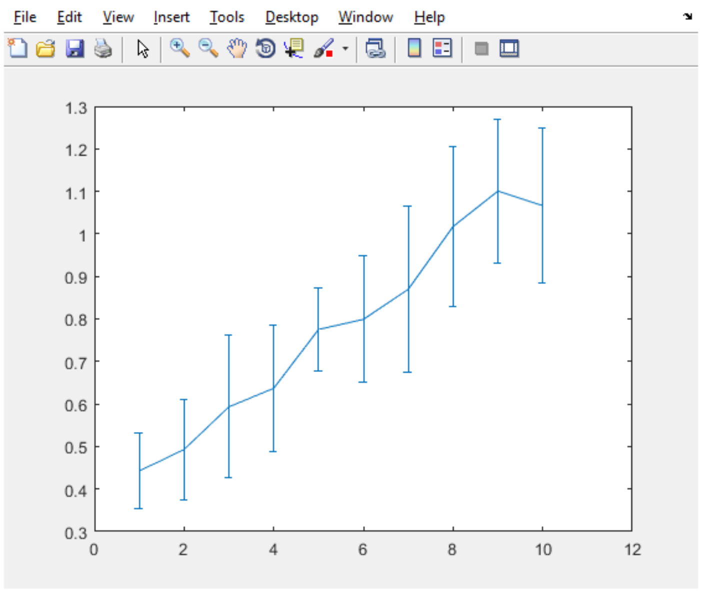
Now we've calculated the mean and standard deviation along the second dimension (the fish dimension.) The syntax for std and mean are slightly different. As you can see, there is a [] that needs to be inserted for the std function.

Now to get standard error:

```
>> StdError_ZScore = Variation_ZScore/sqrt(size(ZScore,3));
>> whos StdError_ZScore
  Name                     Size               Bytes  Class     Attributes

  StdError_ZScore        10x1                    80  double
```

Here we take the size of ZScore's third dimension (the fish dimension) which is 13, take the square root (sqrt) and divide the variation by this. We can now plot out the data with errorbars:

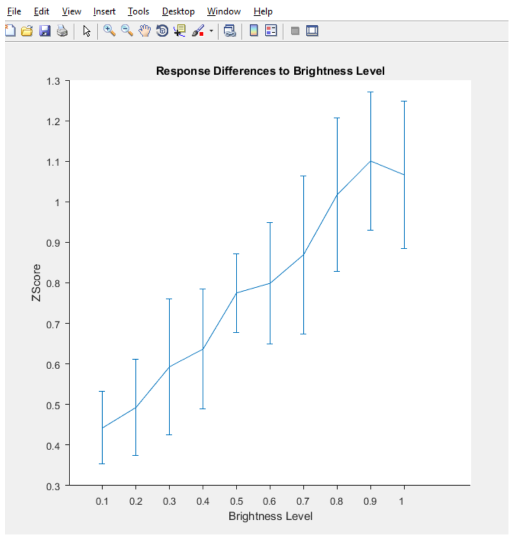
```
>> errorbar(Average_ZScore,StdError_ZScore)
```



```
>> xlabel('Brightness Level')
>> set(gca,'XTick',[1:10])
>> set(gca,'XTickLabel',[0.1:0.1:1])
>> ylabel('ZScore')
>> box off
>> set(gca,'TickDir','out')
>> title('Response Differences to Brightness Level')
```

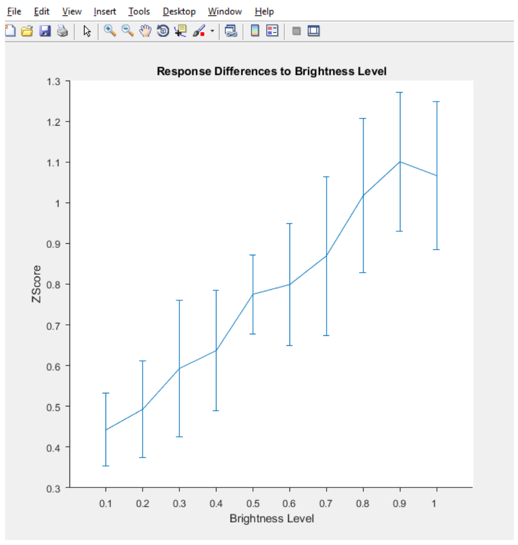
This formats the graph nicely. I set the x-axis ticks to the 1,2,3,…,10 and then labeled them 0.1, 0.2, …, 1.0. All the other lines are fairly self-explanatory. “gca” refers to the current axes.

