CaliberVariation_KAC00

KAC

2024-11-22

Introduction

Starting notes: This is an R Markdown document. For more details on using R Markdown see http://rmarkdown.rstudio.com (http://rmarkdown.rstudio.com). When you click the **Knit** button, an html document will be generated that includes both content as well as the output of any embedded R code chunks within the document. An R code chunk is embedded below.

Note that the `echo = FALSE` parameter can be added to the code chunk (instead of echo = TRUE) to prevent printing of the R code that generated the output you see (e.g. the plot).

RbDotPlotBranchRules_KAC01:

This file takes the RB catalog multipoint measurement data and makes various dot plots from it for analysis. It includes analyses for comparing branching results/rules to Paheli Desai-Chowdry's paper (Desai-Chowdry et al., 2022).

Get started

Get started by loading the tidyverse tools, which includes ggplot2. If you haven't used tidyverse on this computer before, you need to do install.packages("tidyverse") first. You'll only need to install once, but you need to load at the start of each session. You could, alternatively, just do library("ggplot2"), but know there might be some tools used here that are in tidyverse but not ggplot2.

```
library("tidyverse")
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.6     v dplyr   1.0.8
## v tidyr   1.2.0     v stringr 1.4.0
## v readr   2.1.2     v forcats 0.5.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Load the data

Read the csv file to load the measurements data. This will give you the data as a data frame. (A data frame is list of a specific class, so typeof(measurements) will tell you that it's a list.)

Note that you might have to change the direction of the slashes on different computers.

```
measurements = read.csv("G:/My Drive/DesktopItems/Measurements/RbCatalog/MultiPointMeasurement/KAC_RbCatalog_MultiMeasure1Dpf.csv")

NumberOfAxons = length(measurements$PrimaryDiameter)
```

Sort the secondaries

Go through the measurements and make the larger of the two secondary branches be Secondary1.

Note: I am not taking triple branches into consideration here. If a neuron has three branches, it was excluded from this data set.

```
SortedMeasurements <- measurements
for (i in c(rep(1:length(SortedMeasurements$PrimaryDiameter)))) {
  if (SortedMeasurements$Secondary1Diameter[i] < SortedMeasurements$Secondary2Diameter[i]) {
    Smaller <- SortedMeasurements$Secondary1Diameter[i]
    Larger <- SortedMeasurements$Secondary2Diameter[i]
    SortedMeasurements$Secondary1Diameter[i] <- Larger
    SortedMeasurements$Secondary2Diameter[i] <- Smaller
  }
}
```

Make a better-formatted data frame

Make a data frame using those data. Note: 34 is the number of neurons measured. This can be changed manually based on how many measurements you have.

```
d <- data.frame(Diameters = c(SortedMeasurements$PrimaryDiameter, SortedMeasurements$Secondary1Diameter, SortedMeasurements$Secondary2Diameter),
                group = as.factor(rep(c('PrimaryCaliber', 'Secondary1Caliber', 'Secondary2Caliber'), each = NumberOfAxons)),
                id = SortedMeasurements$ImageID)
```

Show some mean values for 1 dpf data

```
## [1] 1 dpf axon calibers:
```

```
## The average Primary caliber is 0.4320108 and the standard deviation is 0.1066366
```

```
## The average Secondary1 caliber is 0.322314 and the standard deviation is 0.07415999
```

```
## The average Secondary2 caliber is 0.2016817 and the standard deviation is 0.05794988
```

Make a dot plot of caliber values

Now, let's make a preliminary dot plot to see caliber values with lines to connect values that come from the same neuron.

First, I'll define some lists (RangeToShow and BranchTypes) that will make the plotting easier.

```
RangeToShow = c((min(SortedMeasurements$Secondary2Diameter) - 0.1), (max(SortedMeasurements$Prim
aryDiameter) + 0.1))
BranchTypes = as.factor(rep(c('Primary', 'Secondary1', 'Secondary2'), each = NumberOfAxons))
```

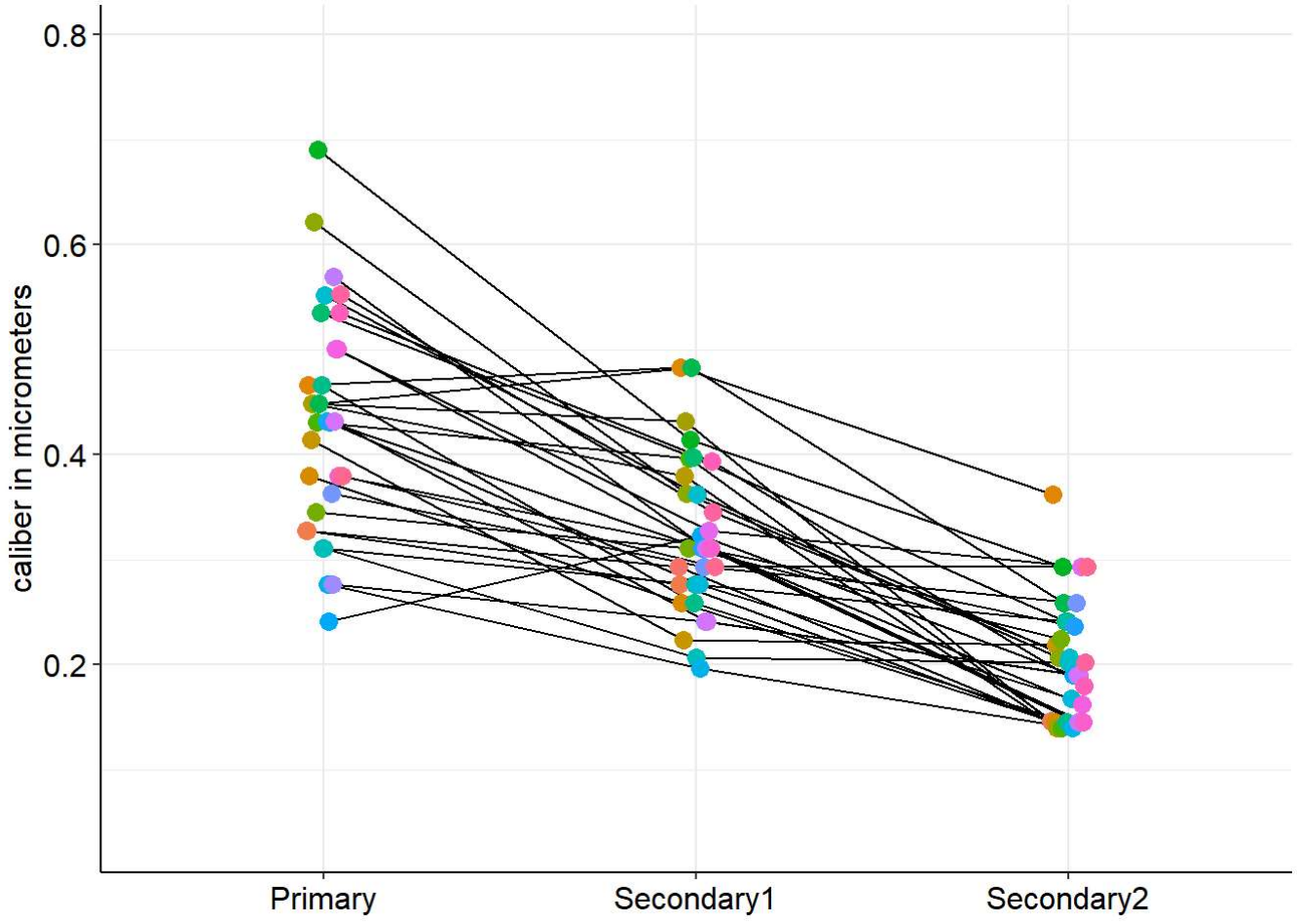
Then, I'll use ggplot2 to make a dot plot. The basic template for making a plot with ggplot2 is:

ggplot(data = DATA) + GEOM_FUNCTION(mapping = aes(MAPPINGS))

I have replaced all of the CAPS SECTIONS with my own data/details, and continued adding + GEOM_FUNCTIONs to add things to the plot. Here are the details for the different parts:

- d is the data frame we made earlier.
- geom_line makes the lines on the plot. I put this part first so that the lines are underneath of the dots. x = BranchTypes tells it where on the x axis to put the lines, and group = id links each data point to its id (PrimaryDiameter, Secondary1Diameter, or Secondary2Diameter)
- geom_point makes the dots for each data point on the plot. Examples of adjustments I've made to various plots in this code: x = BranchTypes tells it where to put the dots, and color = group says that the dots will be color coded by their group. size = 3 gives the size of the dots, and position = position_jitterdodge() adjusts the position of each dot by a small amount of jitter so that all of the dots aren't on top of each other. dodge.width = 0.1 gives how far the jitter will go.
- "scale_colour_manual(values = c("magenta", "green", "green"), guide = "none") +" can be added (along with changing "color = id" to "color = group" in geom_point) if you would rather make the colors match for the different types of branches. scale_colour_manual allows you to set the colors and things manually. values = c("magenta", "green", "green") is how I defined the colors of the dots in each group. guide = "none" removes the guide as to what the colors mean so that it isn't on the plot, as is the default.
- ylim(RangeToShow) defines the limits of the y axis, as defined above.
- theme_bw is the black on white theme. Without this, the default theme is grey.
- theme() gives more details about the theme where all element_blank() sections remove the part that might have been there before, such as the plot background. The other element_ parts specify the colors and sizes of specific parts of the axis. legend.position = "None" is, again, specifying that we don't want a legend on the plot. This may be redundant with the part above.

```
ggplot(d, aes(y = Diameters)) +
  geom_line(aes(x = BranchTypes, group = id), position=position_dodge(width = 0.1)) +
  geom_point(aes(x = BranchTypes, , group = id, color = id), size = 3, position=position_dodge(w
idth = 0.1)) +
  ylim(RangeToShow) +
  ylab("caliber in micrometers") +
  theme_bw() +
  theme(plot.background = element_blank(),
                      panel.border = element_blank(),
                      panel.background = element_blank(),
                      axis.line = element_line(colour = "black"),
                      axis.title.x = element_blank(),
                      axis.title.y = element_text(size = 12, colour = 'black'),
                      axis.text.x = element_text(size = 12, colour = 'black'),
                      axis.text.y = element_text(size = 12, colour = 'black'),
                      legend.position = "None")
```



Assess the ratio of calibers

Now I want to make a dot plot with the ratios of primary to secondary calibers an connect them for values in the same neuron. To start, I'll find the ratios of primary to each secondary.

To make this simpler than how I did it above, I'll start by just making a vector of values for the size of the measurements data frame.

