Analyzing and visualizing fiber photometry data with fluoR

Andrew Tamalunas

Last updated 2020-12-08

Contents

Chapter 1

Introduction

Analyzing and visualizing fiber photometry data with fluoR is a continuously updated book of tutorials and background information for analyzing and visualizing time series data from behavioral experiments.

1.1 Why use fluoR?

The `fluoR` R package is the successor to the `GCalcium` package, which I initially wrote to help ensure that my fiber photometry data analyses were accurate, consistent, and transparent. Both R packages and this continuously-updated document were publicly released to save other researchers the time and uncertainty in working with their own data, along with helping keep my knowledge fresh. I will mostly avoid discussion on biological topics for now.

Calcium imaging technologies such as GCaMP produce massive datasets. With repeated trials for each subject, researchers can end up with tens of thousands of data points per subject. Complicating matters further, there seems to be no gold standard for working with and analyzing this type of data.

fluoR...

- is open-source and transparent
- has its own user manual
- is free to use

1.2 Who is fluoR for?

Some examples of researchers who may find fluoR useful are those who:

- have recorded multiple trials of time series waveform data
- want to look at their recorded data with minimal coding while still knowing how their code works
- want to test GCaMP data before using the hardware (see `GCaMP` sample dataset)

1.3 Background

1.3.1 pre-recording

I initially wrote the fluoR package (then named GCalcium) for a pilot study by Dr. Michael Baratta. After the main investigators set up the GCaMP6m fiber photometry technology, neural activity was continuously recorded in the infralimbic region (IL) of the ventromedial prefrontal cortex (mPFCv) during escapable tail shock - a "controllable" stressor. We wanted to examine if post-trial neural activity in the rat IL changes after repeated exposure to a controllable stressor.

There were two main issues that resulted in us using GCaMP for measuring neural activity:

1. electrophysiology is difficult to conduct when using tail shock as a stressor
2. fos examination is limited temporally

The included `GCaMP` dataset is a sample of 10 random trials from this pilot study.

1.3.2 recording

For recording, we used fiber photometry equipment from Tucker-Davis Technologies, which was set up by Dr. David Root.

1.3.3 post-recording

Dr. Root also provided us with a modified version of a Matlab script to load and pre-process the data [4]. Ultimately, I decided to export the data for use in another program after the signal averaging step.

There were a couple of Python add-ons that looked promising. Unfortunately, none of them seemed to discuss how they worked or why the authors took the steps that they did. Using these would make me at odds with my philosophy of "don't run analyses if you don't know how they work". I decided that R, which I had a much more solid background in, was the answer.

While R is generally slower than Matlab, the syntax is much simpler and it is easier to keep track of the steps you took along the way. Additionally, there are a handful of R packages (e.g. ggplot2, plotly) that allow for publication-ready graphs.

Did I mention R is free?

Chapter 2

Setting up fluoR

2.1 Installation

Currently, fluoR can only be installed from Github.

```r
### Install from Github
devtools::install_github('atamalu/fluoR', build_vignettes = TRUE)
```

2.2 Convert to fluoR format

After exporting your recorded data from your preferred software (e.g. MATLAB, Doric software), the first step is to convert your data for quick use with the fluoR package. The `format_data` function is used for this.

There are two formats of data supported as input:

1. Vertical format

- timestamps in the first column
- one column for each trial's recorded values

2. Horizontal format

- timestamps in the first row
- one row for each trial's recorded values

2.3 Input/Output

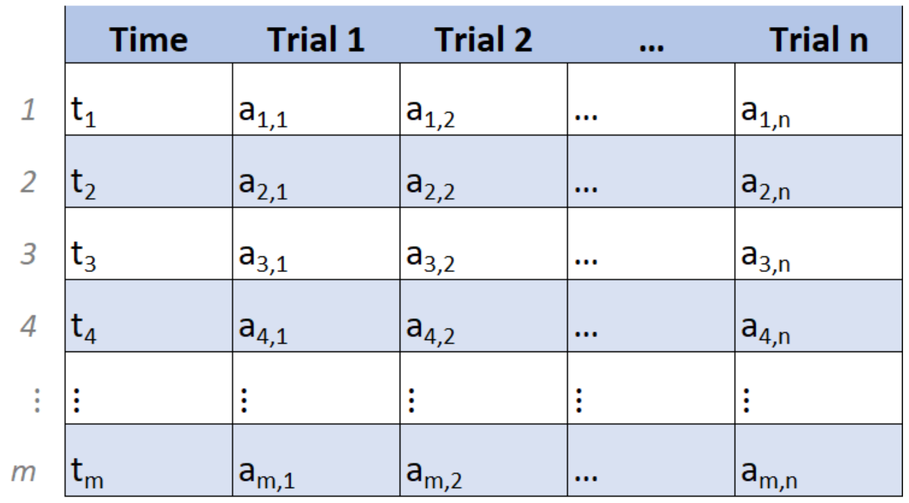
Input

The input for `format_data` can be a matrix or data frame - labeled or unlabeled. The function detects whether the number of rows are greater than the number of columns (vertical) and vice versa (horizontal).

Output

If the input is horizontal, then the object is transposed to vertical format. The table is then converted to a data frame, with each column being labeled. This makes manually working with the data more convenient and R-friendly.

The below table is an informal matrix representation of what the returned data frame will look like.



	Time	Trial 1	Trial 2	...	Trial n
1	t_1	$a_{1,1}$	$a_{1,2}$	...	$a_{1,n}$
2	t_2	$a_{2,1}$	$a_{2,2}$	...	$a_{2,n}$
3	t_3	$a_{3,1}$	$a_{3,2}$	...	$a_{3,n}$
4	t_4	$a_{4,1}$	$a_{4,2}$	...	$a_{4,n}$
⋮	⋮	⋮	⋮	⋮	⋮
m	t_m	$a_{m,1}$	$a_{m,2}$	...	$a_{m,n}$

2.4 Examples

2.4.1 Format data

Format the `GCaMP` dataset included with the fluoR package so the data is easier to work with.

```
library(fluoR)

df <- format_data(GCaMP)
head(df)
```

```
##      Time Trial1 Trial2 Trial3 Trial4 Trial5 Trial6 Trial7 Trial8 Trial9
## 1 -3.9902 82.689 82.858 81.709 89.747 90.788 90.365 123.53 120.39 119.95
## 2 -3.9803 82.656 82.922 81.702 89.637 90.847 90.418 123.30 120.41 119.82
## 3 -3.9705 82.650 82.984 81.713 89.545 90.927 90.485 123.13 120.45 119.73
## 4 -3.9607 82.671 83.042 81.726 89.476 91.003 90.578 123.02 120.52 119.66
## 5 -3.9508 82.712 83.085 81.732 89.428 91.056 90.696 122.96 120.59 119.62
## 6 -3.9410 82.757 83.116 81.714 89.397 91.080 90.833 122.95 120.66 119.60
##   Trial10
## 1  116.94
## 2  116.97
## 3  117.00
## 4  117.01
## 5  117.00
## 6  116.97
```

Chapter 3

Standardizing Data

```r
library(fluoR)
df <- format_data(GCaMP)
```

3.1 Reasons to standardize data

There are a few reasons to standardize your data before exploring your data.

1. Signal differs between subjects

   - Regardless of the specific technologies used, there is almost always differences in signal strength for each subject

2. Signal differs between trials

   - The strength of recording signal tends to decay over time

3. Utilizing baseline values

   - Using transformations such as percent change allows you to center the data at an objective value
   - After centering your trial and post-trial data, the data is interpreted as relative to baseline values
   - The baseline period is typically assumed to be a “resting” period prior to exposure to the experimental manipulation. This means that using standardization methods (particularly z-scores) also takes baseline deviations into consideration.

3.2 Methods of standardization

A little alteration in how we compute z-scores can make a significant difference.