Now our graph looks like this:

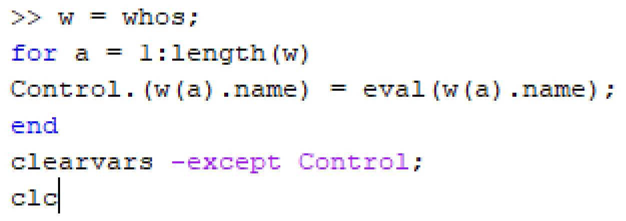


If you want to center the data in the graph, you can change the x limits:

```
>> xlim([0 11])
```

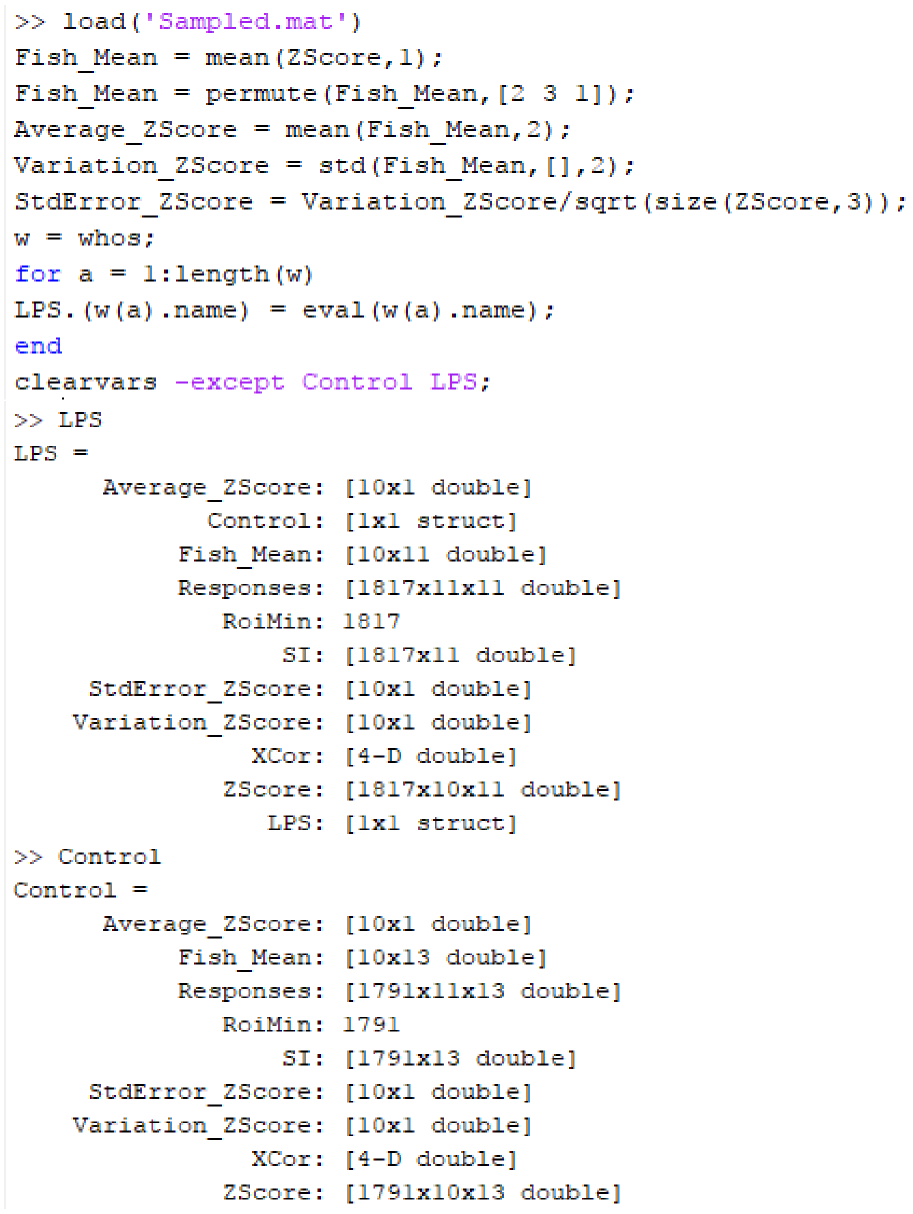


We can then repeat this entire procedure with the LPS Folder, but first I like to do this:



```
>> w = whos;
for a = 1:length(w)
Control.(w(a).name) = eval(w(a).name);
end
clearvars -except Control;
clc
```

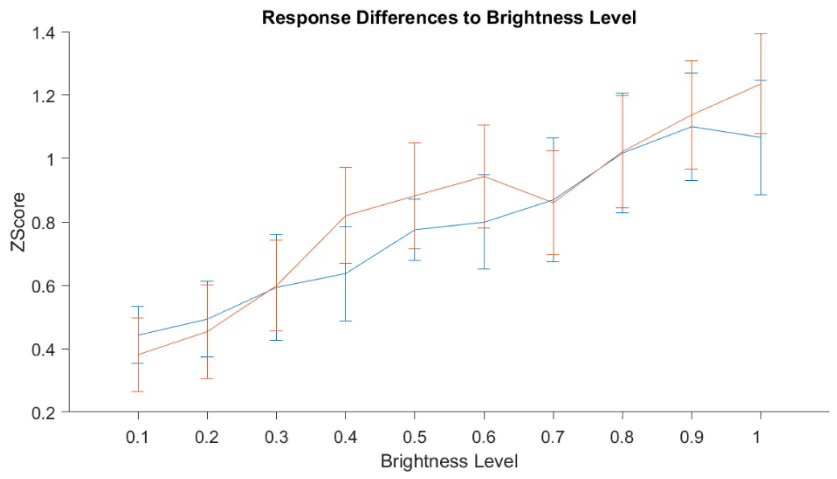
This saves all the current variables into a structure called Control. And clears all variable except this new Control structure. This way the variables won't be overwritten by the LPS variables, and we can work with both of them at the same time. Now we can do the same for the LPS Folder.



```
>> load('Sampled.mat')
Fish_Mean = mean(ZScore,1);
Fish_Mean = permute(Fish_Mean,[2 3 1]);
Average_ZScore = mean(Fish_Mean,2);
Variation_ZScore = std(Fish_Mean,[],2);
StdError_ZScore = Variation_ZScore/sqrt(size(ZScore,3));
w = whos;
for a = 1:length(w)
LPS.(w(a).name) = eval(w(a).name);
end
clearvars -except Control LPS;
>> LPS
LPS =
      Average_ZScore: [10x1 double]
             Control: [1x1 struct]
           Fish_Mean: [10x11 double]
           Responses: [1817x11x11 double]
              RoiMin: 1817
                  SI: [1817x11 double]
     StdError_ZScore: [10x1 double]
    Variation_ZScore: [10x1 double]
                XCor: [4-D double]
              ZScore: [1817x10x11 double]
                 LPS: [1x1 struct]
>> Control
Control =
      Average_ZScore: [10x1 double]
           Fish_Mean: [10x13 double]
           Responses: [1791x11x13 double]
              RoiMin: 1791
                  SI: [1791x13 double]
     StdError_ZScore: [10x1 double]
    Variation_ZScore: [10x1 double]
                XCor: [4-D double]
              ZScore: [1791x10x13 double]
```