```
RangeOfNeurons = c(rep(1:length(SortedMeasurements$PrimaryDiameter)))
```

Next, I'll create some empty vectors to store the ratio values in.

```
S1toP <- c()
S2toP <- c()
```

Then, I'll go through all of the neurons and store the ratio of secondary to primary in the appropriate vector. You can add "print(R1) + print(R2)" to the end of the for loop to check what values you're adding.

```
for (i in RangeOfNeurons) {
  P = SortedMeasurements$PrimaryDiameter[i]
  S1 = SortedMeasurements$Secondary1Diameter[i]
  S2 = SortedMeasurements$Secondary2Diameter[i]
  R1 = S1/P
  S1toP <- c(S1toP, R1)
  R2 = S2/P
  S2toP <- c(S2toP, R2)
}

cat("The average S1/P ratio is", mean(S1toP),
    "with SEM", sd(S1toP)/sqrt(length(S1toP)), ",", "\n",
    "and the average S2/P ratio is", mean(S2toP),
    "with SEM", sd(S2toP)/sqrt(length(S2toP)), ".")
```

```
## The average S1/P ratio is 0.7684307 with SEM 0.03256788 ,
##  and the average S2/P ratio is 0.4893174 with SEM 0.02975211 .
```

Next, I'll define a new data frame to put the ratios in.

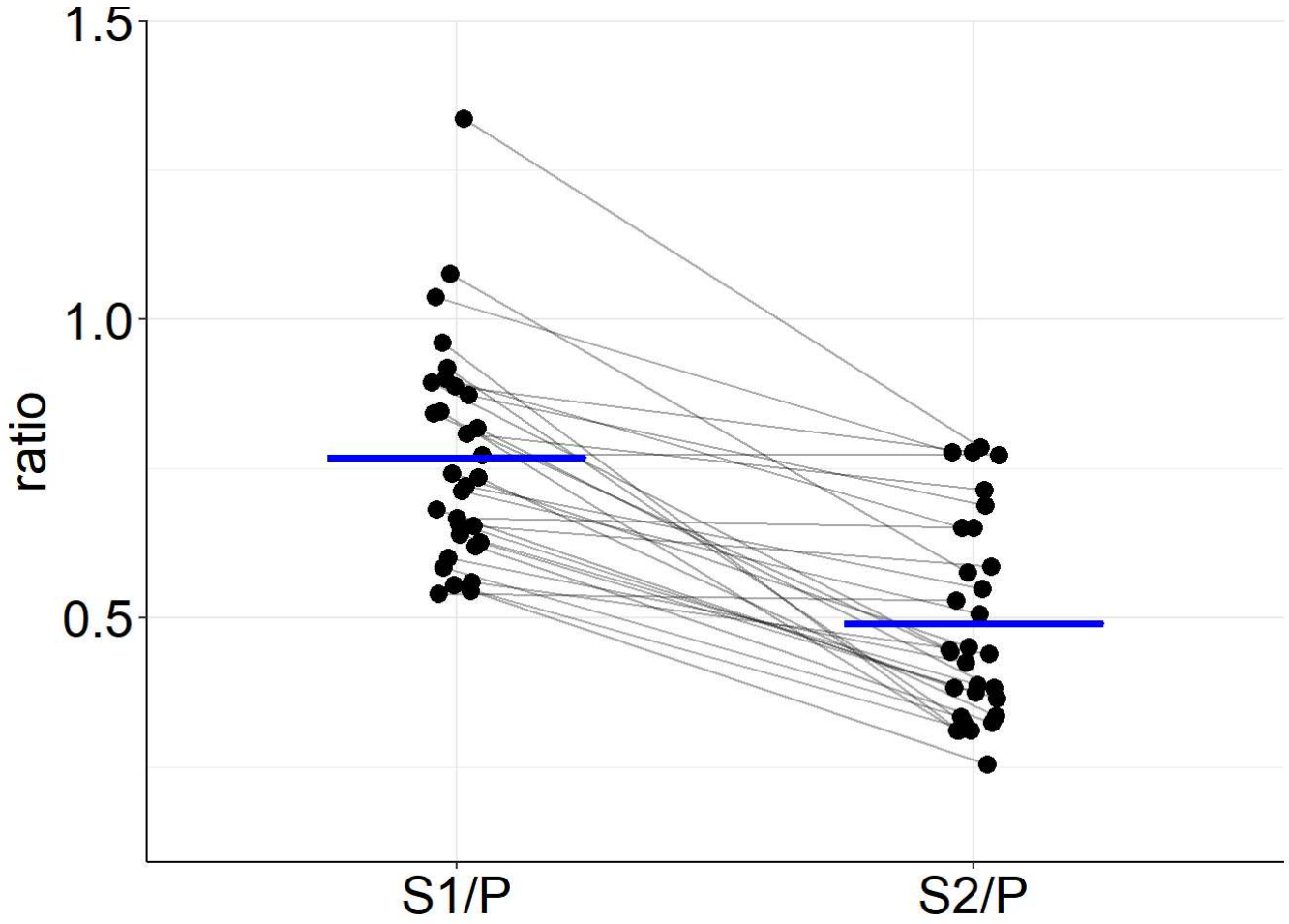
```
RatiosDataFrame <- data.frame(Ratios = c(S1toP, S2toP),
                group = as.factor(rep(c("S1P", "S2P"), each = NumberOfAxons)),
                id = SortedMeasurements$             ImageID)
```

I'll define some other things to make plotting easier.

```
RatioTypes = as.factor(rep(c("S1/P", "S2/P"), each = NumberOfAxons))
RatioRange = c((min(S2toP) - 0.1), (max(S1toP) + 0.1))
```

Using the same principles as above, I make a plot with the ratio data. In this plot, dots are colored based on the neuron id so that points can be matched more easily between the two columns. (Note: More recently, I added lines for the mean values in each column. This is explained in comments further down in the code.)

```
ggplot(RatiosDataFrame, aes(y = Ratios)) +
  geom_line(aes(x = RatioTypes, group = id), position = position_dodge(width = 0.1), alpha = 1/
3) +
  geom_point(aes(x = RatioTypes, group = id), size = 3, position = position_dodge(width = 0.1))
+
  geom_crossbar(data = data.frame(S1toP), aes(x = "S1/P", ymin=mean(S1toP), ymax=mean(S1toP), y
= mean(S1toP)), width = 0.5, color = "blue") +
  geom_crossbar(data = data.frame(S2toP), aes(x = "S2/P", ymin=mean(S2toP), ymax=mean(S2toP), y=
mean(S2toP)), width = 0.5, color = "blue") +
  ylim(RatioRange) +
  ylab("ratio") +
  theme_bw() + theme(plot.background = element_blank(),
                     panel.border = element_blank(),
                     panel.background = element_blank(),
                     axis.line = element_line(colour = "black"),
                     axis.title.x = element_blank(),
                     axis.title.y = element_text(size = 20, colour = "black"),
                     axis.text.x = element_text(size = 20, colour = "black"),
                     axis.text.y = element_text(size = 20, colour = "black"),
                     legend.position = "None")
```



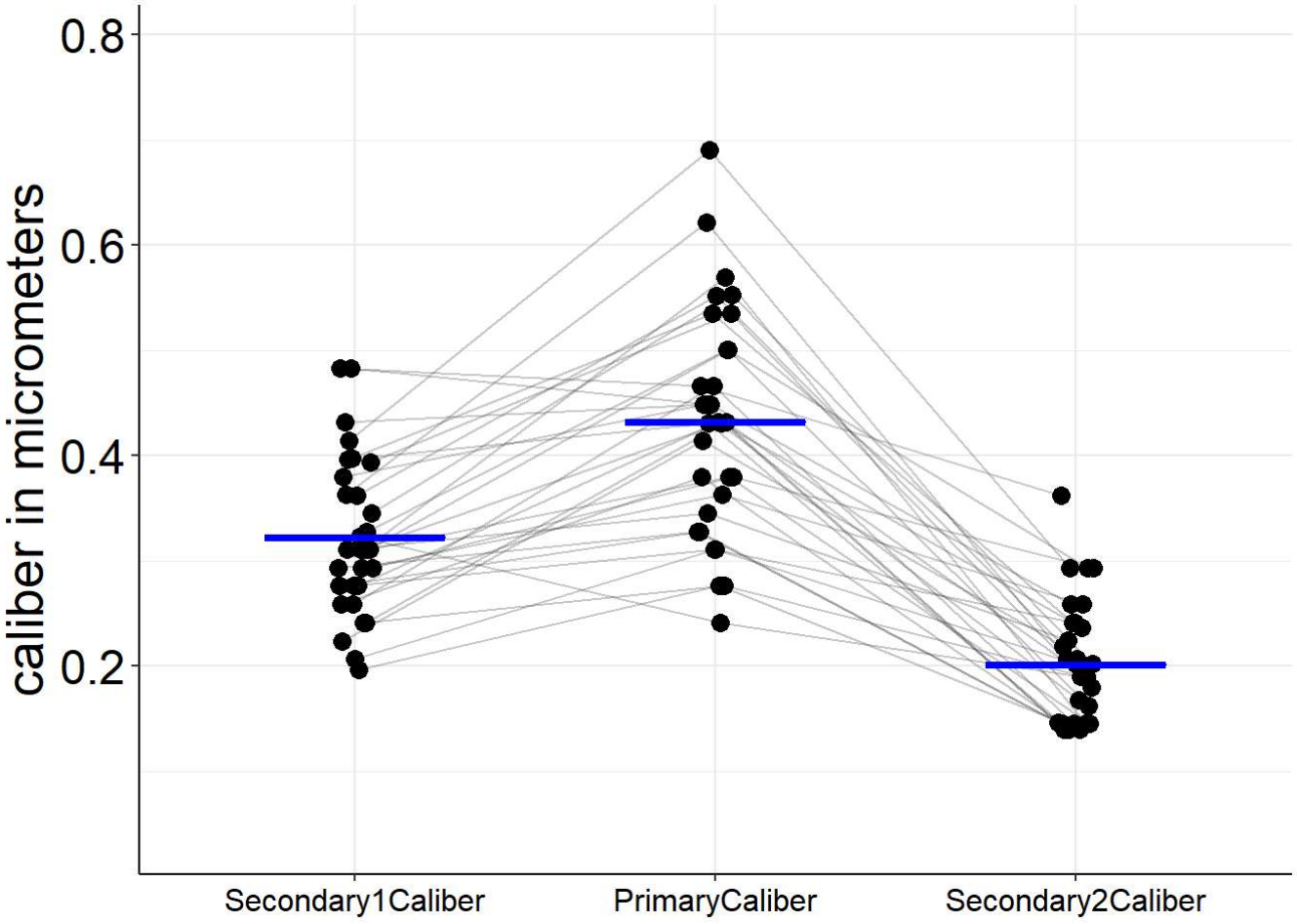
Assessing all branch calibers

First, I sorted for some symmetry assessments, which will be used below.