3.2.1 z-scores

Standard z-score transformations work the same way with time series data as with any other. The formula:

1. centers every value (x) at the mean of the full time series (mu)
2. divides it by the standard deviation of the full time series (sigma).

$$z_i = \frac{x_i - \mu}{\sigma}$$

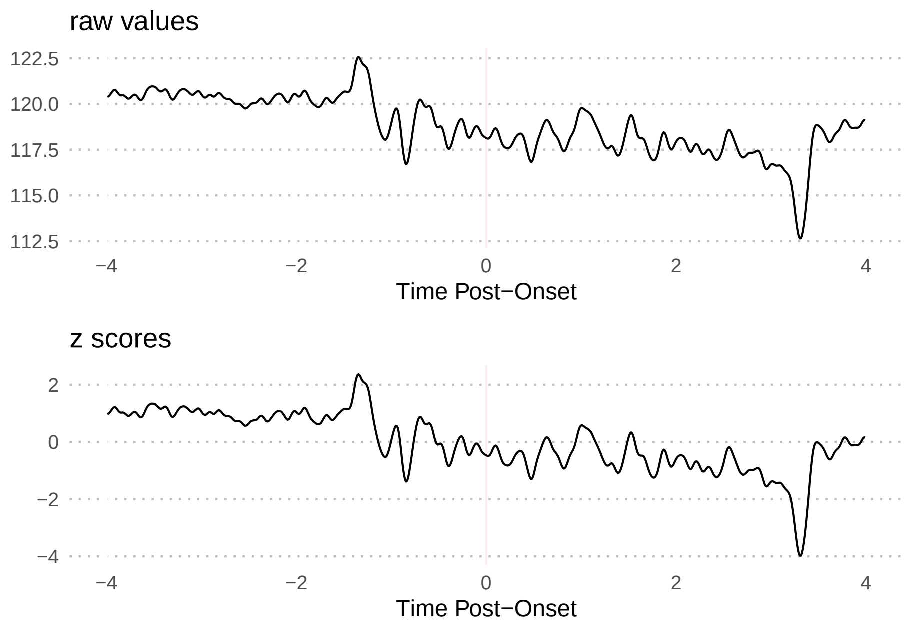
$$\begin{aligned} &\text{where...} \\ &\qquad \mu = \text{mean of population}, \\ &\qquad \sigma = \text{standard deviation of population} \end{aligned}$$

This results in the same time series in terms of standard deviations from the mean, all in the context of the full time series.

3.2.1.1 R Code

```r
z.scores <- z_score(xvals = df$Trial8,
                    mu = mean(df$Trial8), # manual input of mu/sigma optional;
                    sigma = sd(df$Trial8)) # used for example purposes
```

3.2.1.2 Visualization



3.2.2 baseline z-scores

Using the pre-event baseline period as the input values for computing z-scores can be useful in revealing changes in neural activity that you may not find by just comparing pre-trial and trial periods. This is in part because baseline periods tend to have relatively low variability.

As you can see from the formula, a lower standard deviation will increase positive values and decrease negative values - thus making changes in neural activity more apparent.

$$baseline\ z_i = \frac{x_i - \bar{x}_{baseline}}{s_{baseline}}$$

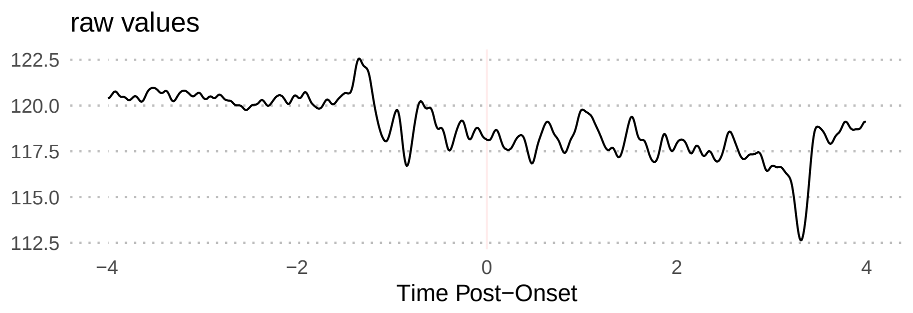
where...
$\bar{x}_{baseline}$ = mean of values from baseline period,
$s_{baseline}$ = standard deviation of values from baseline period

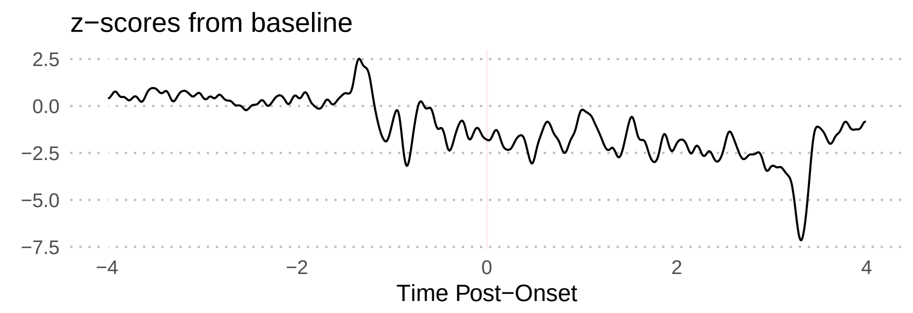
This results in a time series interpreted in terms of standard deviations and mean during the baseline period. Baseline z-scores are conceptually justifiable because the standard deviation is then the number of deviations from the mean when a subject is at rest. The values outside of the baseline period will be different using this version, but not within the baseline period.

3.2.2.1 R Code

```r
### Extract baseline values
baseline.vals <- df$Trial8[df$Time >= -4 & df$Time <= 0]

### Compute z-scores
z.scores.baseline <- z_score(xvals = df$Trial8,
                             mu = mean(baseline.vals),
                             sigma = sd(baseline.vals))
```

3.2.2.2 Visualization





3.2.3 modified z scores

Waveform data fluctuates naturally. But in the event of a change in activity due to external stimuli, signal variation tends to rapidly increase and/or decrease and becomes unruly.

$$modified\ z_i = \frac{0.6745(x_i - \tilde{x})}{MAD}$$

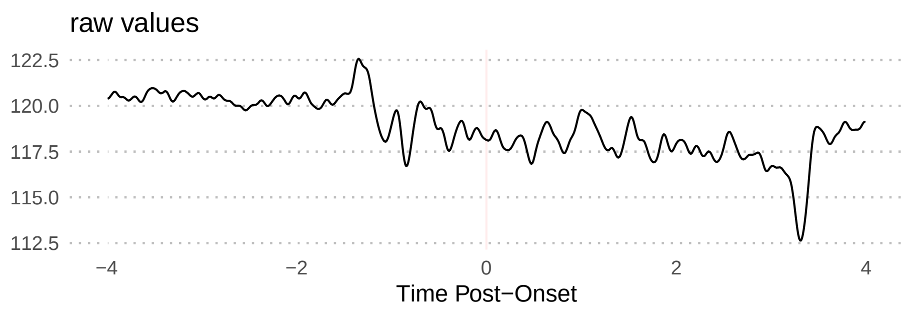
where...

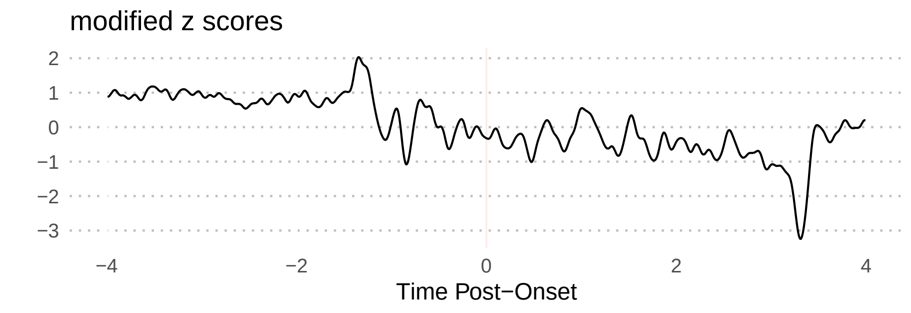
$$\tilde{x} = \text{sample median,}$$
$$MAD = \text{median absolute deviation}$$

3.2.3.1 R Code

```
z.scores.modified <- z_score(xvals = df$Trial8,
                             z.type = 'modified')
```