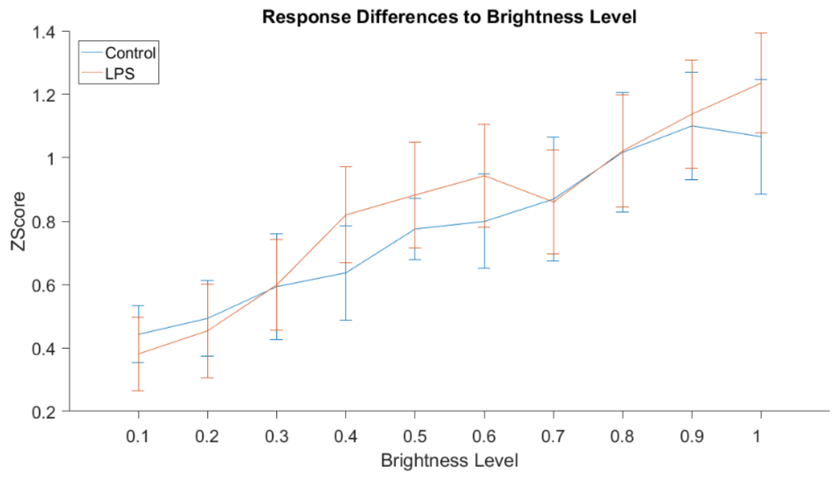
Now we can plot both on the same graph. If you want to save the figure as a picture, make sure to change the font size on the graph like in the last line here. "Hold on" ensures the second plot doesn't overwrite the first plot, allowing both to appear at once.

```
errorbar(Control.Average_ZScore,Control.StdError_ZScore)
hold on;
errorbar(LPS.Average_ZScore,LPS.StdError_ZScore)
xlabel('Brightness Level')
set(gca,'XTick',[1:10])
set(gca,'XTickLabel',[0.1:0.1:1])
ylabel('ZScore')
box off
set(gca,'TickDir','out')
title('Response Differences to Brightness Level')
xlim([0 11])
set(gca,'FontSize',18)
```



Finally, a legend:

```
>> legend('Control','LPS')
>> set(legend,'Location','northwest')
```



The exact same code can be used for the spatial frequency data, except the x-axis a bit trickier.

Spatial Frequency Stimulus Information

Full Screen Width:	11.5 cm
Screen Pixel Width:	800 px
Visible Width:	6.5 cm
Visible Height:	4.0 cm
Distance From Screen:	2.3 cm

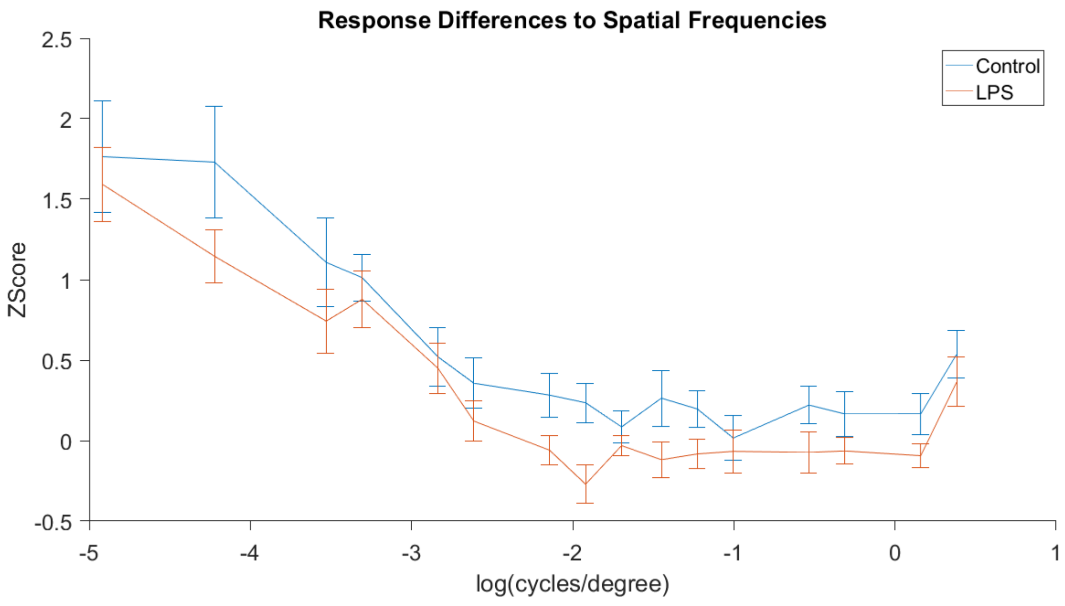
	Cycles Width (Pixels)	Aprx. Spatial Frequency* (Cycles/Degree)
Stimulus 1	800	0.0073
Stimulus 2	400	0.0147
Stimulus 3	200	0.0293
Stimulus 4	160	0.0367
Stimulus 5	100	0.0587
Stimulus 6	80	0.0733
Stimulus 7	50	0.1173
Stimulus 8	40	0.1466
Stimulus 9	32	0.1833
Stimulus 10	25	0.2346
Stimulus 11	20	0.2933
Stimulus 12	16	0.3666
Stimulus 13	10	0.5865
Stimulus 14	8	0.7332
Stimulus 15	5	1.173
Stimulus 16	4	1.4663