```
SortedMeasurementsSym <- SortedMeasurements

for (i in c(rep(1:length(SortedMeasurementsSym$PrimaryDiameter)))) {
  comparison = 2 * (SortedMeasurementsSym$Secondary1Diameter[i] - SortedMeasurementsSym$Secondar
y2Diameter[i]) / (SortedMeasurementsSym$Secondary1Diameter[i] + SortedMeasurementsSym$Secondary2
Diameter[i])

  SortedMeasurementsSym$Symmetry[i] <- comparison
}

RatiosDataFrameSym <- data.frame(Ratios = c(S1toP, S2toP),
                group = as.factor(rep(c("S1P", "S2P"), each = NumberOfAxons)),
                id = SortedMeasurementsSym$ImageID,
                symmetry = SortedMeasurementsSym$Symmetry)
```

I also want to plot all of the data together with blue lines for the mean caliber values.

```
##add the symmetry information to the dataframe d
##Now, it's called dall
dall <- data.frame(Diameters = d$Diameters,
                group = d$group,
                id = d$id,
                symmetry = SortedMeasurementsSym$Symmetry)

all.p <- data.frame(dall[which(dall$group == "PrimaryCaliber"),])
all.s1 <- data.frame(dall[which(dall$group == "Secondary1Caliber"),])
all.s2 <- data.frame(dall[which(dall$group == "Secondary2Caliber"),])

level_order <- c('Secondary1Caliber', 'PrimaryCaliber', 'Secondary2Caliber')


##Plot all axons.
ggplot(dall, aes(y = Diameters)) +
  scale_x_discrete(limits = level_order) +
  geom_line(aes(x = group, group = id), alpha = 1/5, position=position_dodge(width = 0.1)) +
  geom_point(aes(x = group, group = id), size = 3, position=position_dodge(width = 0.1)) +

  geom_crossbar(data = data.frame(all.p), aes(x = "PrimaryCaliber", ymin=mean(Diameters), ymax=m
ean(Diameters), y=mean(Diameters)), width=0.5, color="blue") +
  geom_crossbar(data = data.frame(all.s1), aes(x = "Secondary1Caliber", ymin=mean(Diameters), ym
ax=mean(Diameters), y=mean(Diameters)), width=0.5, color="blue") +
  geom_crossbar(data = data.frame(all.s2), aes(x = "Secondary2Caliber", ymin=mean(Diameters), ym
ax=mean(Diameters), y=mean(Diameters)), width=0.5, color="blue") +

  ylim(RangeToShow) +
  ylab("caliber in micrometers") +
  theme_bw() +
  theme(plot.background = element_blank(),
                      panel.border = element_blank(),
                      panel.background = element_blank(),
                      axis.line = element_line(color = "black"),
                      axis.title.x = element_blank(),
                      axis.title.y = element_text(size = 20, color = 'black'),
                      axis.text.x = element_text(size = 12, color = 'black'),
                      axis.text.y = element_text(size = 18, color = 'black'),
                      legend.position = "None")
```



Histogram of symmetry values

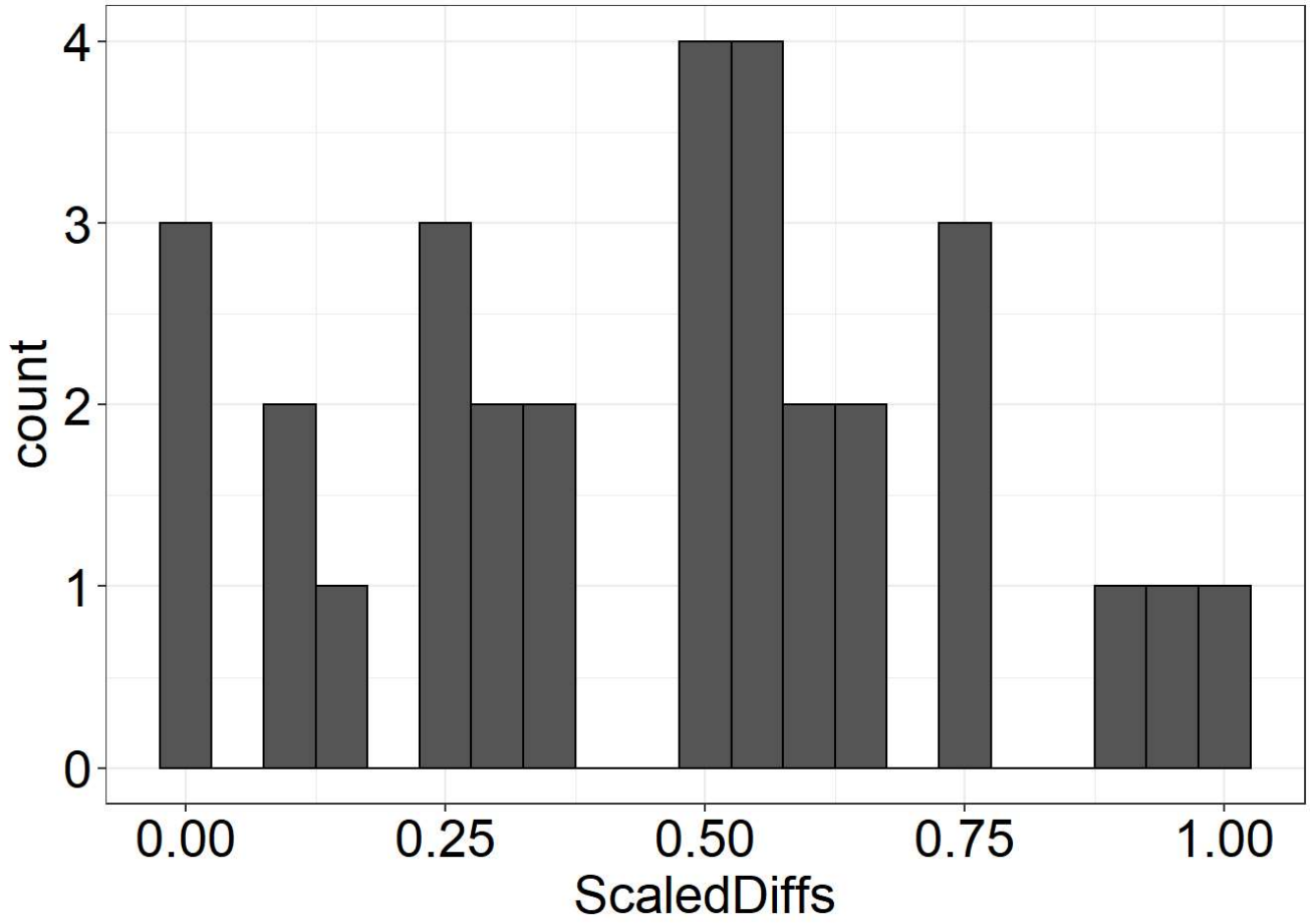
I'd like to see a histogram of values describing how symmetric each neuron's proximal branch point is. I'm interested to know:

Are there two categories (symmetric and asymmetric), or is symmetry a continuous variable, with neurons ranging from symmetric to very asymmetric?

To do this, I'll start by making a dataframe with a value for symmetry for each neuron. This value will be the difference between the secondary calibers divided by the mean of the secondaries' calibers.

After that, I'll make the plot using ggplot.

```
ScaledDiffs <- (SortedMeasurements$Secondary1Diameter - SortedMeasurements$Secondary2Diameter) /
((SortedMeasurements$Secondary1Diameter + SortedMeasurements$Secondary2Diameter) /2)

dfHist <- data.frame(SortedMeasurements[c(1,3,4,5)], ScaledDiffs)

ggplot(dfHist, aes(x = ScaledDiffs)) +
  geom_histogram(binwidth = 0.05, color = "black") +
  theme_bw() +
  theme(axis.title.x = element_text(size = 20, color = "black"),
                    axis.title.y = element_text(size = 20, color = 'black'),
                    axis.text.x = element_text(size = 20, color = 'black'),
                    axis.text.y = element_text(size = 20, color = 'black'))
```



Assessing distribution of the symmetry data.

A possibility that would potentially change my interpretation of the data is that the differences in caliber between secondaries are just uniformly distributed. This might suggest that the axon asymmetry is not a programmed cell shape, but rather something that happens in a large portion of the data, possibly by chance. (Regardless of this, the developmental data suggest that branches are regulated differently over time. This section addresses the starting state at 27 - 30 hpf.)

To test this, I will perform a Kolmogorov-Smirnov test, which can test the likelihood that a set of unbinned data came from a given distribution. (Note: A slightly different way of doing this test can compare two sets of data, but I'm just comparing to a uniform distribution here.)

To get around the ties, I could make a new list where I break ties by adding 0.000000001 to one of the tied the values, as follows, and rerun the KS test. AdjDiffs <- c(ScaledDiffs[1:23], 0.7262488131, ScaledDiffs[25:31]).

As it stands, this should be approximately correct, as long as there aren't too many ties.

```
ks.test(ScaledDiffs, punif)
```

```
## Warning in ks.test(ScaledDiffs, punif): ties should not be present for the
## Kolmogorov-Smirnov test
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  ScaledDiffs
## D = 0.15721, p-value = 0.4277
## alternative hypothesis: two-sided
```

The result of this shows that this distribution of data could have occurred if sampling from a uniform distribution. In contrast, if I run ks.test(AdjDiffs, pnorm) I get an extremely small p-value, meaning that it is unlikely to have arisen from a normal distribution. This makes me believe that the uniform distribution is definitely possible.

So, from the 1 dpf data, we can conclude that RB neurons frequently produce asymmetric axon branches at the most proximal branch point. By tracking neurons across development, we can determine if local regulation of axon caliber occurs. _____

Caliber Dynamics

Load the data: Longitudinal fluctuation measurements

Read a new csv file to load the data. This will give you the data as a data frame. (A data frame is list of a specific class, so typeof(measurements) will tell you that it's a list.)

Note that you might have to change the direction of the slashes for different operating systems.

