Analyzing data

Mike Blazanin

Table of Contents

# Where are we so far?

1. Introduction: vignette("gcplyr")
2. Importing and transforming data: vignette("import\_transform")
3. Incorporating design information: vignette("incorporate\_designs")
4. Pre-processing and plotting your data: vignette("preprocess\_plot")
5. Processing your data: vignette("process") **6. Analyzing your data: vignette("analyze")**
6. Statistics, merging other data, and other resources: vignette("conclusion")

So far, we’ve imported and transformed our measures, combined them with our design information, pre-processed, processed, and plotted our data. Now we’re going to analyze our data by summarizing our growth curves into a number of metrics.

If you haven’t already, load the necessary packages.

library(gcplyr)  
  
library(dplyr)  
library(ggplot2)

#This code was explained in sections 2, 3, 4, and 5  
#Here we're re-running it so it's available for us to work with  
example\_tidydata <- trans\_wide\_to\_tidy(example\_widedata,  
 id\_cols = "Time")  
example\_design <- make\_design(  
 pattern\_split = ",", nrows = 8, ncols = 12,  
 "Bacteria\_strain" = make\_designpattern(  
 values = paste("Strain", 1:48),  
 rows = 1:8, cols = 1:6,  
 pattern = 1:48,  
 byrow = TRUE),  
 "Bacteria\_strain" = make\_designpattern(  
 values = paste("Strain", 1:48),  
 rows = 1:8, cols = 7:12,  
 pattern = 1:48,  
 byrow = TRUE),  
 "Phage" = make\_designpattern(  
 values = c("No Phage"),  
 rows = 1:8, cols = 1:6,  
 pattern = "1"),  
 "Phage" = make\_designpattern(  
 values = c("Phage Added"),  
 rows = 1:8, cols = 7:12,  
 pattern = "1"))  
ex\_dat\_mrg <- merge\_dfs(example\_tidydata, example\_design)  
#> Joining, by = "Well"  
ex\_dat\_mrg$Well <-   
 factor(ex\_dat\_mrg$Well,  
 levels = paste(rep(LETTERS[1:8], each = 12), 1:12, sep = ""))  
ex\_dat\_mrg <- group\_by(ex\_dat\_mrg, Well, Bacteria\_strain, Phage)  
ex\_dat\_mrg <-  
 mutate(ex\_dat\_mrg,  
 smoothed\_med3 =   
 smooth\_data(x = Time, y = Measurements,  
 sm\_method = "moving-median", window\_width\_n = 3),  
 #Note that for the second round, we're using the   
 #first smoothing as the input y  
 smoothed =   
 smooth\_data(x = Time, y = smoothed\_med3,  
 sm\_method = "moving-average", window\_width\_n = 3),  
 deriv = calc\_deriv(x = Time, y = smoothed),  
 deriv\_percap\_hr = calc\_deriv(x = Time, y = smoothed,  
 percapita = TRUE, blank = 0,  
 x\_scale = 3600))  
sample\_wells <- c("A1", "F1", "F10", "E11")

# Analyzing data with summarize

Ultimately, analyzing growth curves requires summarizing the entire time series of data by some metric or metrics. For instance, we may calculate metrics like:

* the maximum density
* the total area under the curve
* the lag time (approximated as the time from the start until maximum per-capita growth rate is achieved)
* the maximum per-capita growth rate
* the density when a diauxic shift occurs
* the time until diauxic shift occurs
* the peak per-capita growth rate after a diauxic shift
* the peak density before a decline from phage predation
* the time when bacteria drop below some density because of phage predation

gcplyr contains a number of functions that make it easier to carry out these calculations. Additionally, gcplyr functions are flexible enough that you can use them in designing your own metric calculations. The following sections highlight general-use gcplyr functions and provide examples to calculate the common metrics above.

But first, we need to familiarize ourselves with one more dplyr function: summarize. Why? Because the upcoming gcplyr analysis functions *must* be used *within* dplyr::summarize. **If you’re already familiar with dplyr’s summarize, feel free to skip the primer in the next section.** If you’re not familiar yet, don’t worry! Continue to the next section, where I provide a primer that will teach you all you need to know on using summarize with gcplyr functions.