*Does not include distortion due to flat screen. Naively calculated as:

```
   T = compFact(800)/(180/pi*2*atan(11.5/(2*2.3)));
   T = T(1:16);
```

```
compFact(n)
  Input:  Integer
  Output: All factors of n
```

```
>> T = compFact(800)/(180/pi*2*atan(11.5/(2*2.3)));
>> T = T(1:16);
>> figure
>> errorbar(log(T),Control.Average_ZScore,Control.StdError_ZScore)
>> hold on;
>> errorbar(log(T),LPS.Average_ZScore,LPS.StdError_ZScore)
>> xlabel('log(cycles/degree)')
ylabel('ZScore')
box off
set(gca,'TickDir','out')
title('Response Differences to Spatial Frequencies')
set(gca,'FontSize',18)
legend('Control','LPS')
```



"figure" creates a new figure. We also called errorbar a bit different this time. We defined the x coordinates with the first parameter "log(T)". Previously we didn't define the x coordinates, and got around it by labeling the x-axis differently. Either way works. Because the spacing isn't linear for spatial frequency, defining the x coordinates explicitly is a lot easier.

Making Videos

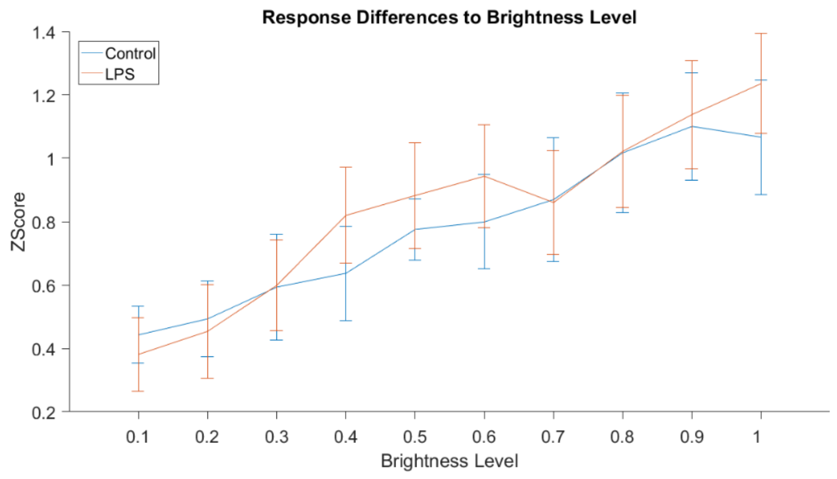
The function BuildVideo(Folder,FrameRate) will create a video with averaged frames for each stimulus. For example

```
>> BuildVideo('E:\Data\Spatial\LPS\Cassian spatial frequency 1',10)
```

Will create a video for experiment Cassian spatial frequency, with 10 frame per second. Since each stimulus response lasts 25 frames, this entire video will last 25*17/10 seconds

Extracting Data from Graphs

Suppose you have a graph and you want the data that made the graph, for example:



It's actually not too difficult. With the figure open in MATLAB,

```
>> Children = get(gca,'Children')
Children =
  2x1 ErrorBar array:

  ErrorBar
  ErrorBar
```

You see it has two "Children." Remember gca refers to the axes of the current figure.

```
>> Children(1)
ans =
  ErrorBar with properties:

        Color: [0.85 0.325 0.098]
    LineStyle: '-'
    LineWidth: 0.5
       Marker: 'none'
        XData: [1 2 3 4 5 6 7 8 9 10]
        YData: [1x10 double]
        LData: [1x10 double]
        UData: [1x10 double]

  Show all properties
```