```
measurements = read.csv("G:/My Drive/DesktopItems/Measurements/Fluctuation/CaliberFluctuationCor
relation_LogitudinalStats.csv")
```

Make longitudinal comparison.

Start by comparing the caliber fluctuation between 1 and 2 dpf. To do this, we'll subtract the SD at 1 dpf from the SD at 2 dpf. (This can be done for both absolute and normalized, %RSD.) We can take the mean of these values, and we'll call this the OrigEffectSize.

```
OrigEffectSize = mean(measurements$X2dpf_percentSD - measurements$X1dpf_percentSD)

print(OrigEffectSize)
```

```
## [1] -8.091811
```

```
#Note that a negative value indicates that the caliber fluctuates less at 2 dpf than it did at 1
dpf.
```

Perform a paired null hypothesis permutation test for

fluctuation.

For this test, we'll assume that there's actually no difference between the 1 and 2 dpf caliber fluctuations. We can simulate this by randomly flipping the 1 and 2 dpf SD and calculating the test statistic (the mean change in fluctuation, as calculated above) for each simulated data set. These will all be stored. Then, we can calculate the p-value by seeing how many of our simulated values have a magnitude greater than what was actually observed. This tells us how likely it is that our data would have occurred by random chance.

```
#Duplicate the measurements data frame for the permutation test.
MeasurementsForPerm = measurements

#List all rows. (This should give you a list with an index for each sample.)
indices = 1:nrow(MeasurementsForPerm)

#Choose the number of times you want to resample the data.
numResamples = 1000

#Initialize the data frame.
TestStatisticDistribution = rep(NA,numResamples)

#Run through a bunch of imaginary data sets in which the samples have been assigned to 1 vs. 2 d
pf by random chance, like a coin flip. The test statistic will be calculated for each one and st
ored to see the outcome of what random chance might look like, assuming that 1 vs. 2 dpf aren't
actually different.
for (i in 1 : numResamples) {

  #Simulate chance:
  #Use random chance to decide which rows should be swapped.
  shouldFlipRows = runif( nrow(MeasurementsForPerm) ) > .5
  rowsToFlip = which (shouldFlipRows)
  #Temporarily store the 1 and 2 dpf SDs.
  temp1dpf = MeasurementsForPerm[rowsToFlip, 4]
  temp2dpf = MeasurementsForPerm[rowsToFlip, 11]
  #Swap the 1 and 2 dpf measurements.
  MeasurementsForPerm[rowsToFlip, 4] = temp2dpf
  MeasurementsForPerm[rowsToFlip, 11] = temp1dpf

  # calculate statistic
  TestStatisticDistribution[i] = mean(MeasurementsForPerm$X2dpf_percentSD - MeasurementsForPerm
$X1dpf_percentSD)

}

#Calculate the p-value.
pval <- (sum(TestStatisticDistribution >= -OrigEffectSize) +
           sum(TestStatisticDistribution <= OrigEffectSize) ) / (numResamples)


print(pval)
```

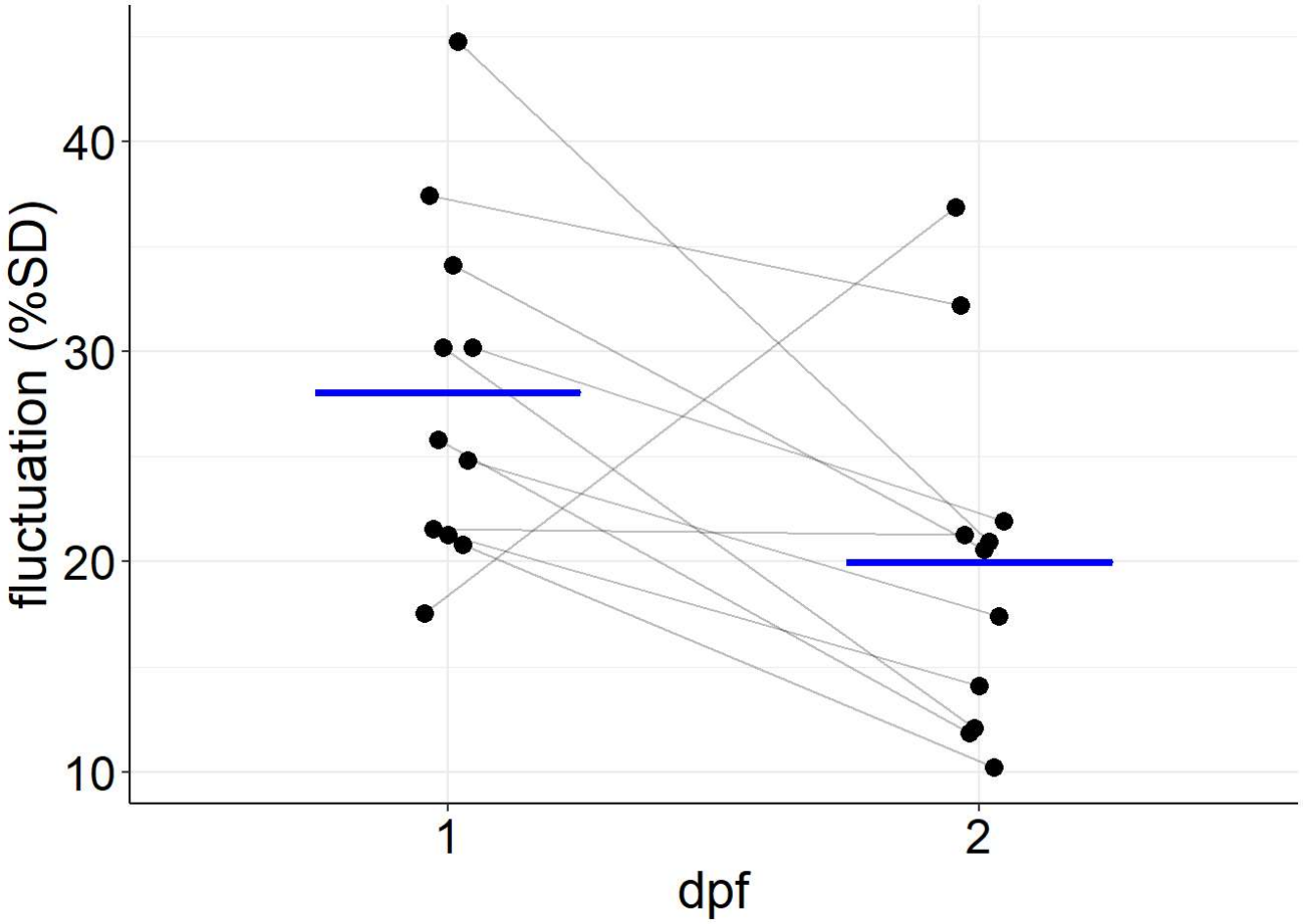
```
## [1] 0.042
```

Plot: Compare dynamics at 1 and 2 dpf.

In this plot, blue lines show the mean value for that column.

I have already loaded tidyverse above, so now, I can make my plots.

```
#Make a data frame specifically for this plot that has all of the percentSD values for 1 and 2 d
pf.
n = length(measurements$axon)
df <- data.frame(axon = as.character(c(measurements$axon, measurements$axon)),
                 percentSD = as.numeric(c(measurements$X1dpf_percentSD, measurements$X2dpf_percen
tSD)),
                 dpf = as.factor(c(rep(1, n), rep(2, n))))

#Make plot using this data frame.
ggplot(df, aes(y = percentSD)) +
  geom_line(aes(x = dpf, group = axon), alpha = 1/4, position=position_dodge(width = 0.1)) +
  geom_point(aes(x = dpf, group = axon), size = 3, position=position_dodge(width = 0.1)) +

  geom_crossbar(data = data.frame(measurements$X1dpf_percentSD), aes(x = "1", ymin=mean(measurem
ents$X1dpf_percentSD), ymax=mean(measurements$X1dpf_percentSD), y=mean(measurements$X1dpf_percen
tSD)), width=0.5, color="blue") +
  geom_crossbar(data = data.frame(measurements$X2dpf_percentSD), aes(x = "2", ymin=mean(measurem
ents$X2dpf_percentSD), ymax=mean(measurements$X2dpf_percentSD), y=mean(measurements$X2dpf_percen
tSD)), width=0.5, color="blue") +

  #ylim(RangeToShow) +
  ylab("fluctuation (%SD)") +
  xlab("dpf") +
  theme_bw() +
  theme(plot.background = element_blank(),
                     panel.border = element_blank(),
                     panel.background = element_blank(),
                     axis.line = element_line(color = "black"),
                     axis.title.x = element_text(size = 20, color = 'black'),
                     axis.title.y = element_text(size = 20, color = 'black'),
                     axis.text.x = element_text(size = 18, color = 'black'),
                     axis.text.y = element_text(size = 18, color = 'black'),
                     legend.position = "None")
```