## Another brief primer on dplyr: summarize

Here we’re going to focus on the summarize function from dplyr, which *must* be used with the group\_by function we covered in our first primer: **[A brief primer on dplyr]**. summarize carries out user-specified calculations on *each* group in a grouped data.frame independently, producing a new data.frame where each group is now just a single row.

For growth curves, this means we will:

1. group\_by our data so that every well is a group
2. summarize each well with calculations like maximum density or area under the curve

Since summarize will drop columns that the data aren’t grouped by and that aren’t summarized, we will typically want to list all of our design columns for group\_by, along with the plate name and well. Again, make sure you’re *not* grouping by Time, Absorbance, or anything else that varies *within* a well, since if you do dplyr will group timepoints within a well separately.

In the next section, I provide a simple example of how the max function is used with group\_by and summarize to calculate lag time and the maximum per-capita growth rate. If you want to learn more, dplyr has extensive documentation and examples of its own online. Feel free to explore them as desired, but this primer and the coming example should be sufficient to use the remaining gcplyr functions.

## Summarizing with simple base functions: maximum and minimum density

One of the most common steps is calculating global maxima and minima of data. For instance, with bacterial growth, maximum density is one of the most commonly measured traits. Here, we’ll show how to find it using the built-in max function.

First, we need to group our data. As before, group\_by simply requires the data.frame to be grouped, and the names of the columns we want to group by.

#First, drop unneeded columns (optional)  
ex\_dat\_mrg <- dplyr::select(ex\_dat\_mrg,  
 Time, Well, Measurements, Bacteria\_strain, Phage,  
 smoothed, deriv, deriv\_percap\_hr)  
#Then, carry out grouping  
grouped\_ex\_dat\_mrg <- group\_by(ex\_dat\_mrg, Bacteria\_strain, Phage, Well)

Then, we run summarize. Just like for mutate, we specify:

1. the name of the variable we want results saved to
2. the function that calculates the summarized results

In this case, the function should return just a single value for each group. For instance, in the code below we’ve calculated the maximum of the smoothed column, and saved it in a column named max\_dens (note that we need to specify na.rm = TRUE to tell max to ignore all NA values). We’ve saved the output from summarize to a new data.frame: ex\_dat\_mrg\_sum, short for example\_data\_merged\_summarized.

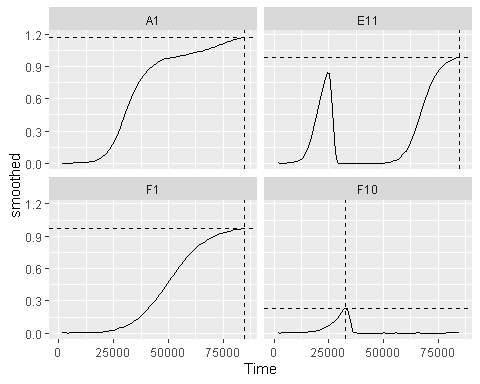
ex\_dat\_mrg\_sum <- summarize(grouped\_ex\_dat\_mrg,  
 max\_dens = max(smoothed, na.rm = TRUE))  
#> `summarise()` has grouped output by 'Bacteria\_strain', 'Phage'. You can override using the  
#> `.groups` argument.  
head(ex\_dat\_mrg\_sum)  
#> # A tibble: 6 × 4  
#> # Groups: Bacteria\_strain, Phage [6]  
#> Bacteria\_strain Phage Well max\_dens  
#> <chr> <chr> <fct> <dbl>  
#> 1 Strain 1 No Phage A1 1.17   
#> 2 Strain 1 Phage Added A7 0.453  
#> 3 Strain 10 No Phage B4 1.21   
#> 4 Strain 10 Phage Added B10 0.959  
#> 5 Strain 11 No Phage B5 1.22   
#> 6 Strain 11 Phage Added B11 1.02