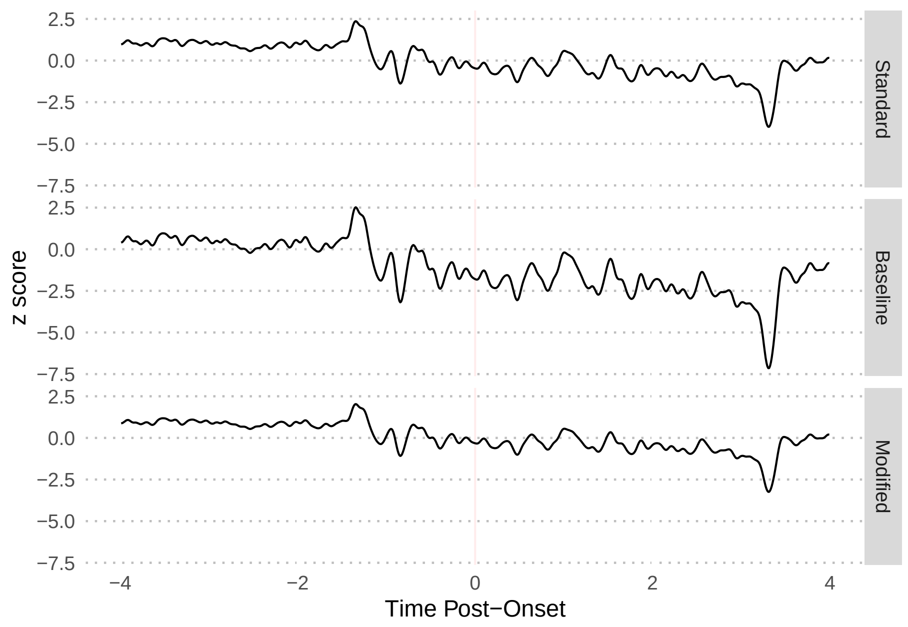
3.2.3.2 Visualization



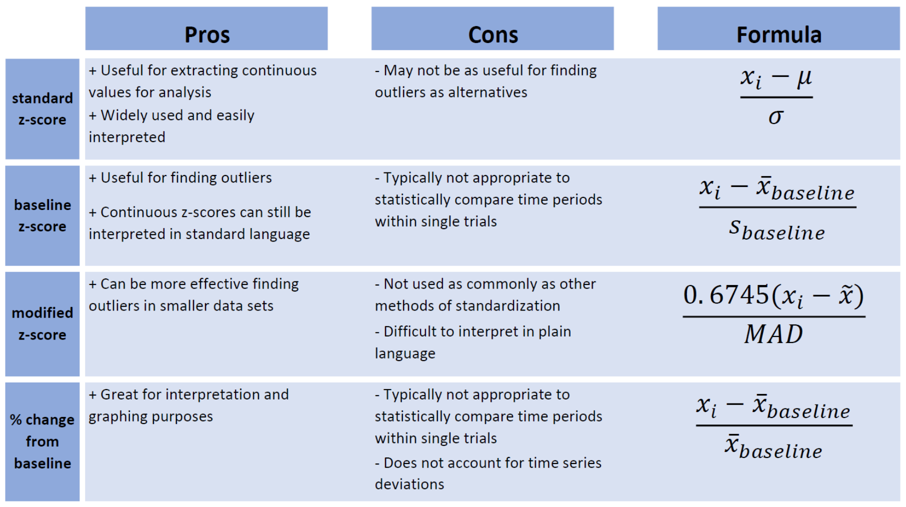


3.3 z-score comparison

3.3.1 Visualization



3.3.2 Summary table



	Pros	Cons	Formula
standard z-score	+ Useful for extracting continuous values for analysis + Widely used and easily interpreted	- May not be as useful for finding outliers as alternatives	$\frac{x_i - \mu}{\sigma}$
baseline z-score	+ Useful for finding outliers + Continuous z-scores can still be interpreted in standard language	- Typically not appropriate to statistically compare time periods within single trials	$\frac{x_i - \bar{x}_{baseline}}{s_{baseline}}$
modified z-score	+ Can be more effective finding outliers in smaller data sets	- Not used as commonly as other methods of standardization - Difficult to interpret in plain language	$\frac{0.6745(x_i - \tilde{x})}{MAD}$
% change from baseline	+ Great for interpretation and graphing purposes	- Typically not appropriate to statistically compare time periods within single trials - Does not account for time series deviations	$\frac{x_i - \bar{x}_{baseline}}{\bar{x}_{baseline}}$

3.4 Examples

3.4.1 Example 1

Standardize trial 8 so that the units are in terms of standard deviations from the mean of the full time series.

```
z_score(xvals = df$Trial8,
        z.type = 'standard')
```

3.4.2 Example 2

Standardize trial 8 so that the units are in terms of standard deviations from the mean of the pre-event period.

We can do this manually for each trial.

```
### Manual
baseline.vals <- df$Trial8[df$Time >= -4 & df$Time <= 0] # extract baseline values

baseline.z <- z_score(xvals = df$Trial8,
                      mu = mean(baseline.vals), # mean of baseline
                      sigma = sd(baseline.vals), # sd of baseline
                      z.type = 'standard')
```

Or we can use the `baseline_transform` wrapper.

```
baseline.zdf <- baseline_transform(dataframe = df,
                                   trials = 8,
                                   baseline.times = c(-4, 0),
                                   type = 'z_standard')
```

Both methods will result in the same values.

```
all(baseline.zdf$Trial8 - baseline.z == 0)
```

```
## [1] TRUE
```

3.4.3 Example 3

Standardize trial 8 so that the units are in terms of deviations from the median of the full time series.

```
z_score(xvals = df$Trial8,
        z.type = 'modified')
```

3.4.4 Example 4

Convert trial 8 so that the units are in terms of percent change from the mean of the pre-event period.

We can do this manually for each trial.

```
### Manual
baseline.vals <- df$Trial8[df$Time >= -4 & df$Time <= 0] # extract baseline values

perc.change <- percent_change(xvals = df$Trial8,
                              base.val = mean(baseline.vals))
```

Or we can use the `baseline_transform` wrapper.

```
perc.changedf <- baseline_transform(dataframe = df,
                                    trials = 8,
                                    baseline.times = c(-4, 0),
                                    type = 'percent_change')
```

Both methods will result in the same values.

```
all(perc.changedf$Trial8 - perc.change == 0)
```

```
## [1] TRUE
```

Chapter 4

Visualization

```r
library(fluoR)
df <- format_data(GCaMP)
```

4.1 Multiple trials

The `plot_trials` command uses R's base plotting functions for data exploration purposes. Some benefits of using this function are:

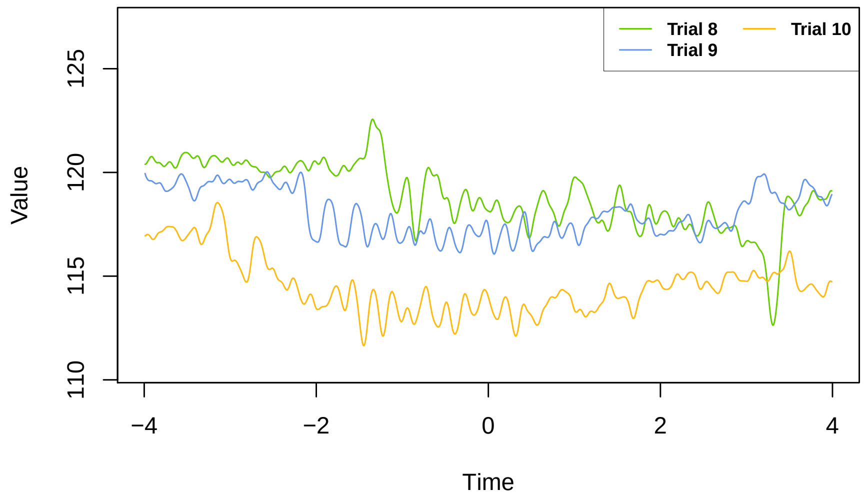
1. Much faster graphing than alternatives (e.g. ggplot2, plotly)
2. Automatic scaling of x- and y- axes for graphing multiple trials so that all data points fit in the plot boundaries
3. Automatic color coding and trial number labeling
4. Plotting multiple trials with a single command while using default fluoR data frame format

The more you need to graph your data in R, the more clear the tradeoff between `plot_trials` and `ggplot` becomes.