```
print(mean(measurements$X1dpf_percentSD))
```

```
## [1] 28.04556
```

```
print (mean(measurements$X2dpf_percentSD))
```

```
## [1] 19.95375
```

This plot shows, visually, that the fluctuation decreases from 1 to 2 dpf for all axon segments except for one. The mean fluctuation is also lower at 2 than 1 dpf, as I found in the paired permutation test above.

Plots: Does caliber correlate with dynamicity?

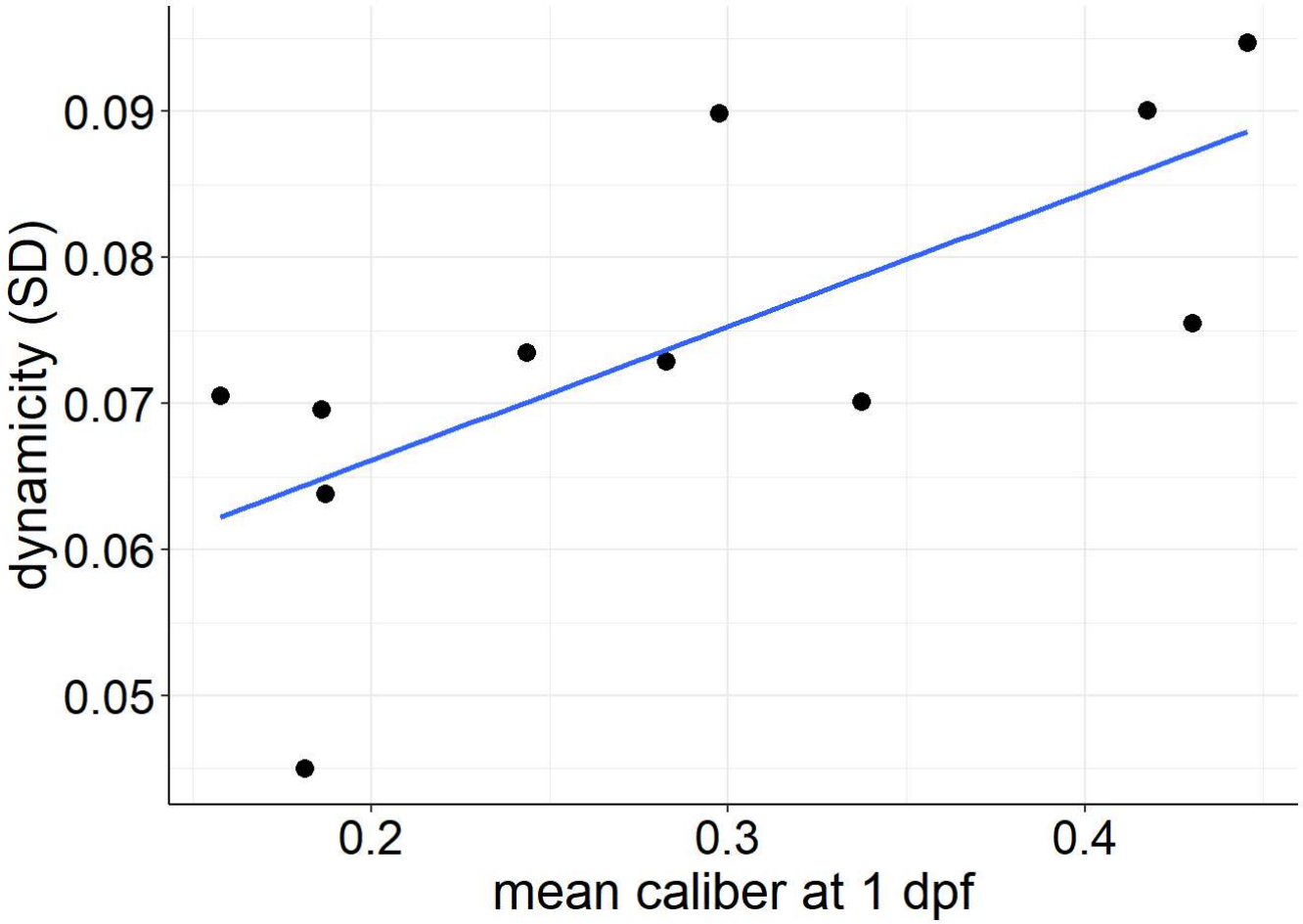
I'd like to plot these data to evaluate trends in how dynamic caliber is. First, I want to ask: Does mean or min caliber predict how dynamic an axon segment is likely to be?

In other words, are thinner or thicker axons more dynamic?

The foundation of these plots comes from RbDotPlotForSym_KAC04.Rmd. I can create this plot for both 1 and 2 dpf.

```
ggplot(measurements, aes(y = X1dpf_SD)) +
  geom_point(aes(x = X1dpf_caliber), size = 3) +
  geom_smooth(aes(x = X1dpf_caliber, y = X1dpf_SD), method=lm, se=FALSE) +
  xlab("mean caliber at 1 dpf") +
  ylab("dynamicity (SD)") +
  theme_bw() +
  theme(plot.background = element_blank(),
                      panel.border = element_blank(),
                      panel.background = element_blank(),
                      axis.line = element_line(color = "black"),
                      axis.title.y = element_text(size = 20, color = 'black'),
                      axis.title.x = element_text(size = 20, color = 'black'),
                      axis.text.x = element_text(size = 18, color = 'black'),
                      axis.text.y = element_text(size = 18, color = 'black'),
                      legend.position = "None")
```

```
## `geom_smooth()` using formula 'y ~ x'
```



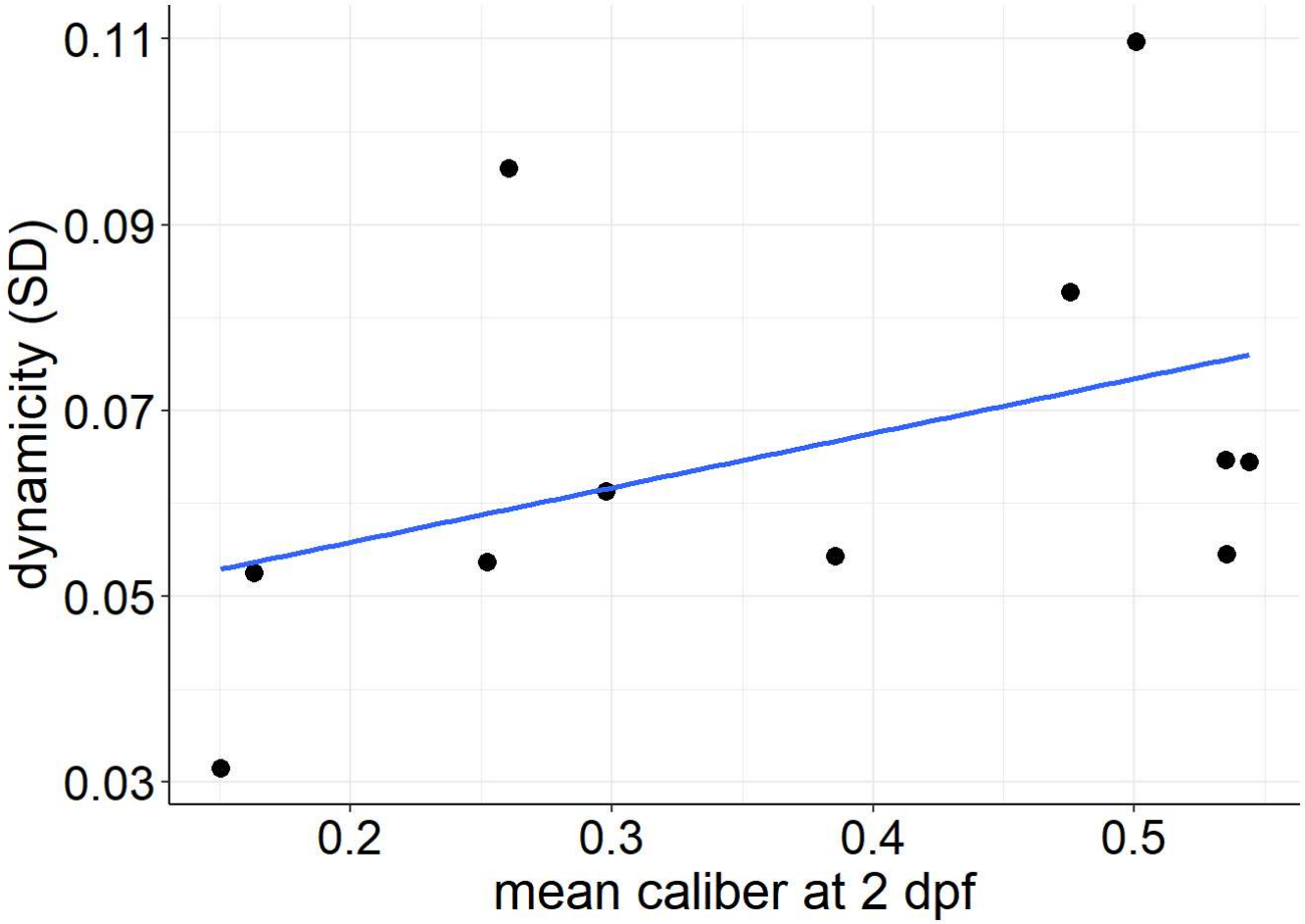

```
print(summary(lm(X1dpf_caliber ~ X1dpf_SD, measurements))$r.squared)
```

```
## [1] 0.5011961
```

From these plots, it seems like thicker axons are less dynamic at 1 dpf. Is this also true on 2 dpf?

```
ggplot(measurements, aes(y = X2dpf_SD)) +
  geom_point(aes(x = X2dpf_caliber), size = 3) +
  geom_smooth(aes(x = X2dpf_caliber, y = X2dpf_SD), method=lm, se=FALSE) +
  xlab("mean caliber at 2 dpf") +
  ylab("dynamicity (SD)") +
  theme_bw() +
  theme(plot.background = element_blank(),
                      panel.border = element_blank(),
                      panel.background = element_blank(),
                      axis.line = element_line(color = "black"),
                      axis.title.y = element_text(size = 20, color = 'black'),
                      axis.title.x = element_text(size = 20, color = 'black'),
                      axis.text.x = element_text(size = 18, color = 'black'),
                      axis.text.y = element_text(size = 18, color = 'black'),
                      legend.position = "None")
```

```
## `geom_smooth()` using formula 'y ~ x'
```




```
print(summary(lm(X2dpf_caliber ~ X2dpf_SD, measurements))$r.squared)
```

```
## [1] 0.1647556
```

Axon Caliber Near Dividing Cells

Load the data: Caliber near divisions measurements

Read a new csv file to load the data. This will give you the data as a data frame. (A data frame is list of a specific class, so typeof(measurements) will tell you that it's a list.)

Note that you might have to change the direction of the slashes for different operating systems.

```
measurements = read.csv("G:/My Drive/DesktopItems/Measurements/Division/Div_DynamOnOff-1st3_norm
-AllData-ForR.csv")
```

Stats: Determine if difference in means is statistically

significant.

```
#Sort calibers by cell roundness (roundness) and location on or off of dividing cell (loc).
on_high <- c()
on_medium <- c()
on_low <- c()
off_high <- c()
off_medium <- c()
off_low <- c()

for (i in c(rep(1:length(measurements$normcal)))) {
  if (measurements$roundness[i] == "high" & measurements$loc[i] == "on") {
    on_high <- c(on_high, measurements$normcal[i])
  }
  if (measurements$roundness[i] == "medium" & measurements$loc[i] == "on") {
    on_medium <- c(on_medium, measurements$normcal[i])
  }
  if (measurements$roundness[i] == "low" & measurements$loc[i] == "on") {
    on_low <- c(on_low, measurements$normcal[i])
  }
  if (measurements$roundness[i] == "high" & measurements$loc[i] == "off") {
    off_high <- c(off_high, measurements$normcal[i])
  }
  if (measurements$roundness[i] == "medium" & measurements$loc[i] == "off") {
    off_medium <- c(off_medium, measurements$normcal[i])
  }
  if (measurements$roundness[i] == "low" & measurements$loc[i] == "off") {
    off_low <- c(off_low, measurements$normcal[i])
  }
}

#Put these stats in a data frame to facilitate plotting later.
stats <- data.frame(roundness = c("high", "medium", "low", "high", "medium", "low"),
                    loc = c("on", "on", "on", "off", "off", "off"),
                    normcal = c(mean(on_high), mean(on_medium), mean(on_low), mean(off_high), me
an(off_medium), mean(off_low)))
```

In Excel, I've paired data by location, and I've taken the two data points between which the border-to-border length of the axon changes the most. I'll consider the first of these time points "round" and the second "flat." I will load these data and perform a paired permutation test.

Load the data: Paired division measurements

Read this csv file to load the data.

Note that, for simplicity, I'm loading the "on" and "off" data into separate data frames, but these are the same data that are in the full data set, which is in df_paired.