If you want additional characteristics, you simply add them to the summarize. For instance, if we want the time when the maximum density occurs, you just add that as a second argument. In this case, we use the which.max function, which returns the index of the maximum value, to get the index of the Time when the maximum occurs, and save it to a column titled max\_time:

ex\_dat\_mrg\_sum <- summarize(grouped\_ex\_dat\_mrg,  
 max\_dens = max(smoothed, na.rm = TRUE),  
 max\_time = Time[which.max(smoothed)])  
#> `summarise()` has grouped output by 'Bacteria\_strain', 'Phage'. You can override using the  
#> `.groups` argument.  
head(ex\_dat\_mrg\_sum)  
#> # A tibble: 6 × 5  
#> # Groups: Bacteria\_strain, Phage [6]  
#> Bacteria\_strain Phage Well max\_dens max\_time  
#> <chr> <chr> <fct> <dbl> <dbl>  
#> 1 Strain 1 No Phage A1 1.17 84600  
#> 2 Strain 1 Phage Added A7 0.453 30600  
#> 3 Strain 10 No Phage B4 1.21 78300  
#> 4 Strain 10 Phage Added B10 0.959 30600  
#> 5 Strain 11 No Phage B5 1.22 65700  
#> 6 Strain 11 Phage Added B11 1.02 84600

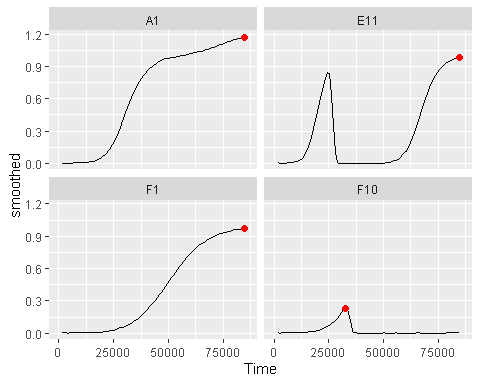
And we can quite easily plot such summarized values as a horizontal line or vertical line on top of our original growth curves data with the geom\_hline or geom\_vline functions:

ggplot(data = dplyr::filter(ex\_dat\_mrg, Well %in% sample\_wells),  
 aes(x = Time, y = smoothed)) +  
 geom\_line() +  
 facet\_wrap(~Well) +  
 geom\_hline(data = dplyr::filter(ex\_dat\_mrg\_sum, Well %in% sample\_wells),   
 aes(yintercept = max\_dens), lty = 2) +  
 geom\_vline(data = dplyr::filter(ex\_dat\_mrg\_sum, Well %in% sample\_wells),   
 aes(xintercept = max\_time), lty = 2)  
#> Warning: Removed 4 row(s) containing missing values (geom\_path).



Alternatively, we could plot these summary points as a point:

ggplot(data = dplyr::filter(ex\_dat\_mrg, Well %in% sample\_wells),  
 aes(x = Time, y = smoothed)) +  
 geom\_line() +  
 facet\_wrap(~Well) +  
 geom\_point(data = dplyr::filter(ex\_dat\_mrg\_sum, Well %in% sample\_wells),   
 aes(x = max\_time, y = max\_dens),  
 size = 2, color = "red")  
#> Warning: Removed 4 row(s) containing missing values (geom\_path).



## Summarizing with simple gcplyr functions: area under the curve

One common metric of growth curves is the total area under the curve. gcplyr has an auc function to easily calculate this area. Just like min and max, it needs to be used inside summarize on a data.frame that has been grouped.

To use auc, simply specify the x and y data whose area-under-the-curve you want to calculate. Here, we calculate the area-under-the-curve of the smoothed column and save it to a column titled auc.

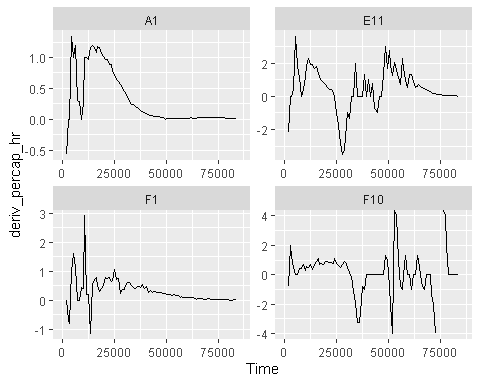
ex\_dat\_mrg\_sum <-  
 summarize(grouped\_ex\_dat\_mrg,  
 auc = auc(x = Time, y = smoothed))  
#> `summarise()` has grouped output by 'Bacteria\_strain', 'Phage'. You can override using the  
#> `.groups` argument.  
head(ex\_dat\_mrg\_sum)  
#> # A tibble: 6 × 4  
#> # Groups: Bacteria\_strain, Phage [6]  
#> Bacteria\_strain Phage Well auc  
#> <chr> <chr> <fct> <dbl>  
#> 1 Strain 1 No Phage A1 55223.  
#> 2 Strain 1 Phage Added A7 3846   
#> 3 Strain 10 No Phage B4 71393.  
#> 4 Strain 10 Phage Added B10 20743.  
#> 5 Strain 11 No Phage B5 73257   
#> 6 Strain 11 Phage Added B11 26149.