- `plot_trials` will save significant time from extra data manipulation when looking through your data, especially when examining a large number of trials
- `ggplot` will be much more appropriate for publication and presentation graphing

4.1.1 plot_trials

```r
base.trials <- plot_trials(dataframe = df,
                           trials = 8:10)
```



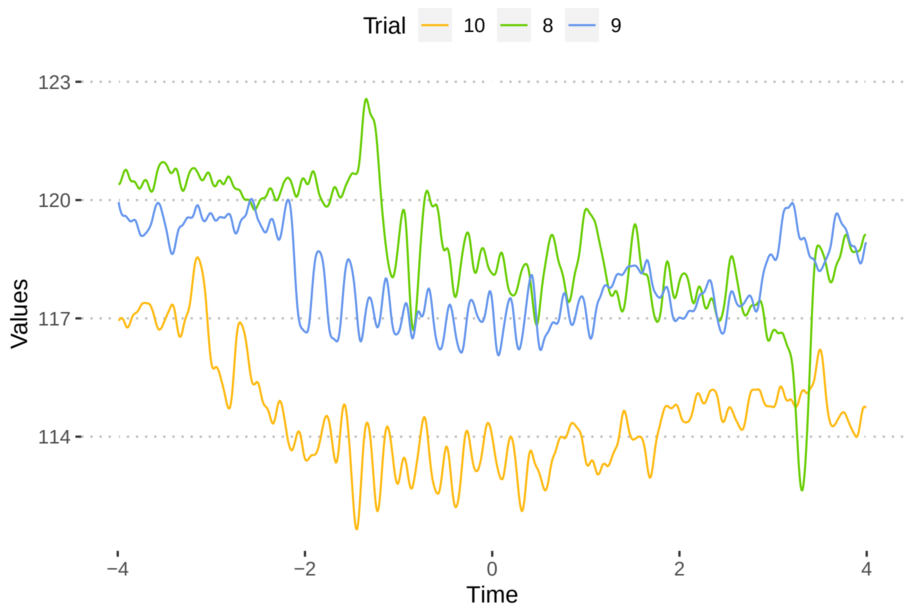
4.1.2 ggplot2

For plotting multiple trials using ggplot2, we will need the data frame to be in "long" format. So, we stack the trials we want to plot.

```r
df.long <- data.frame(
  Time = rep(df$Time, times = 3), # repeat time values by number of trials
  Values = c(df$Trial8, df$Trial9, df$Trial10), # vector of trial values
  Trial = c(rep("8", length(df$Trial8)), # label trial numbers
            rep("9", length(df$Trial9)),
            rep("10", length(df$Trial10)))
)
```

Now we can make the graph using ggplot2.

```r
library(ggplot2)
library(ggpubr) # for theme

ggplot(df.long) +
  geom_line(aes(x = Time, y = Values,
                color = Trial)) +
  scale_color_manual(values = c("8" = 'chartreuse3',
                                "9" = 'cornflowerblue',
                                "10" = 'darkgoldenrod1')) +
  theme_pubclean()
```



4.2 Trial ranges

Let's say that you have 30 trials. As part of your hypothesis, you believe that neural activity will change shortly after exposure to a stimulus during trials 1-10, but not subsequent ones.

One method of examining this could be:

1. Standardize your data so it accounts for within-subject and between-trial differences
2. Break your data into "blocks" of 10 and collapse the data across trials for each time point using a summary statistic such as the mean or median

3. Plot the values using 3 groups: trials 1-10, trials 11-20, and trials 21-30

Unfortunately, a 30+ trial dataset would not be ideal to include in the fluoR package. So, we will use trial "blocks" of 3, which results in the groups being trials 1-3, 4-6, and 7-9.

For the first part, we standardize the data into z-scores from baseline.

```r
### 1. Standardize data
df.stand <- baseline_transform(dataframe = df,
                               trials = 1:10,
                               baseline.times = c(-4,0),
                               type = 'z_standard')
```

For the second part, we find the median value for each time point from each of the 3 trial blocks.

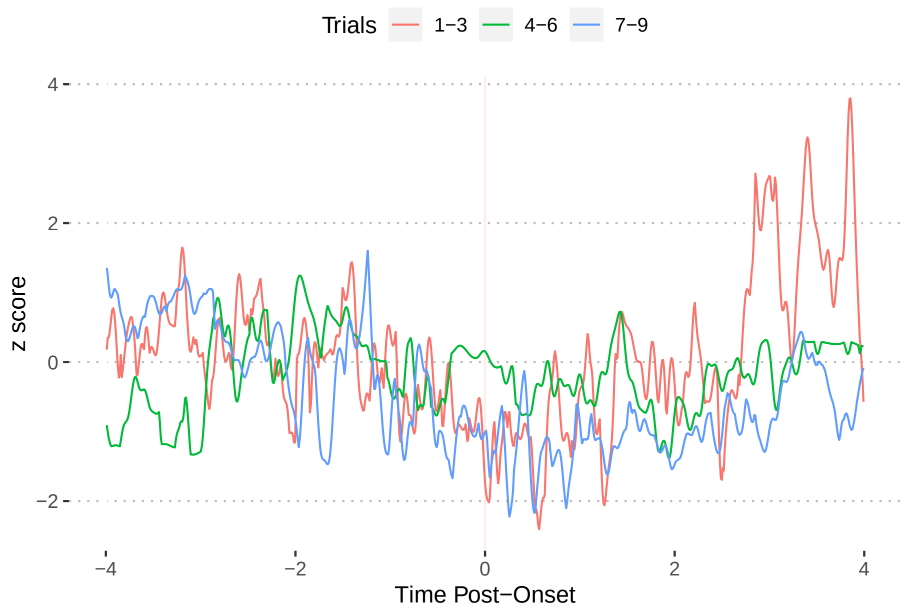
```r
### 2. Summarize/collapse across blocks of trials
df.block1 <- summarize_trials(dataframe = df.stand, trials = 1:3,
                              summary.type = 'median')
df.block2 <- summarize_trials(dataframe = df.stand, trials = 4:6,
                              summary.type = 'median')
df.block3 <- summarize_trials(dataframe = df.stand, trials = 7:9,
                              summary.type = 'median')
```

Graphing summarize data tends to be more for publication than exploration purposes, so we want to use ggplot2. Fortunately, creating multiple data frames `summarize_trials` allows us to simply assign a label column to each data frame and combine them before graphing.

```r
df.block1$trial.range <- '1-3'
df.block2$trial.range <- '4-6'
df.block3$trial.range <- '7-9'

df.blocked <- rbind(df.block1, df.block2, df.block3)
```

Last, we plot the summarized values.

```r
ggplot(df.blocked) +
  geom_vline(xintercept = 0,
             color = 'red', alpha = 0.075) + # event onset at 0
  geom_line(aes(x = Time, y = summ.trials,
                color = trial.range)) + # trial lines
  labs(
    x = 'Time Post-Onset',
    y = 'z score',
    color = 'Trials'
  ) +
  theme_pubclean()
```



4.3 Smoothing

As you can see by the previous graphs, the recorded data looks sharp when graphed across time points. While this should be used for the data extraction and analysis phase, we do not have this limitation when graphing.

The idea is to keep your data as similar as possible to the original time series while making an appealing visualization. Time series filters can help with this.

4.3.1 Whittaker filter

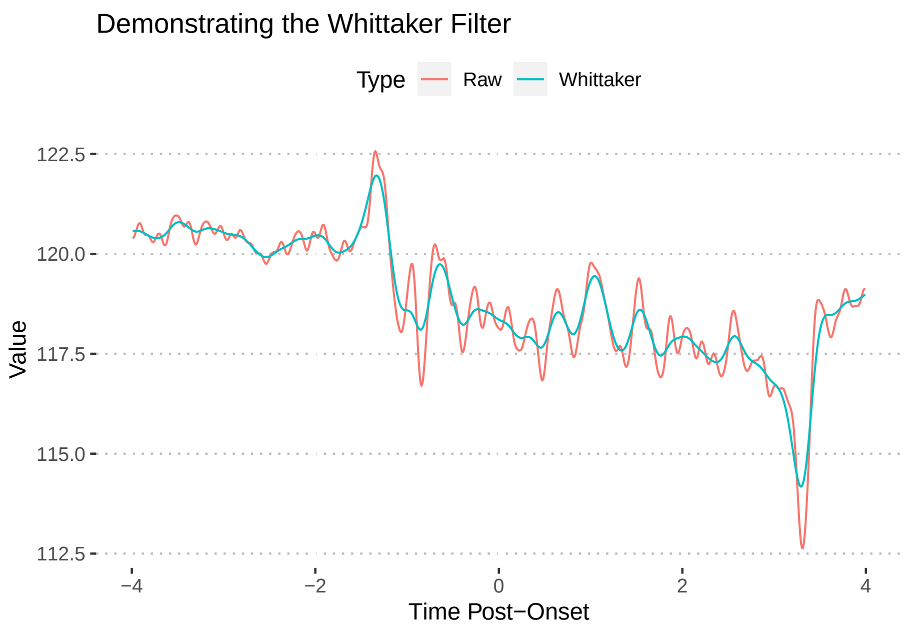
The Whittaker filter is a smoother that fits a curve to the data and penalizes highly-deviating points using the penalized least squares (PLS) method. There are two important parameters for this formula:

- lambda - smoothing parameter that controls the amount of curvature allowed for the least-squares fit. A smaller lambda permits more curvature

- d - order of differences for the least-squares penalty

The `pracma` R package includes an implementation of the Whittaker smoother [1].

```
library(pracma)

Trial8.whittaker <- whittaker(y = df$Trial8)
```



As shown by the above figure, the Whittaker filter produces a much more “tame” line. In turn, the line makes extreme values (peaks and valleys) smaller. This filter tends to be useful when trying to smooth data without flattening too many curves.