```
df_paired = read.csv("G:/My Drive/DesktopItems/Measurements/Division/Div_Dynam_Paired.csv")
df_on = read.csv("G:/My Drive/DesktopItems/Measurements/Division/Div_Dynam_Paired-On.csv")
df_off = read.csv("G:/My Drive/DesktopItems/Measurements/Division/Div_Dynam_Paired-Off.csv")
```

Make longitudinal comparison.

Start by comparing the normalized caliber round to the normalized caliber flat. We can take the mean of these values, and we'll call this the OrigEffectSize.

```
OrigEffectSize_on = mean(df_on$cal_flat - df_on$cal_round)

print(OrigEffectSize_on)
```

```
## [1] -0.02557407
```

```
#Note that a negative value indicates that the axon becomes thinner when the basal cell flattens.

OrigEffectSize_off = mean(df_off$cal_flat - df_off$cal_round)

print(OrigEffectSize_off)
```

```
## [1] -0.001297297
```

Perform a paired null hypothesis permutation test.

For this test, we'll assume that there's actually no difference between the caliber when the basal cell is round vs. flat. We can simulate this by randomly flipping the round and flat values and calculating the test statistic (the mean change in fluctuation, as calculated above) for each simulated data set. These will all be stored. Then, we can calculate the p-value by seeing how many of our simulated values have a magnitude greater than what was actually observed. This tells us how likely it is that our data would have occurred by random chance.

```
#Duplicate the measurements data frame for the permutation test.
MeasurementsForPerm = df_on

#List all rows. (This should give you a list with an index for each sample.)
indices = 1:nrow(MeasurementsForPerm)

#Choose the number of times you want to resample the data.
numResamples = 1000

#Initialize the data frame.
TestStatisticDistribution = rep(NA,numResamples)

#Run through a bunch of imaginary data sets in which the samples have been assigned to round vs.
flat by random chance, like a coin flip. The test statistic will be calculated for each one and
stored to see the outcome of what random chance might look like, assuming that flat vs. round ar
en't actually different.
for (i in 1 : numResamples) {

  #Simulate chance:
  #Use random chance to decide which rows should be swapped.
  shouldFlipRows = runif( nrow(MeasurementsForPerm) ) > .5
  rowsToFlip = which (shouldFlipRows)
  #Temporarily store the round and flat calibers.
  tempround = MeasurementsForPerm[rowsToFlip, 4]
  tempflat = MeasurementsForPerm[rowsToFlip, 5]
  #Swap the round and flat measurements.
  MeasurementsForPerm[rowsToFlip, 5] = tempround
  MeasurementsForPerm[rowsToFlip, 4] = tempflat

  # calculate statistic
  TestStatisticDistribution[i] = mean(MeasurementsForPerm$cal_flat - MeasurementsForPerm$cal_rou
nd)

}

#Calculate the p-value.
pval <- (sum(TestStatisticDistribution >= -OrigEffectSize_on) +
           sum(TestStatisticDistribution <= OrigEffectSize_on) ) / (numResamples)

print('The p-value on the dividing cell is: ')
```

```
## [1] "The p-value on the dividing cell is: "
```

```
print(pval)
```

```
## [1] 0.137
```

Repeat for points off of the rounded cell.

```
#Duplicate the measurements data frame for the permutation test.
MeasurementsForPerm = df_off

#List all rows. (This should give you a list with an index for each sample.)
indices = 1:nrow(MeasurementsForPerm)

#Choose the number of times you want to resample the data.
numResamples = 1000

#Initialize the data frame.
TestStatisticDistribution = rep(NA,numResamples)


for (i in 1 : numResamples) {

  #Simulate chance:
  #Use random chance to decide which rows should be swapped.
  shouldFlipRows = runif( nrow(MeasurementsForPerm) ) > .5
  rowsToFlip = which (shouldFlipRows)
  #Temporarily store the round and flat calibers.
  tempround = MeasurementsForPerm[rowsToFlip, 4]
  tempflat = MeasurementsForPerm[rowsToFlip, 5]
  #Swap the round and flat measurements.
  MeasurementsForPerm[rowsToFlip, 5] = tempround
  MeasurementsForPerm[rowsToFlip, 4] = tempflat

  # calculate statistic
  TestStatisticDistribution[i] = mean(MeasurementsForPerm$cal_flat - MeasurementsForPerm$cal_rou
nd)

}

#Calculate the p-value.
pval <- (sum(TestStatisticDistribution >= -OrigEffectSize_off) +
           sum(TestStatisticDistribution <= OrigEffectSize_off) ) / (numResamples)

print('The p-value off the dividing cell is: ')
```

```
## [1] "The p-value off the dividing cell is: "
```

```
print(pval)
```

```
## [1] 0.925
```

There is only approx. 15% chance that the measurements observed on the dividing cell could have arisen by random chance, which is generally not considered statistically significant. That p-val is even a little higher than what I observed in my original pilot data set, which I based my power analysis on. The data off of the dividing cell are very likely to have occurred by random chance.

Load the data: Caliber on vs. off rounded cell

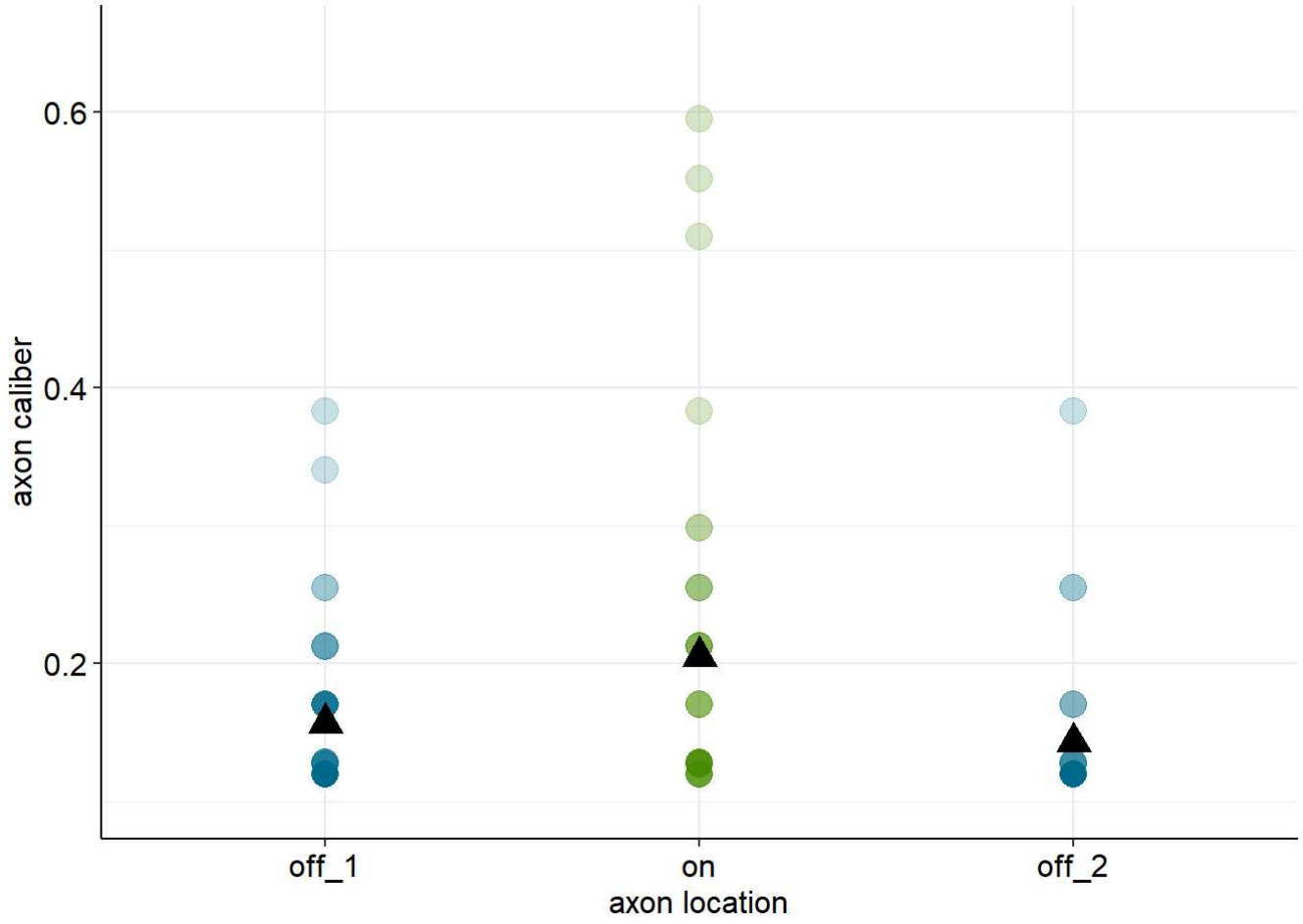
Read the csv file to load the data. This will give you the data as a data frame. (A data frame is list of a specific class, so typeof(measurements) will tell you that it's a list.)

Note that you might have to change the direction of the slashes for different operating systems.

```
df_round = read.csv("G:/My Drive/DesktopItems/Measurements/Division/Div_AxonOnRoundedCell_AllDat
aForR.csv")

df_round_stats = read.csv("G:/My Drive/DesktopItems/Measurements/Division/Div_AxononRoundedCell_
Stats.csv")
```

I already loaded tidyverse, I can make plots, just as I did above.