## Summarizing on subsets: maximum growth rate

Sometimes, we need to provide limits on the data passed to our simple functions. We can demonstrate this in the process of calculating one of the most common metrics we want to identify: the maximum per-capita growth rate

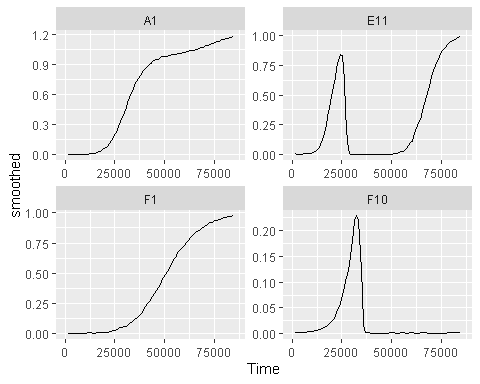
Let’s look again at our smoothed per-capita growth rates:

ggplot(data = dplyr::filter(ex\_dat\_mrg, Well %in% sample\_wells),  
 aes(x = Time, y = deriv\_percap\_hr)) +  
 geom\_line() +  
 facet\_wrap(~Well, scales = "free")  
#> Warning: Removed 5 row(s) containing missing values (geom\_path).



Hmmm, there’s a lot of noise in these plots, what’s going on? We can begin to understand if we also look at our smoothed density values:

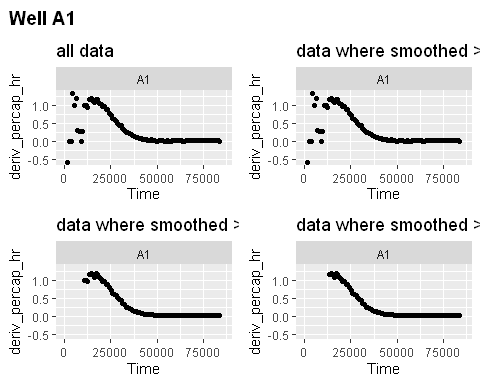
ggplot(data = dplyr::filter(ex\_dat\_mrg, Well %in% sample\_wells),  
 aes(x = Time, y = smoothed)) +  
 geom\_line() +  
 facet\_wrap(~Well, scales = "free")  
#> Warning: Removed 4 row(s) containing missing values (geom\_path).



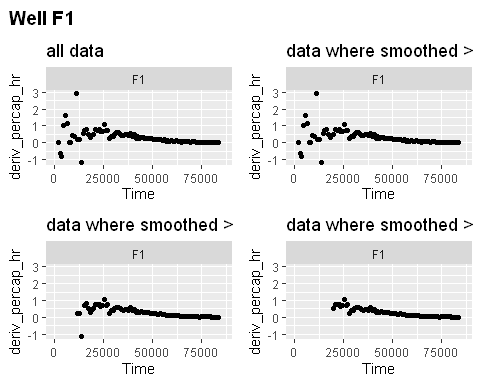
If we compare these plots with the previous ones, we can begin to see that most of the noise is arising when the bacterial populations are very small. Indeed, **this is common with per-capita growth rates, which are very sensitive to noise at low densities**. What can we do about it? We can simply exclude all the values when the *density* is really low.