4.3.2 Generalized additive modeling

I have only found generalized additive modeling useful as a smoother or for predictive modeling. This is because GAM’s tend to overfit the model to the data, which is a big no-no in hypothesis testing.

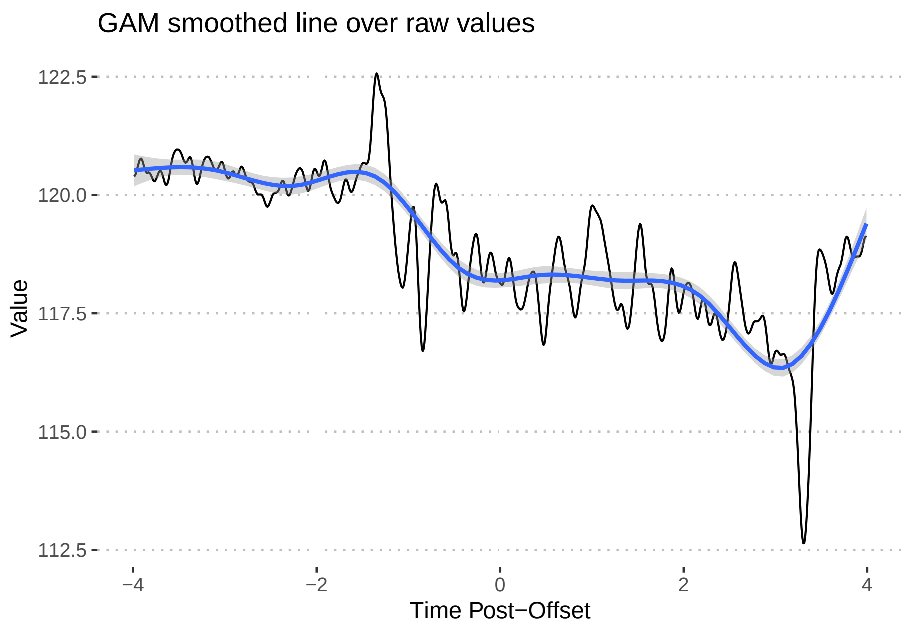
The `mgcv` R package includes a popular implementation of GAM’s. The formula input for the package’s `gam` function is styled like R’s base `glm` function. To smooth our time series data, we use Time as the dependent variable and our observation values as the independent variable.

```r
### Compute gam
library(mgcv)

### Construct model
gam1 <- gam(Time ~ s(Trial8, bs = 'cs'), data = df) # construct model
gamfit <- predict(gam1, data = df) # use model to create fitted line
```

There are also a handful of other parameters found in the documentation that can be changed, but I typically avoid. If you don't need to change the additional parameters, `ggplot2` has a command that smooths the data automatically and scales it to your original data points.

```r
ggplot(df, aes(x = Time, y = Trial8)) +
  geom_line() +
  stat_smooth(method = 'gam') +
  labs(
    x = 'Time Post-Offset',
    y = 'Value',
    title = 'GAM smoothed line over raw values'
    ) +
  theme_pubclean()
```

```
## `geom_smooth()` using formula 'y ~ s(x, bs = "cs")'
```



In this situation, the GAM captures the general direction of the data points but almost completely removes smaller peaks/valleys and flattens extreme values.

4.3.3 Loess

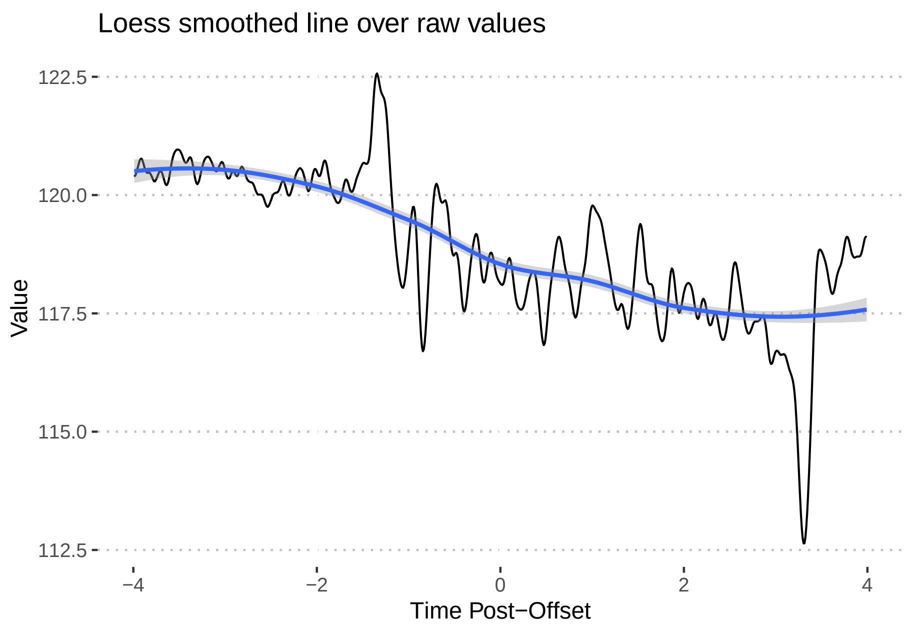
4.3.3.1 model

Locally estimated scatterplot smoothing (loess) is a non-parametric regression that uses multiple regression in k-nearest-neighbor meta-models.

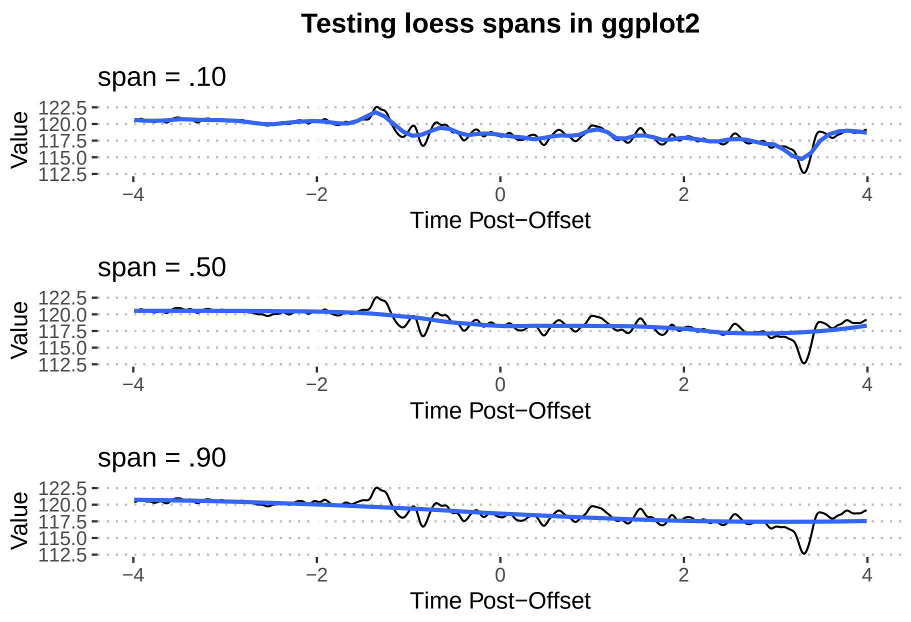
Loess models require much more data than standard regression. Luckily, this makes loess a solid choice for smoothing fiber photometry data. Much less so for testing hypotheses.

The `stats` package includes the `loess` function that allows us to implement it in typical R regression form. However, researchers will seldom find use for loess as a model itself.

```r
ggplot(df, aes(x = Time, y = Trial8)) +
  geom_line() +
  stat_smooth(method = 'loess') +
  labs(
    x = 'Time Post-Offset',
    y = 'Value',
    title = 'Loess smoothed line over raw values'
    ) +
  theme_pubclean()
```



Additionally, you can change the `span` parameter to control the degree of smoothing.



4.4 Further reading

Programming

- UCLA's statistics consulting released a guide to working with time series smoothing using the `ggplot2` package [2]

Math

- [5] offer examples of time series smoothing using the `mgcv` package. The authors also go very in-depth with the math behind the methods.