```
w = 0 #Change this to make the dots jitter side to side instead of being in a single column.
OffColor = "deepskyblue4"
OnColor = "chartreuse4"
level_order_round <- c("off_1", "on", "off_2")
colors_round <- c("off_1" = OffColor, "on" = OnColor, "off_2" = OffColor)
RangeToShow <- c(0.1, 0.65)

#Make plot using the measurements data frame.
ggplot(df_round, aes(y = caliber)) +
  scale_x_discrete(limits = level_order_round) +
  ylim(RangeToShow) +

  geom_point(aes(x = loc, color = loc, size = 3), alpha = 0.2, position = position_jitterdodge(j
itter.width = w), na.rm = FALSE) +

  scale_color_manual(values = colors_round, aesthetics = c("color", "fill")) +

  geom_point(data = df_round_stats, aes(x = loc, size = 3), shape = 17) +

  ylab("axon caliber") +
  xlab("axon location") +
  theme_bw() +
  theme(plot.background = element_blank(),
                      panel.border = element_blank(),
                      panel.background = element_blank(),
                      axis.line = element_line(color = "black"),
                      axis.title.x = element_text(size = 12, color = 'black'),
                      axis.title.y = element_text(size = 12, color = 'black'),
                      axis.text.x = element_text(size = 12, color = 'black'),
                      axis.text.y = element_text(size = 12, color = 'black'),
        legend.position = "none")
```



Compare on vs. off rounded cell.

We need to see if the data are normally distributed. Then, we can pick an appropriate test to compare each population of measurements.

```
with(df_round, shapiro.test(caliber[loc == "off_1"]))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  caliber[loc == "off_1"]
## W = 0.67461, p-value = 7.999e-09
```

```
with(df_round, shapiro.test(caliber[loc == "off_2"]))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  caliber[loc == "off_2"]
## W = 0.47403, p-value = 1.363e-09
```

```
with(df_round, shapiro.test(caliber[loc == "on"]))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  caliber[loc == "on"]
## W = 0.70229, p-value = 3.074e-07
```

```
#p-value greater than 0.05 means that the data are unlikely to differ from a normal distribution

#All three groups in the plot above have a very small p-value, meaning that they are not normall
y distributed.

#Because the data are not normally distributed, a Mann-Whitney test (aka an unpaired two-samples
Wilcoxon test) is recommended. I used this website as a reference: https://www.sthda.com/englis
h/wiki/unpaired-two-samples-wilcoxon-test-in-r

#First, I put each category in its own vector, just to simplify a little:
test_off1 <- with(df_round, c(caliber[loc == "off_1"]))
test_off2 <- with(df_round, c(caliber[loc == "off_2"]))
test_on <- with(df_round, c(caliber[loc == "on"]))

#Then, I can perform the Mann-Whitney test pairwise on these vectors:
wilcox.test(test_on, test_off1, exact = FALSE)
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  test_on and test_off1
## W = 1064, p-value = 0.02484
## alternative hypothesis: true location shift is not equal to 0
```

```
wilcox.test(test_on, test_off2, exact = FALSE)
```

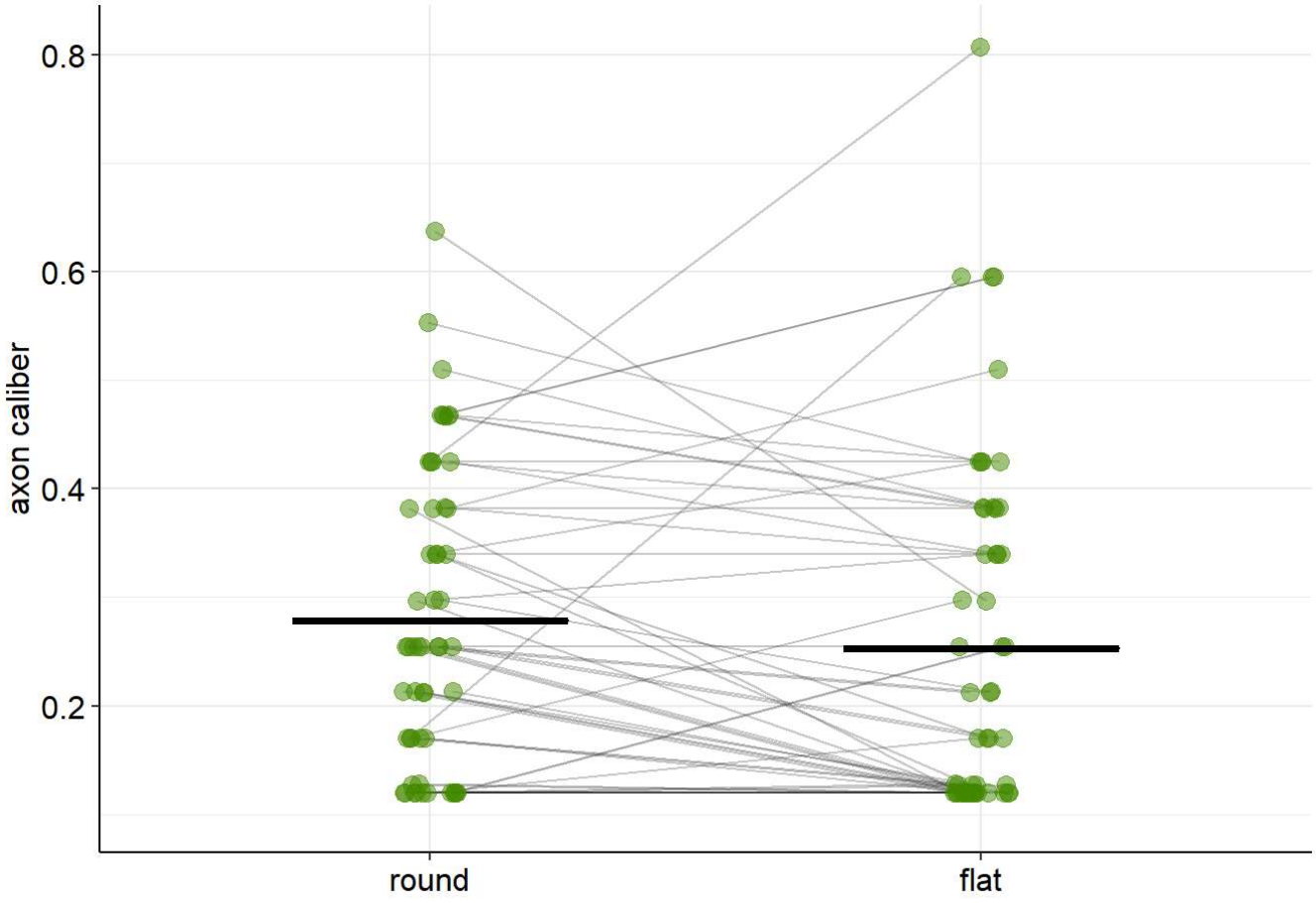
```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  test_on and test_off2
## W = 858, p-value = 0.0003472
## alternative hypothesis: true location shift is not equal to 0
```

```
wilcox.test(test_off1, test_off2, exact = FALSE)
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  test_off1 and test_off2
## W = 910, p-value = 0.05864
## alternative hypothesis: true location shift is not equal to 0
```

Plot paired data for axon caliber as the dividing cell flattens (used for paired statistics above)

```
#Make a data frame specifically for this plot that has all of the calibers for flat and round.
n = length(df_on$id)
df_onplot <- data.frame(id = as.character(c(df_on$id, df_on$id)),
                cal = as.numeric(c(df_on$cal_round, df_on$cal_flat)),
                div = as.factor(c(rep("round", n), rep("flat", n))))
#Add a unique identification for each measurement's location.
id_un <- paste(df_on$id, df_on$loc.un, sep = "-")
df_onplot$id_un <- id_un

#Choose order shown on plot
level_order_paired <- c("round", "flat")
#Choose range shown on plot
RangeToShow <- c(0.1, 0.81)

OnColor = "chartreuse4"

#Make plot using this data frame.
ggplot(df_onplot, aes(y = cal)) +
  scale_x_discrete(limits = level_order_paired) +

  geom_line(aes(x = div, group = id_un), color = "black",  position=position_dodge(width = 0.1),
alpha = 0.2) +
  geom_point(aes(x = div, group = id_un), size = 3, color = OnColor, alpha = 0.5, position=posit
ion_dodge(width = 0.1)) +

  geom_crossbar(data = data.frame(df_on$cal_round), aes(x = "round", ymin=mean(df_on$cal_round),
ymax=mean(df_on$cal_round), y=mean(df_on$cal_round)), width=0.5, color="black") +
  geom_crossbar(data = data.frame(df_on$cal_flat), aes(x = "flat", ymin=mean(df_on$cal_flat), ym
ax=mean(df_on$cal_flat), y=mean(df_on$cal_flat)), width=0.5, color="black") +

  ylim(RangeToShow) +
  ylab("axon caliber") +
  xlab("") +
  theme_bw() +
  theme(plot.background = element_blank(),
                      panel.border = element_blank(),
                      panel.background = element_blank(),
                      axis.line = element_line(color = "black"),
                      axis.title.y = element_text(size = 12, color = 'black'),
                      axis.text.x = element_text(size = 12, color = 'black'),
                      axis.text.y = element_text(size = 12, color = 'black'),
                      legend.position = "None")
```



Check stats for axon on round vs. flat basal cell

```
#First, I put each category in its own vector, just to simplify a little:
test_round <- with(df_onplot, c(cal[div == "round"]))
test_flat <- with(df_onplot, c(cal[div == "flat"]))