Let’s plot our per-capita growth rate data at different cutoffs for the minimum *density* of bacteria. Even though these are smoothed values, we’ll use points here, since it better showcases where data are being excluded:

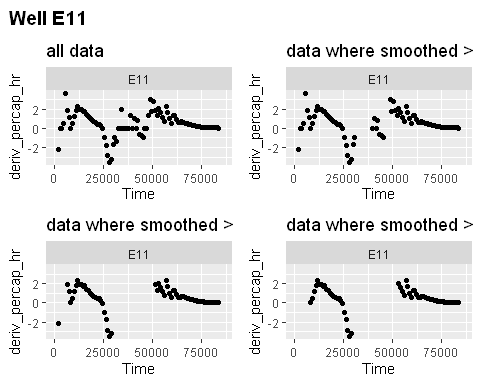
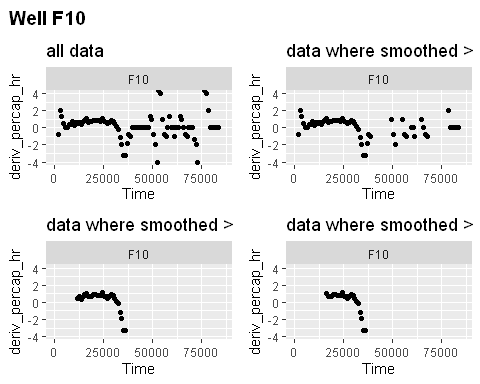
for (my\_well in sample\_wells) {  
 #Title  
 title <- cowplot::ggdraw() +   
 cowplot::draw\_label(paste("Well", my\_well),   
 fontface = "bold", x = 0, hjust = 0) +  
 theme(plot.margin = margin(0, 0, 0, 7))  
   
 #Save x and y limits for all plots so they're all on the same axes  
 xdat <- dplyr::filter(ex\_dat\_mrg, Well == my\_well)$Time  
 ydat <- dplyr::filter(ex\_dat\_mrg, Well == my\_well)$deriv\_percap\_hr  
 xlims <- c(min(xdat[is.finite(xdat)], na.rm = TRUE),  
 max(xdat[is.finite(xdat)], na.rm = TRUE))  
 ylims <- c(min(ydat[is.finite(ydat)], na.rm = TRUE),  
 max(ydat[is.finite(ydat)], na.rm = TRUE))  
   
 #Plot unfiltered data  
 p1 <- ggplot(data = dplyr::filter(ex\_dat\_mrg, Well == my\_well),  
 aes(x = Time, y = deriv\_percap\_hr)) +  
 geom\_point() + facet\_wrap(~Well, scales = "free") +  
 ggtitle("all data") +  
 xlim(xlims[1], xlims[2]) + ylim(ylims[1], ylims[2])  
   
 #Plot data with filters for density  
 p2 <- ggplot(data = dplyr::filter(ex\_dat\_mrg,   
 Well == my\_well, smoothed > 0.001),  
 aes(x = Time, y = deriv\_percap\_hr)) +  
 geom\_point() + facet\_wrap(~Well, scales = "free") +  
 ggtitle("data where smoothed > 0.001") +  
 xlim(xlims[1], xlims[2]) + ylim(ylims[1], ylims[2])  
 p3 <- ggplot(data = dplyr::filter(ex\_dat\_mrg,   
 Well == my\_well, smoothed > 0.005),  
 aes(x = Time, y = deriv\_percap\_hr)) +  
 geom\_point() + facet\_wrap(~Well, scales = "free") +  
 ggtitle("data where smoothed > 0.005") +  
 xlim(xlims[1], xlims[2]) + ylim(ylims[1], ylims[2])  
 p4 <- ggplot(data = dplyr::filter(ex\_dat\_mrg,   
 Well == my\_well, smoothed > 0.01),  
 aes(x = Time, y = deriv\_percap\_hr)) +  
 geom\_point() + facet\_wrap(~Well, scales = "free") +  
 ggtitle("data where smoothed > 0.01") +  
 xlim(xlims[1], xlims[2]) + ylim(ylims[1], ylims[2])  
   
 print(cowplot::plot\_grid(title, cowplot::plot\_grid(p1, p2, p3, p4, ncol = 2),  
 ncol = 1, rel\_heights = c(0.1, 1)))  
}  
#> Warning: Removed 5 rows containing missing values (geom\_point).  
#> Warning: Removed 1 rows containing missing values (geom\_point).  
#> Removed 1 rows containing missing values (geom\_point).  
#> Removed 1 rows containing missing values (geom\_point).  
#> Warning: Removed 5 rows containing missing values (geom\_point).  
#> Warning: Removed 1 rows containing missing values (geom\_point).  
#> Removed 1 rows containing missing values (geom\_point).  
#> Removed 1 rows containing missing values (geom\_point).