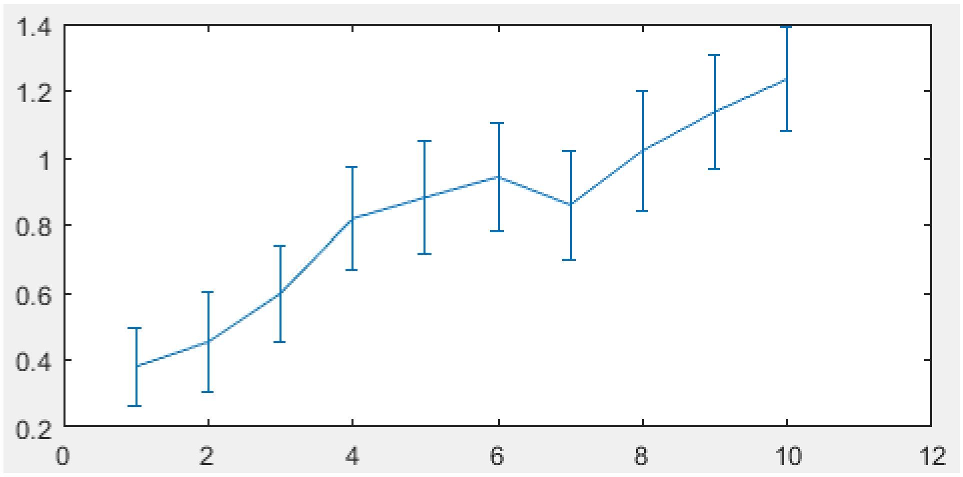
```
>> Children(2)
ans =
  ErrorBar with properties:

        Color: [0 0.447 0.741]
    LineStyle: '-'
    LineWidth: 0.5
       Marker: 'none'
        XData: [1 2 3 4 5 6 7 8 9 10]
        YData: [1x10 double]
        LData: [1x10 double]
        UData: [1x10 double]

  Show all properties
```

Each one of these Children is an "ErrorBar," Notably, with properties XData, YData, LData, UData. X- and YData are self-explanatory. The LData and UData stand for upper and lower data.

```
>> X = Children(1).XData
X =
     1     2     3     4     5     6     7     8     9    10
>> Control_Mean = Children(1).YData
Control_Mean =
  Columns 1 through 2
         0.380875949990283         0.453575249743376
  Columns 3 through 4
         0.598613025273544         0.819650919422158
  Columns 5 through 6
         0.882295635260979         0.943283419366387
  Columns 7 through 8
         0.860266828394159          1.02191074402883
  Columns 9 through 10
          1.13783686055511           1.2356794918525
>> Control_SE = Children(1).LData
Control_SE =
  Columns 1 through 2
         0.116265020376648         0.148615528394817
  Columns 3 through 4
         0.142466435491292          0.15103417399634
  Columns 5 through 6
         0.166590124672407         0.163258015480116
  Columns 7 through 8
         0.162876936985939         0.177287974829286
  Columns 9 through 10
         0.170430355862437         0.157504172817877
```

We can see that with this, we can recreate the plot[4]

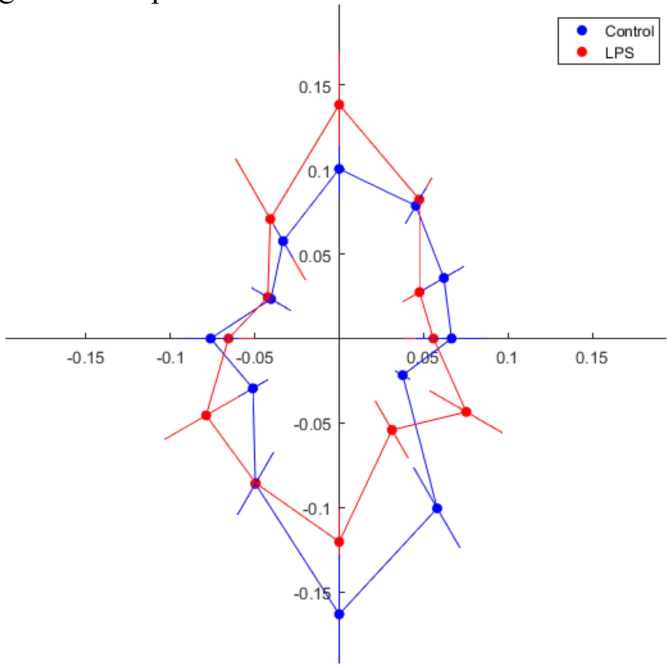
```
>> errorbar(X,Control_Mean,Control_SE)
```



Turns out Children(1) was actually the LPS data, not the control data…whoops

[4] We won't need both the UData and LData, because the error bars are symmetric around the line. Hence, UData and LData actually contain the same numbers, namely the Standrad Error

Let's look at something more complicated:



```
>> Children = get(gca,'Children')
Children =
  29x1 graphics array:

  Line
  Line
  Line
  Line
  Line
  Line
  Line
...
  Line
  Line
  Line
  Scatter     (LPS)
  Scatter     (Control)
>> |
```

With all the individually defined error bars, there are a lot of children here. However, getting the scatter information is still just as simple:

```
>> Children(25)
ans =
  Scatter with properties:

             Marker: 'o'
    MarkerEdgeColor: 'none'
    MarkerFaceColor: 'flat'
           SizeData: 36
          LineWidth: 0.5
              XData: [1x12 double]
              YData: [1x12 double]
              ZData: [1x0 double]
              CData: [1 0 0]
```

Making Histograms

In very complicated situations like the one above, it may be easier to recalculate the data yourself. The figure was a "radial histogram," so here's an opportunity to show how to make histograms in MATLAB.

```
>> load('E:\Data\Direction\Control\Sampled.mat')
>> for i = 1:RoiMin
for j = 1:size(ZScore,3)
Preferred_Direction(i,j) = find(ZScore(i,:,j) == max(ZScore(i,:,j)));
end
end
```