Chapter 5

Analysis

```r
library(fluoR)
library(ggplot2)
library(ggpubr)
library(cowplot)
theme_set(theme_pubclean())

df <- format_data(GCaMP)
```

5.1 Metrics

5.1.1 Peaks

Because of R's relative ease in subsetting data, I did not include a z-score filter in any of the functions.

5.1.1.1 peak/valley finding

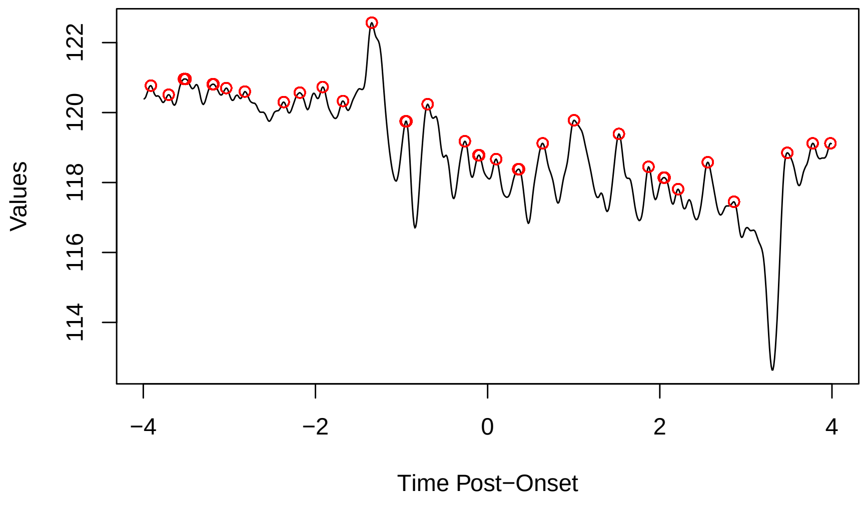
`find_peaks` from the `fluoR` package lets the user find peaks and valleys using a vector of numbers.

The `n.points` parameter is used to determine how many decreasing/increasing points on each side of a peak/valley are required for the point to be considered a peak or valley.

5.1.1.1.1 peaks

Using a positive value for `find_peaks` will return peaks

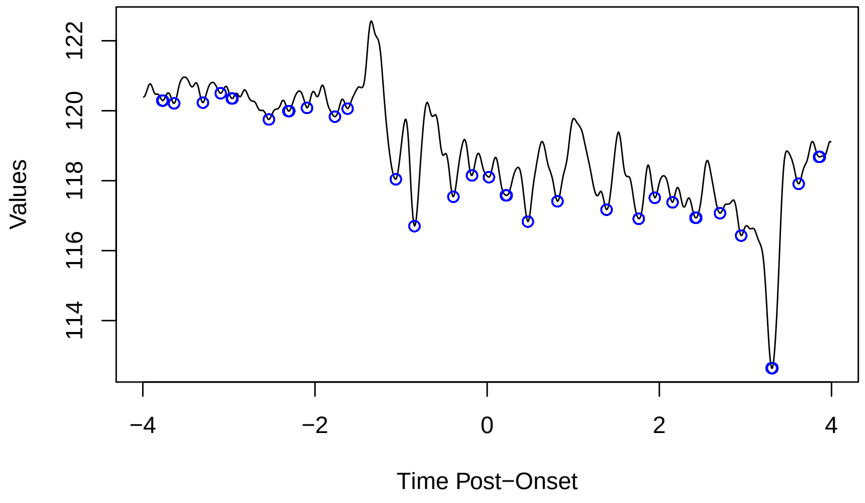
```
pks <- find_peaks(xvals = df$Trial8,
                  n.points = 10) # 10 decreasing points on each side
```



5.1.1.1.2 peaks

Using a negative value for `find_peaks` will return valleys (lower extremes)

```
pks <- find_peaks(xvals = df$Trial8,
                  n.points = -10) # 10 increasing points on each side
```

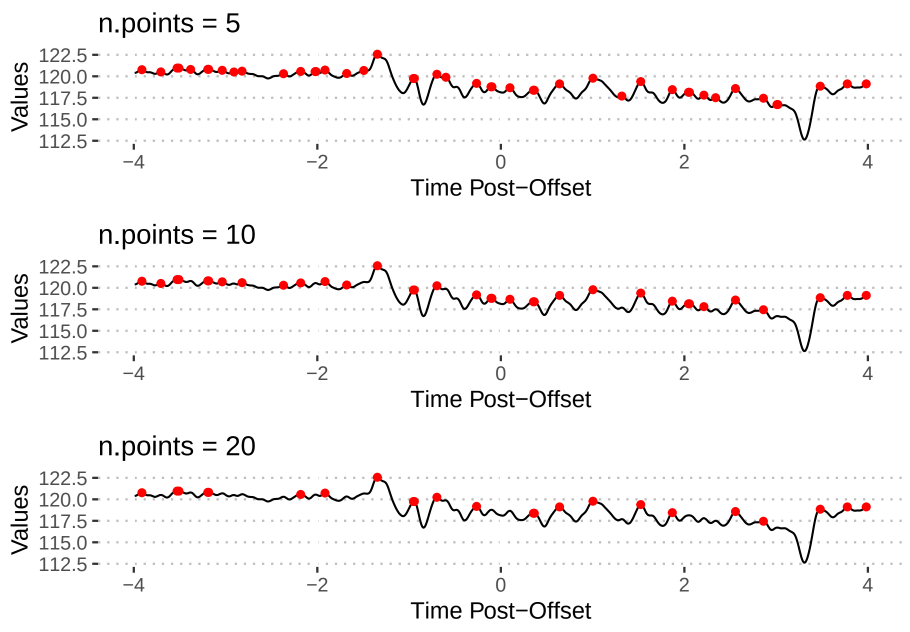


5.1.1.2 adjusting sensitivity

The `n.points` parameter can be changed to prevent returning false positives.

```r
pks5 <- find_peaks(xvals = df$Trial8, n.points = 5)
pks10 <- find_peaks(xvals = df$Trial8, n.points = 10)
pks20 <- find_peaks(xvals = df$Trial8, n.points = 20)
```

- `n.points = 5` returns the indices for 44 peaks
- `n.points = 10` returns the indices for 34 peaks
- `n.points = 20` returns the indices for 23 peaks



5.1.1.3 distance between peaks

fluoR's `distance_between_peaks` function was written to extract values at each peak and between peaks for a single trial. This was inspired by the survival regression analysis done by [3].

```
peak.dist <- distance_between_peaks(dataframe = df,
                                    trial = 8,
                                    time.range = c(-4,4),
                                    n.points = 8)
head(peak.dist)
```

```
##   index.start index.stop index.distance peak.diff.time peak1.intensity
## 1           9         30             21         0.2064          120.77
## 2          30         48             18         0.1770          120.51
## 3          48         50              2         0.0196          120.96
## 4          50         63             13         0.1278          120.96
## 5          63         82             19         0.1868          120.80
## 6          82         83              1         0.0098          120.81
##   peak2.intensity peak.diff.intensity
## 1          120.51               -0.26
## 2          120.96                0.45
## 3          120.96                0.00
## 4          120.80                -0.16
## 5          120.81                 0.01
## 6          120.81                 0.00
```

The variables returned are useful for both graphing and analysis.

- `index.start` and `index.stop` for indexing in reference to the original dataframe
- `index.distance` for discrete counts, regardless of how frequently data points are recorded
- `peak.diff.time` for the time between each pair of peaks
- `peak1.intensity` and `peak2.intensity` for the values each peak is located at (think counting peaks above a specific z score)
- `peak.diff.intensity` for finding continuous increases or decreases in peak fluorescence signal over time

5.1.2 Area under curve

One may decide that it is best to capture the full area underneath a curve. Area under curve (AUC) allows us to combine x (Time) and y (Intensity) variables into one value.

There are three important points one should consider before using AUC as a metric for statistical comparisons: - the length for each time period being compared should be equal (e.g. 2s pre-onset vs. 2s post-onset) - the values should be standardized or centered within each trial (e.g. z-scores from baseline) - computing the AUC includes negative values, which can result in a negative value and is moderately affected by large outliers

fluoR includes the function `auc.pre` to quickly calculate the AUC for a range of trials during a specified period of time. As an example of how this can be used, let's compare full pre-trial and post-trial periods for trials 1-4.

```
### Pre-onset
auc.pre <- auc_trials(dataframe = df, trials = 1:4,
                      time.range = c(-4,0))

### Post-onset
auc.post <- auc_trials(dataframe = df, trials = 1:4,
                       time.range = c(0,4))

head(auc.pre)
```

```
##    Trial1   Trial2   Trial3   Trial4
## 328.5656 329.6170 325.8933 361.4260
```

```r
head(auc.post)
```

```
##   Trial1   Trial2   Trial3   Trial4
## 328.5480 331.3686 329.9789 361.0577
```

The function returns a named vector for each trial's AUC so we can compare AUC values within and between trials.