#I want to know the mean and size of each group:
mean(test_round)
```

```
## [1] 0.2786852
```

```
length(test_round)
```

```
## [1] 54
```

```
mean(test_flat)
```

```
## [1] 0.2531111
```

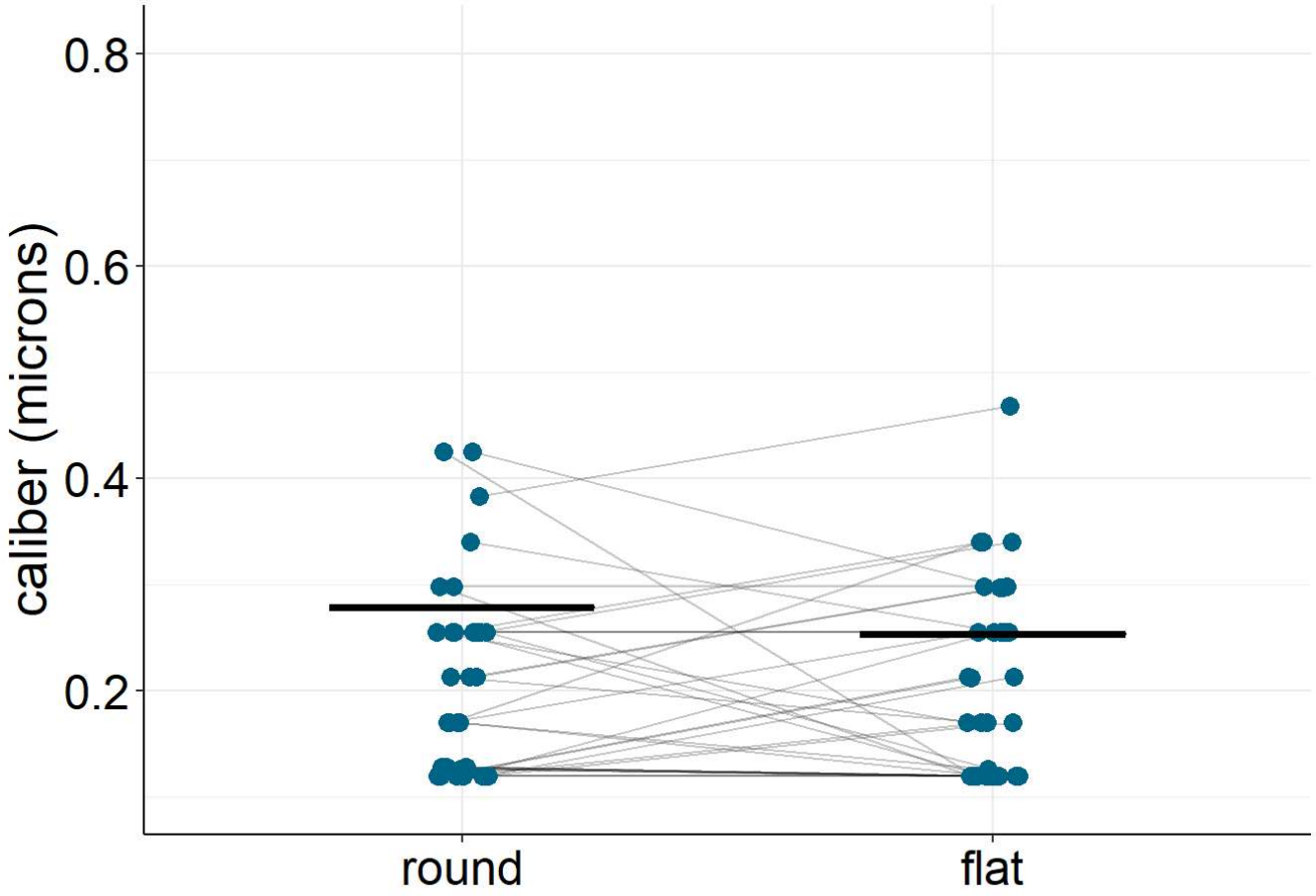
```
length(test_flat)
```

```
## [1] 54
```

```
#Note that the paired permutation test was carried out above for these data.
```

```
#Make a data frame specifically for this plot that has all of the calibers for flat and round.
n = length(df_off$id)
df_offplot <- data.frame(id = as.character(c(df_off$id, df_off$id)),
                cal = as.numeric(c(df_off$cal_round, df_off$cal_flat)),
                div = as.factor(c(rep("round", n), rep("flat", n))))
#Add a unique identification for each measurement's location.
id_un <- paste(df_off$id, df_off$loc.un, sep = "-")
df_offplot$id_un <- id_un

#Choose order shown on plot
level_order_paired <- c("round", "flat")

#Make plot using this data frame.
ggplot(df_offplot, aes(y = cal)) +
  scale_x_discrete(limits = level_order_paired) +

  geom_line(aes(x = div, group = id_un), color = "black", alpha = 0.2, position=position_dodge(w
idth = 0.1), alpha = 1/4) +
  geom_point(aes(x = div, group = id_un), color = "deepskyblue4", size = 3, position=position_do
dge(width = 0.1)) +

  geom_crossbar(data = data.frame(df_on$cal_round), aes(x = "round", ymin=mean(df_on$cal_round),
ymax=mean(df_on$cal_round), y=mean(df_on$cal_round)), width=0.5, color="black") +
  geom_crossbar(data = data.frame(df_on$cal_flat), aes(x = "flat", ymin=mean(df_on$cal_flat), ym
ax=mean(df_on$cal_flat), y=mean(df_on$cal_flat)), width=0.5, color="black") +

  ylim(RangeToShow) +
  ylab("caliber (microns)") +
  xlab("") +
  theme_bw() +
  theme(plot.background = element_blank(),
                      panel.border = element_blank(),
                      panel.background = element_blank(),
                      axis.line = element_line(color = "black"),
                      axis.title.y = element_text(size = 20, color = 'black'),
                      axis.text.x = element_text(size = 18, color = 'black'),
                      axis.text.y = element_text(size = 18, color = 'black'),
                      legend.position = "None")
```

```
## Warning: Duplicated aesthetics after name standardisation: alpha
```



A caveat to these data is that the number of measurements varies between cells. This means that some movies (e.g. one with worse image quality) might impact the analysis more or less than others. I just wasn't sure how to choose exactly the same number of measurements without adding bias into the scoring.

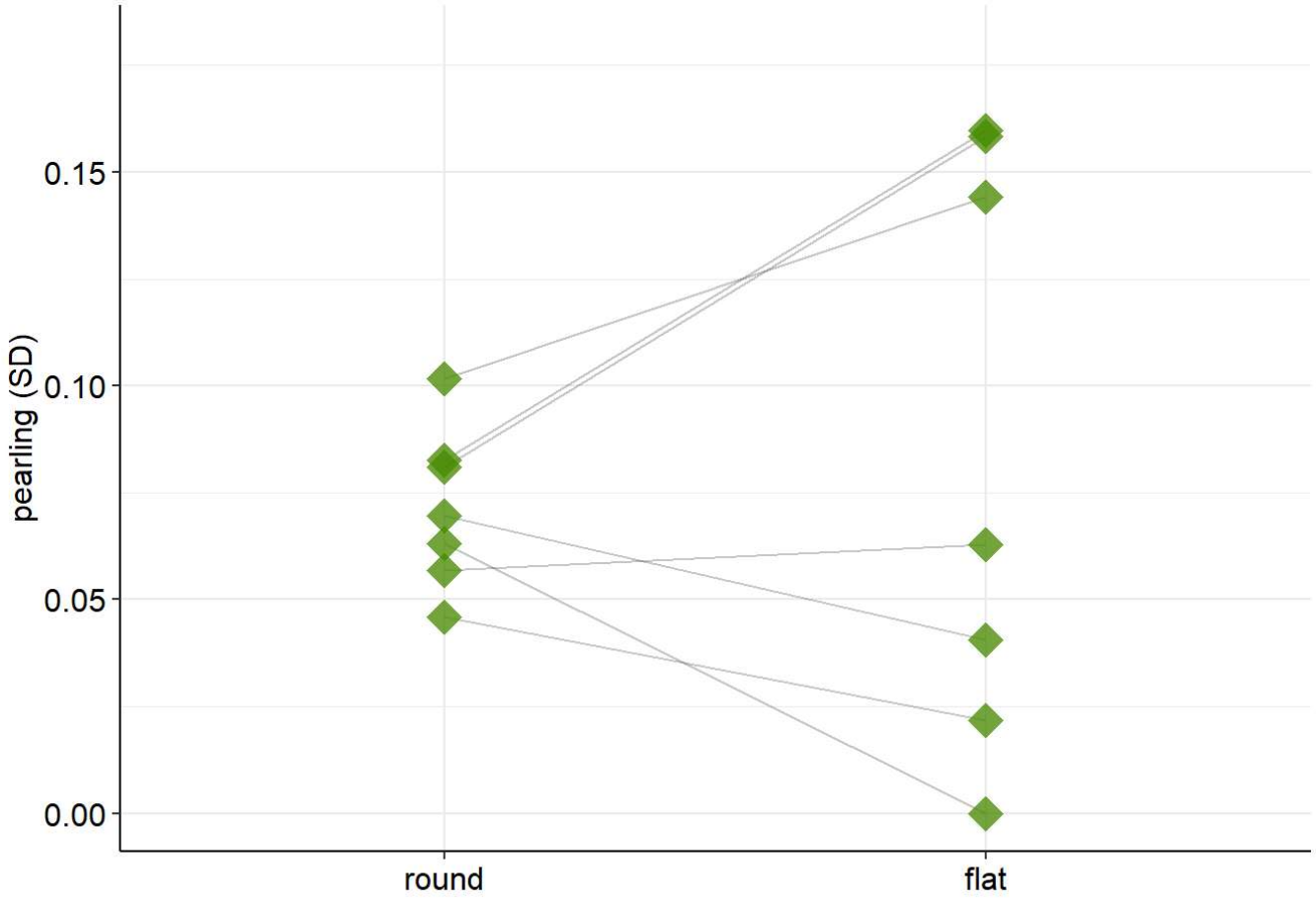
Finally, the biggest difference between round and flat seems to be that the largest measurements (i.e. the varicosities/pearls/bubbles) seem to get smaller, whereas the thin locations remain thin (maybe real, maybe just limited by resolution). That might suggest that the mean caliber won't change much, but the SD will.

Can we compare the SD across the axon when the call is round vs. flat?

Compare SD as the basal cells flatten.

Standard deviation for each cell was calculated in Excel using STDEV.S for calibers on the rounded cell or its daughters. Let's plot these and see if calibers become less variable as the cell flattens.

```
#The standard deviation for each location on each cell was calculated in Excel using STDEV.S. Lo
ad these data here.
df_on_SD <- read.csv("G:/My Drive/DesktopItems/Measurements/Division/Div_Dynam_Paired-On_SD.cs
v")

#Make the data frame usable for plotting.
n = length(df_on_SD$id)
df_onplot_SD <- data.frame(id = as.character(c(df_on_SD$id, df_on_SD$id)),
                sd = as.numeric(c(df_on_SD$sd_round, df_on_SD$sd_flat)),
                div = as.factor(c(rep("round", n), rep("flat", n))))

#Define some values for plotting.
d = 0.5
level_order_onsd <- c("round", "flat")
OnColor = "chartreuse4"
RangeToShow <- c(0.0, 0.18)

#Plot these SD values, connecting each cell's SDs with lines
ggplot(df_onplot_SD, aes(y = sd)) +
  scale_x_discrete(limits = level_order_onsd) +
  ylim(RangeToShow) +

  geom_line(aes(x = div, group = id), size = d, alpha = 0.2, color = "black") +
  geom_point(aes(x = div), shape = 18, size = 6, color = OnColor, alpha = 0.75, na.rm = FALSE) +

  ylab("pearling (SD)") +
  xlab("") +
  theme_bw() +
  theme(plot.background = element_blank(),
                    panel.border = element_blank(),
                    panel.background = element_blank(),
                    axis.line = element_line(color = "black"),
                    axis.title.y = element_text(size = 12, color = 'black'),
                    axis.text.x = element_text(size = 12, color = 'black'),
                    axis.text.y = element_text(size = 12, color = 'black'),
        legend.position = "none")
```



(end)