We load our sampled data. Recall our ZScore dimensions. The first is the ROI dimension, the second is the stimulus dimension, and the third is the fish dimension. Just looking at the code, you can probably intuit that we're finding the stimulus that produced the maximum ZScore for each ROI in each Fish. To see this more clearly, we loop through each ROI, then each Fish, and we get to this line:

```
    Preferred_Direction(i,j) = find(ZScore(i,:,j) == max(ZScore(i,:,j)));
```

So we have the fish and ROI specified by i and j respectively.

```
>> temp = ZScore(i,:,j);
>> whos temp
  Name      Size            Bytes  Class     Attributes

  temp      1x12               96  double
```

Remember, for direction we have 12 stimuli. So we're looking at the ZScores for ROI i in fish j.

Then we say

```
(ZScore(i,:,j) == max(ZScore(i,:,j))
```

"max" is returning the maximum ZScore. "==" compares whether two things are equal.[5] So this returns 1 x 12 variable of 1s and 0s. 1 if that ZScore is equal to the maximum ZScore and 0 otherwise:

```
>> ZScore(i,:,j) == max(ZScore(i,:,j))
ans =
     0     0     0     0     1     0     0     0     0     0     0     0
```

The function "find" simply finds all the 1s in a list of zeros and ones.

```
>> find(ZScore(i,:,j) == max(ZScore(i,:,j)))
ans =
     5
```

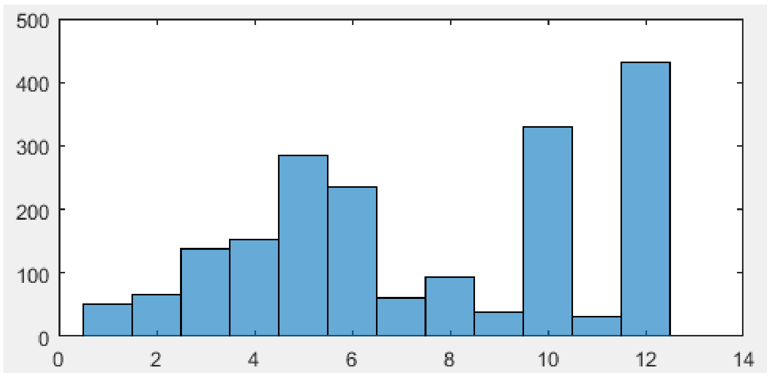
[5] Since the left hand side is a vector, it will compare each element of the vector to the right hand side

And in this way we get the preferred direction for each ROI in each Fish. All in 5 lines of code.

Now for the histogram part. Let's calculate the histogram for each fish 1.

```
>> whos Preferred_Direction
  Name                          Size                Bytes  Class     Attributes

  Preferred_Direction       1913x9                 137736  double

>> histogram(Preferred_Direction(:,1),[0.5:12.5])
```

The second parameter of histogram specifies the edges of each bin, that is, where each bar starts and ends in the plot. Here we defined [0.5 1.5 2.5 3.5 4.5 5.5 6.5 7.5 8.5 9.5 10.5 11.5 12.5] to be the edges, which we can see in the graph it produced:



The Y axis is how many times that preferred direction appeared in the data set. Approximately 450 ROIs in fish 1 responded best to direction 12 (360 degrees.) To get the exact number we need to get the values from the graph:

```
>> H = get(gca,'Children')
H =
  Histogram with properties:

             Data: [1913x1 double]
           Values: [1x12 double]
          NumBins: 12
        BinEdges: [1x13 double]
        BinWidth: 1
       BinLimits: [0.5 12.5]
    Normalization: 'count'
       FaceColor: 'auto'
       EdgeColor: [0 0 0]

  Show all properties
>> Values = H.Values
Values =
  Columns 1 through 9
    52    66   138   153   284   235    60    94    38
  Columns 10 through 12
   330    32   431
```

Now, since our data is interpreted radially, it would be nice to display it polar coordinates instead of horizontally. To do this requires a foray into some basic geometry:

Firstly, we get the angles:

```
>> Theta = [30:30:360]
Theta =
  Columns 1 through 9
    30     60     90    120    150    180    210    240    270
  Columns 10 through 12
   300    330    360
```

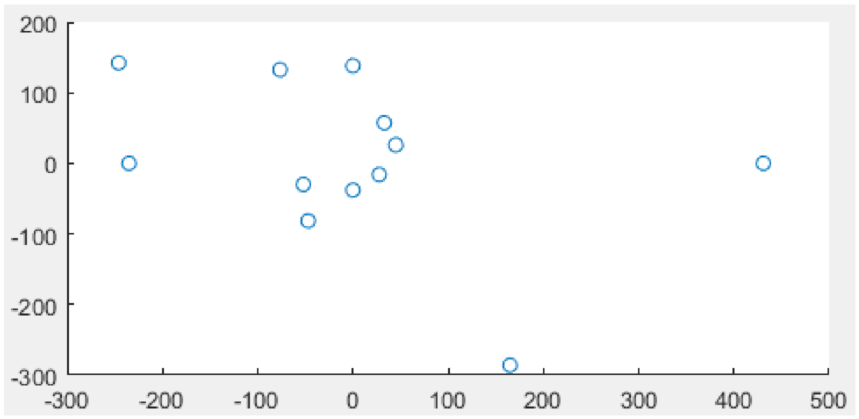
Then convert them from degrees to radians:

```
>> Theta = Theta*pi/180
Theta =
  Columns 1 through 2
         0.523598775598299           1.0471975511966
  Columns 3 through 4
           1.5707963267949           2.0943951023932
  Columns 5 through 6
          2.61799387799149          3.14159265358979
  Columns 7 through 8
          3.66519142918809          4.18879020478639
  Columns 9 through 10
          4.71238898038469          5.23598775598299
  Columns 11 through 12
          5.75958653158129          6.28318530717959
```