5.2 Applications

5.2.1 Number of peaks above z score

Regardless of the periods of time being compared, the first step of using this metric is to determine which peaks are significant. We can do this by using the `n.points` parameter with the `distance_between_peaks` function and a set number z-score cutoff. The idea is to make the best combination to represent measured behaviors without excessively skewing the data toward significant or non-significant results.

- judging by word of mouth from other researchers, "significant" peaks tend to be standard z scores of 1.96 and over. But this is not a strict cutoff. It can also be raised to avoid false positives.
- the `n.points` step is a bit trickier and may require adjusting the parameter.

Note that this approach does not specifically represent increases in activity per wave. In other words, it quantifies peak points of fluorescence, not the amount of increase in fluorescence from the preceding valley.

5.2.1.1 Example 1

We first need to standardize the data so that the intensity values represent standard deviations from the baseline mean.

```r
# transform to z-scores using pre-event period as baseline
df.stand <- baseline_transform(dataframe = df, trials = 1:8,
                               baseline.times = c(-4,0))
```

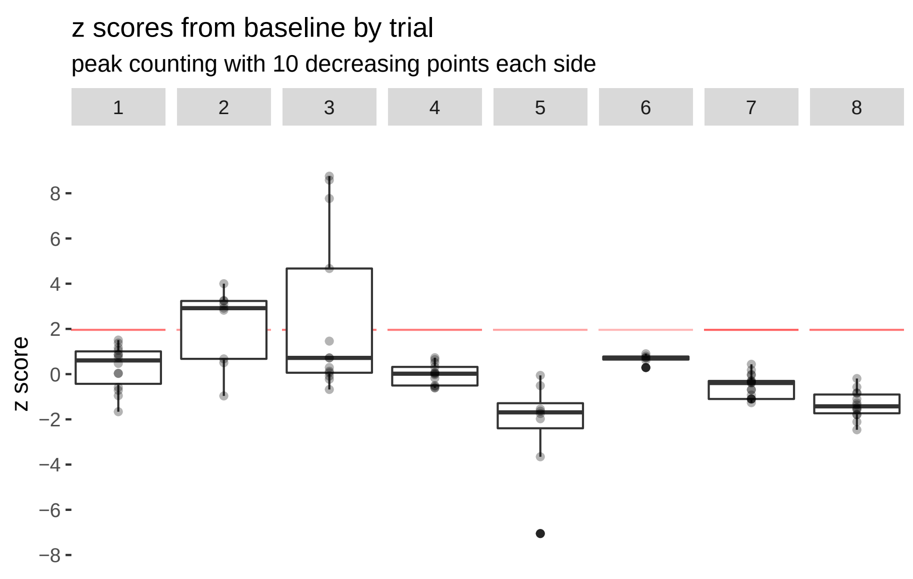
Then, we need to determine criteria for the peaks. Here, we can use the following:

- peak (inflection) points 1.96 z scores and above
- peaks must have 10 data points decreasing on each side to be considered "significant"

To apply this method, is easiest to apply the `distance_between_peaks` function across trials using a loop. I plan to change this function in the future for automatic compatibility with multiple trials.

```r
big.list <- list() # holding list
trial.range <- 1:8

### Loop across trials
for(i in trial.range){
  ## find peaks
  df.pks <- distance_between_peaks(dataframe = df.stand, trial = i,
                                   time.range = c(0,4),
                                   n.points = 10)
  ## specify trial number
  ## add if-else statement incase 0 peaks are returned
  if(nrow(df.pks) > 0){
    df.pks$trial <- i
  }

  ## add to list
  big.list[[i]] <- df.pks
}

### Combine list of data frames
df.peaks <- do.call(rbind, big.list)
```

Now we have our data frame of information on peaks with 10 decreasing points on each side.



It seems like only trials 2 and 3 had any post-event z scores above 1.96. Therefore, the fluorescence signal likely did not spike or increase for a significant number of times past baseline for 6 of 8 trials.

We can get the specific number of peaks that filled the z-score criteria as well. The `dplyr` package has a few functions that let us quickly compute counts by group.

```r
library(dplyr)

### Filter based on z score cutoff
df.peaks.filt <- df.peaks %>%
  filter(peak1.intensity > 1.96 | peak2.intensity > 1.96)

### Count values per trial
total.peaks <- df.peaks.filt %>%
  group_by(trial) %>%
  tally()

print(total.peaks)
```

```
## # A tibble: 2 x 2
##   trial     n
##   <int> <int>
## 1     2     7
## 2      3      5
```

Indeed, trials 2 and 3 were the only trials with fluorescence peaks above 1.96 that had 10 decreasing points on each side.

5.2.2 Distance between peaks

In the context of recording neural activity, measuring the average distance or time between peaks is representative of how frequently significant spikes in activity occur.

5.2.2.1 Example 1

Compare the time between post-event activity peaks for trials 1-8

The first steps for using the distance between peaks are actually the same as Example 1 from z-score peak counting, so we will use the `df.peaks.filt` dataframe from there. Recall that we have already:

- changed the data to represent standard deviations from baseline (baseline z-scores)
- filtered peaks based on the criteria of z > 1.96 and n.points = 10
- found that only trials 2 and 3 had any fluorescence peaks above 1.96

Conveniently, this dataframe already has all of the information we need.

```
mean.time.bet.peaks <- df.peaks.filt %>%
  group_by(trial) %>%
  summarize(time.bet.peaks = mean(peak.diff.time))
print(mean.time.bet.peaks)
```

```
## # A tibble: 2 x 2
##   trial time.bet.peaks
##   <int>          <dbl>
## 1     2          0.399
## 2     3          0.297
```

Trial 2 has an average time of about 0.399 seconds between significant peaks, while trial 3 has about 0.297 seconds between them.

5.2.3 Area under curve

5.2.3.1 Example 1

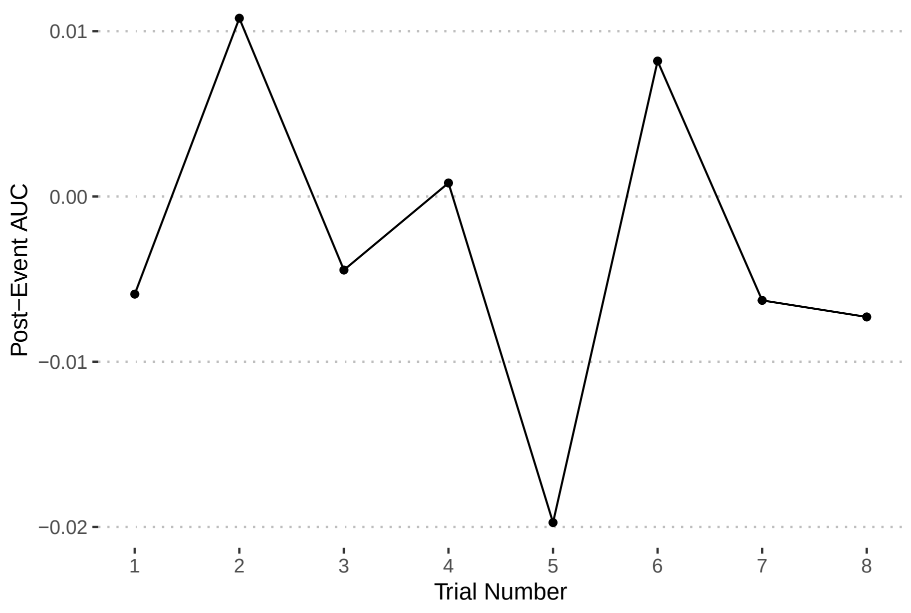
Compare the post-event AUC for trials 1-8

For comparing between trials during the same time period after or before an event, it is appropriate to standardize the data from a baseline point before computing the AUC.

```r
### Format & standardize data
df <- format_data(GCaMP)

df.stand <- baseline_transform(dataframe = df, trials = 1:8,
                               baseline.times = c(0,4),
                               type = 'z_standard')
```

```r
### Compute AUC values and create new data frame
auc.post <- auc_trials(dataframe = df.stand, trials = 1:8,
                       time.range = c(0,4))

auc.postdf <- data.frame(
  Trial = 1:8,
  Value = auc.post
)
```

```r
### Graphing
ggplot(auc.postdf) +
  geom_point(aes(x = Trial, y = Value)) +
  geom_line(aes(x = Trial, y = Value)) +
  scale_x_continuous(breaks = 1:8) +
  labs(
    x = 'Trial Number',
    y = 'Post-Event AUC'
  )
```



5.2.3.2 Example 2

Compare pre- vs. post- event activity for trials 1-8

This is a bit tricker. If you are comparing a baseine period to a post-event period, it will likely no longer be appropriate to use baseline deviations in transforming values for the full time series - at least when using the AUC.

This is because fluctuations in fluorescence on one side of the event (baseline) will be around the same, while post-event deviations from the norm will be exaggerated; causing a higher AUC.

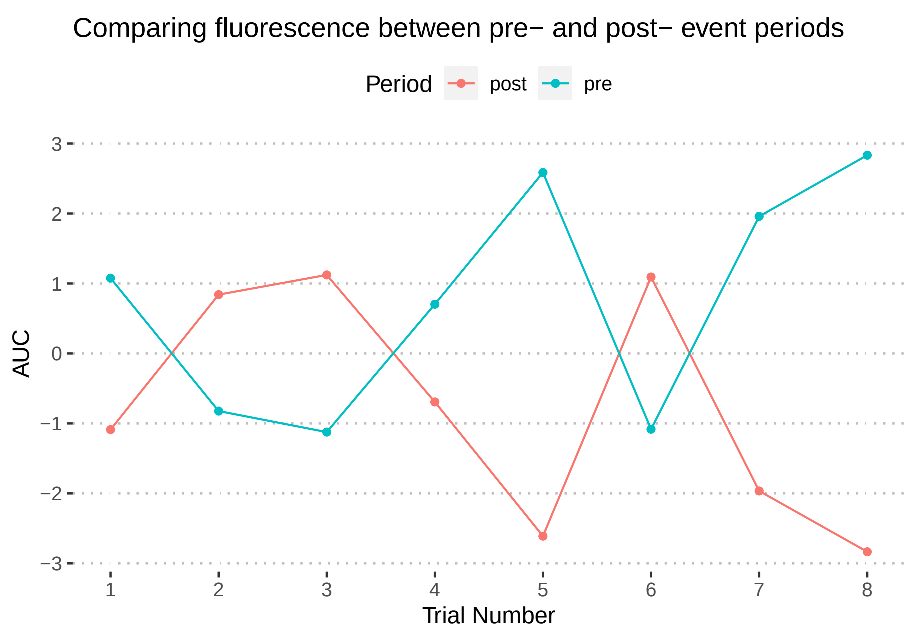
With that in mind, let's do a typical z-score transformation for this data. Since each trial is measured for 4 seconds before event onset and 4 seconds after, `baseline_transform` for the full 8-second period will result in the same thing as manually calculating z-scores for each trial.

```
### Format & standardize data
df <- format_data(GCaMP)

df.stand <- fluoR::baseline_transform(dataframe = df,
                                      trials = 1:8,
                                      baseline.times = c(-4,4), # or min-max timestamps
                                      type = 'z_standard')
```

```r
### Compute AUC values and create new data frame
auc.pre <- auc_trials(dataframe = df.stand, trials = 1:8,
                      time.range = c(-4,0))
auc.post <- auc_trials(dataframe = df.stand, trials = 1:8,
                       time.range = c(0,4))

auc.df <- data.frame(
  Trial = rep(1:8, times = 2),
  Value = c(auc.pre, auc.post), # 8 values pre & 8 values post
  Period = rep(c('pre', 'post'), each = 8)
)
```

We could graph some form of line or bar graph comparison

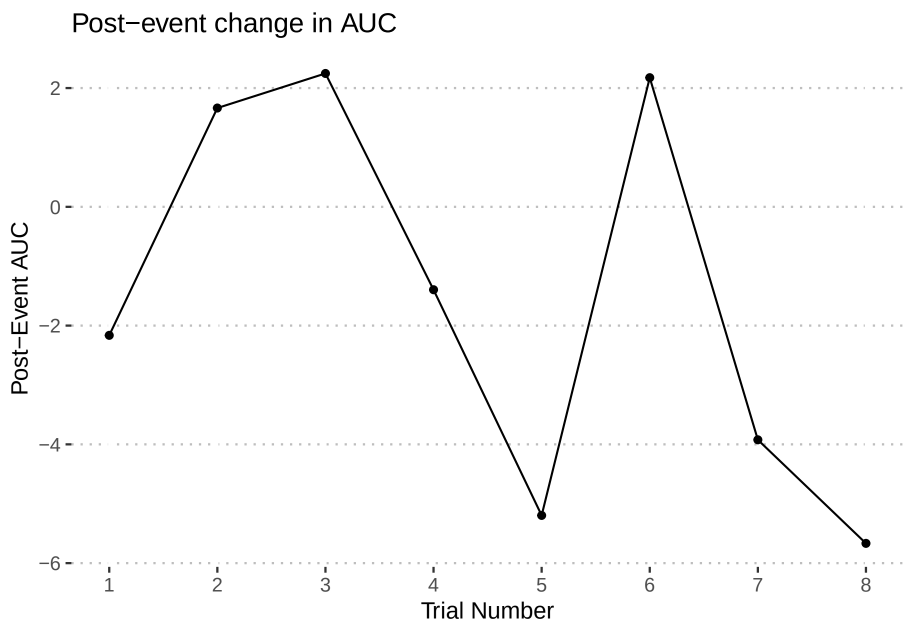
```r
### Graphing
ggplot(auc.df) +
  geom_point(aes(x = Trial, y = Value,
                 color = Period, group = Period)) +
  geom_line(aes(x = Trial, y = Value,
                color = Period, group = Period)) +
  scale_x_continuous(breaks = 1:8) +
  labs(
    x = 'Trial Number',
    y = 'AUC',
    title = "Comparing fluorescence between pre- and post- event periods"
  )
```



Or we could subtract the values and graph the difference between pre- and post-trial AUC values. In other words, how much higher or lower is the post-trial AUC than the pre-trial AUC?

```r
auc.diff <- auc.post - auc.pre

auc.diffdf <- data.frame(
  Trial = 1:8,
  Value = auc.diff # 8 values pre & 8 values post
)
```

```r
### Graphing
ggplot(auc.diffdf) +
  geom_point(aes(x = Trial, y = Value)) +
  geom_line(aes(x = Trial, y = Value)) +
  scale_x_continuous(breaks = 1:8) +
  labs(
    x = 'Trial Number',
    y = 'Post-Event AUC',
    title = 'Post-event change in AUC'
  )
```



On the other hand, it should still be okay to center at the baseline mean or median for interpretation purposes. This is because it is assumed that, after centering at baseline, the base line will hover around 0. Waves alternating above or below the 0-point on the y axis will mostly cancel each other out.

Bibliography

[1] Hans W. Borchers. pracma: Practical Numerical Math Functions, December 2019. URL https://CRAN.R-project.org/package=pracma.

[2] UCLA Statistical Consulting Group. How can I explore different smooths in ggplot2? R FAQ, 2016. URL https://stats.idre.ucla.edu/r/faq/how-can-i-explore-different-smooths-in-ggplot2/.

[3] Lisa A. Gunaydin, Logan Grosenick, Joel C. Finkelstein, Isaac V. Kauvar, Lief E. Fenno, Avishek Adhikari, Stephan Lammel, Julie J. Mirzabekov, Raag D. Airan, Kelly A. Zalocusky, Kay M. Tye, Polina Anikeeva, Robert C. Malenka, and Karl Deisseroth. Natural Neural Projection Dynamics Underlying Social Behavior. *Cell*, 157(7):1535–1551, June 2014. ISSN 0092-8674, 1097-4172. doi: 10.1016/j.cell.2014.05.017. URL https://www.cell.com/cell/abstract/S0092-8674(14)00659-X. Publisher: Elsevier.

[4] David H. Root. Fiber Photometry Epoch Averaging Example, 2018. URL https://www.tdt.com/support/matlab-sdk/offline-analysis-examples/fiber-photometry-epoch-averaging-example/.

[5] Simon N. Wood, Natalya Pya, and Benjamin Safken. Smoothing Parameter and Model Selection for General Smooth Models. *Journal of the American Statistical Association*, 111(516):1548–1563, October 2016. ISSN 0162-1459, 1537-274X. doi: 10.1080/01621459.2016.1180986. URL https://www.tandfonline.com/doi/full/10.1080/01621459.2016.1180986.