Then we break Values into the appropriate X and Y coordinates:

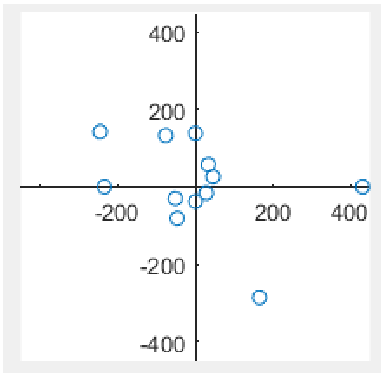
```
>> X = cos(Theta).*Values;
>> Y = sin(Theta).*Values;
```

And then we can graph this using the "Scatter" function:



Now to interpret this, we need to format it a bit nicer.

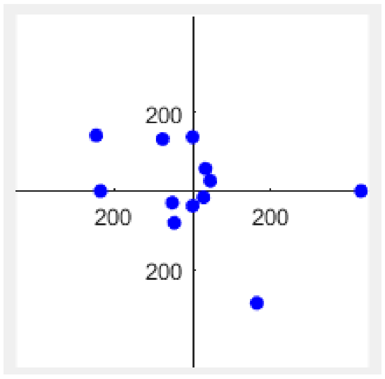
```
>> set(gca,'XAxisLocation','origin')
set(gca,'YAxisLocation','origin')
xlim([-450 450]);
ylim([-450 450]);
axis square
```



We put the axes in the center of our figure and we made the scale on the x and y axis the same so that all our proportions are the same.

More can be done to make it look nice. For example:

```
>> scatter(X,Y,'filled','b')
>> set(gca,'XAxisLocation','origin')
set(gca,'YAxisLocation','origin')
xlim([-450 450]);
ylim([-450 450]);
>> axis square
>> set(gca,'XTick',[-200:200:200])
>> set(gca,'YTick',[-200:200:200])
>> set(gca,'XTickLabel',{'200','','200'})
set(gca,'YTickLabel',{'200','','200'})
```



More Histograms

Finally, let’s look at normalizing our histogram to PDFs and CDFs. For this, let us look at our darkness data.

```
>> load('E:\Data\Darkness\Control\Sampled.mat')
>> whos Activity
  Name                Size                    Bytes  Class     Attributes

  Activity      1781x3000x9             384696000  double
```

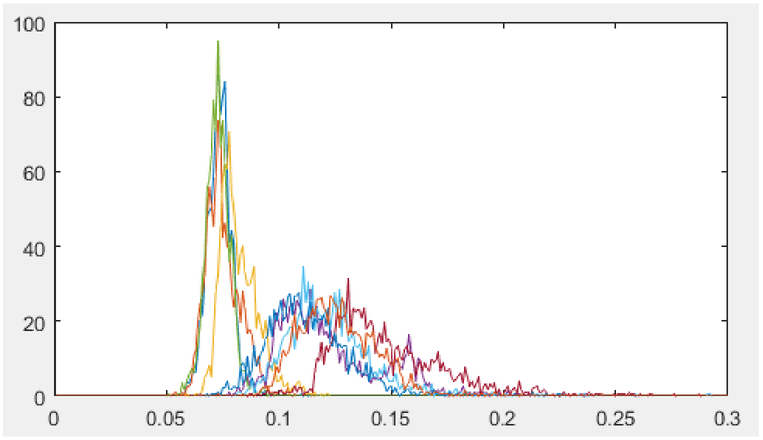
Since Darkness doesn’t have any stimuli, we obviously can’t have response data or ZScore data. Instead we have “Activity” which is simply the raw $\Delta F/F_0$. Here the first dimension is the ROI dimension, the second is the time dimension, and the third is the fish dimension.

Let’s find the average $\Delta F/F_0$ for each ROI.

```
>> Mean_Activity = mean(Activity,2);
>> whos Mean_Activity
  Name                      Size               Bytes  Class     Attributes

  Mean_Activity      1781x1x9             128232  double

>> Mean_Activity = permute(Mean_Activity,[1 3 2]);
>> whos Mean_Activity
  Name                      Size              Bytes  Class     Attributes

  Mean_Activity      1781x9              128232  double
```

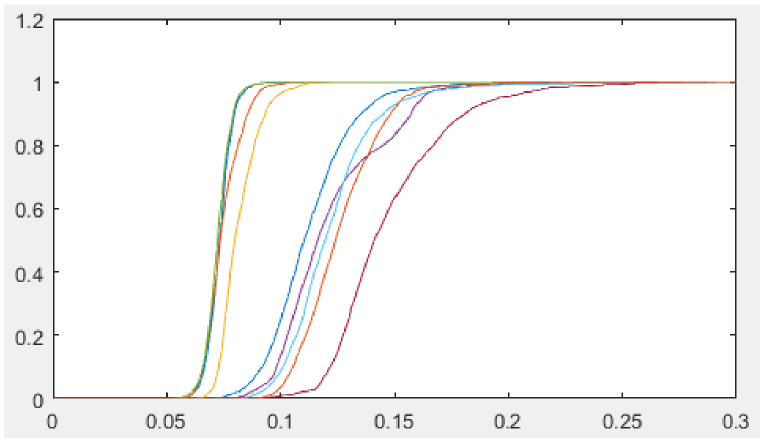
Now let’s look at the distribution for each fish:

```
>> BinEdges = [0:0.001:0.3];
>> for i = 1:size(Activity,3)
A = histogram(Mean_Activity(:,i),BinEdges,'Normalization','pdf');
Values(:,i) = A.Values;
end
>> plot(BinEdges(2:end),Values)
```



We can do the exact same for the CDF:

```
BinEdges = [0:0.001:0.3];
for i = 1:size(Activity,3)
A = histogram(Mean_Activity(:,i),BinEdges,'Normalization','cdf');
Values(:,i) = A.Values;
end
plot(BinEdges(2:end),Values)
```



There are a few differences in how we made the histograms this time compared to last time.

Before we retrieved the histogram handle indirectly, in this example, we assign it explicitly while creating the histogram.

Furthermore, we define the x coordinate explicitly with BinEdges. BinEdges is going to be one longer than the number of actual bins, hence we ignore the first BinEdge. In fact, to do this properly, we should take the midpoint from each bin, but since we have so many bins, there isn't going to be a perceptible difference.

Code Overview

Functions

Name	Inputs	Output
BuildVideo.m	Folder, FrameRate	Builds video with averaged stimulus responses. Saves Experiment(Slice#).avi
FolderRecursion.m	Folder, Function	Runs Function (e.g. Function = @BuildVideo) on all subfolders of Folder
RoiClustering.m	AnalysedData, header	Clusters most-correlated ROIs together into clusters, and returns these groups
Smooth.m	Vectors, Window	Vector with moving average filter applied, filter window defined by Window (e.g. [-5 5])
compFact.m	Number	Returns all factors of the number Number
readLines.m	FileName	Returns text of file as a cell array
tabulate.m	Cell Array	Split each cell at commas and returns new cell array
readRawImage.m	fileName, imageSize, imageNumber	Reads image number ImageNumber from file fileName (Image_0001_0001.raw)
runLinearModel.m	Folder	Open Sampled.mat in Folder, computes linear models on dF data and saves it to Sampled.mat
suint16.m	Matrix	Returns normalized values, where max of matrix is now 2^16-1
suint8.m	Matrix	Returns normalized values, where max of matrix is now 2^8-2
tif2raw.m	firstNumber, lastNumber, SliceCount, Folder	Assumes .tifs of format ChanA_0001_0001_'Slice'_'Number'. Resaves them as a .raw file
time2Frame.m	FrameTimes, AnalysedData	Converts Frame Time (in seconds) to frame number
uniqueElements.m	Vector	Returns all the unique elements in the vector Vector

Code Overview

DataAnalysis\src\Data Extraction

Name	Inputs	Outputs
AnalyzeData.m	None	GUI for extractData
extractData.m	Option 1: Folder Option 2: header, ImageData Option 3: Folder, [], 0	Option 1: Analyzes data in Folder. Outputs .mat files Option 2: outputs Analysed '…'.mat file Option 3: Analyzes data in Folder without registration
sampleData.m	Folder	Sampled.mat containing random sample of each fish in Folder

DataAnalysis\src\Plot

Name	Inputs	Outputs
PlotRoiData.m	None	GUI for looking at data
getDirSelMap.m	Folder	Generates selectivty map for each slice. Saves as .fig
guiCorr.m	Folder, Slice	GUI to look at correlated ROIs in slice Slice

Different Analyses and Future Directions

Cluster Analysis

The correlation between each ROI's calcium signal is calculated. The highest two ROIs are merged into a "Cluster" which is a structure containing the ID of all ROIs merged into it and the averaged calcium trace of all ROIs in the Cluster. The Cluster is itself treated as an ROI, and the process is repeated, merging ROIs with ROIs, ROIs with Clusters, and Clusters with Clusters until no ROIs or Clusters have a correlation greater than some set correlation value.

This is implemented in the function RoiClustering.m

Cluster analysis and unsupervised machine learning classification algorithms is a burgeoning field. More refined methods could be used to identifying groups of cells that behaviour similarly, allowing us to classify different cell types in the tectum. Looking at how cells respond to different stimulus types in addition to different parameters for the same stimulus type could be an interesting thing to investigate.

Stereotyped Calcium Trace

Due to the level of noise in our system, we don't see very nice calcium responses. The response we calculate is the average calcium activity post-stimulation, so no information about the shape of the response is recorded. This is another area where we could explore more.

For the most part, naïve thresholding has proven unhelpful to identify activity. Perhaps with appropriate denoising and deconvolution techniques, we could investigate specific calcium response properties. This could give us more insight into cell properties and classifying cell types.