#> Warning: Removed 8 rows containing missing values (geom\_point).  
#> Removed 1 rows containing missing values (geom\_point).



#> Warning: Removed 5 rows containing missing values (geom\_point).  
#> Removed 1 rows containing missing values (geom\_point).  
#> Removed 1 rows containing missing values (geom\_point).  
#> Removed 1 rows containing missing values (geom\_point).

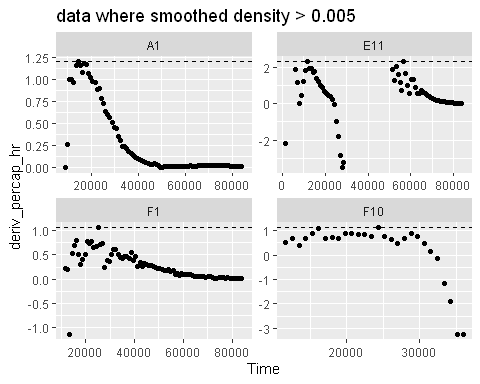


We can see with a cutoff of 0.001, much of the noise still remains. However, once we use a cutoff of 0.005, they are all basically gone, and no high growth rate values are affected by 0.005 vs 0.01. If we checked this pattern for all the wells (as you should in your own analyses), we would see a similar result. **Now, let’s calculate the maximum growth rate of just the subset** of data points where OD is above 0.005. We can specify that subset directly in the summarize command:

ex\_dat\_mrg\_sum <-  
 summarize(grouped\_ex\_dat\_mrg,  
 max\_growth\_rate = max(deriv\_percap\_hr[smoothed > 0.005],   
 na.rm = TRUE))  
#> `summarise()` has grouped output by 'Bacteria\_strain', 'Phage'. You can override using the  
#> `.groups` argument.  
head(ex\_dat\_mrg\_sum)  
#> # A tibble: 6 × 4  
#> # Groups: Bacteria\_strain, Phage [6]  
#> Bacteria\_strain Phage Well max\_growth\_rate  
#> <chr> <chr> <fct> <dbl>  
#> 1 Strain 1 No Phage A1 1.2   
#> 2 Strain 1 Phage Added A7 2.71  
#> 3 Strain 10 No Phage B4 2.84  
#> 4 Strain 10 Phage Added B10 3.48  
#> 5 Strain 11 No Phage B5 2.22  
#> 6 Strain 11 Phage Added B11 3.36

And now we can visualize our findings:

ggplot(data = dplyr::filter(ex\_dat\_mrg,   
 Well %in% sample\_wells, smoothed >= 0.005),  
 aes(x = Time, y = deriv\_percap\_hr)) +  
 geom\_point() +  
 facet\_wrap(~Well, scales = "free") +  
 ggtitle("data where smoothed density > 0.005") +  
 geom\_hline(data = dplyr::filter(ex\_dat\_mrg\_sum, Well %in% sample\_wells),   
 aes(yintercept = max\_growth\_rate), lty = 2)  
#> Warning: Removed 3 rows containing missing values (geom\_point).



## Finding local extrema: peak density, maximum growth rate, lag time, and diauxic shifts

We’ve previously shown how you can use max and min to find the global maxima and minima in data. However, what about *local* maxima or minima? That is, peaks and valleys that are obvious to the eye but aren’t the highest or smallest values in the entire time series. In this section, we’ll show how you can use the gcplyr functions first\_peak and find\_local\_extrema to find points that are local maxima or minima in your data.

### Finding the first peak: peak density, maximum growth rate, and lag time

One particular special case we’re often interested in is the first peak in some set of data. For instance, when bacteria are grown with phages, the density they reach before they start declining due to phage predation (a measure of their susceptibility to the phage)? Alternatively, in the previous section we found the global maximum per-capita growth rate, but some of these maxima happened after near-extinction and recovery. What if we wanted to find the peak growth rate before near-extinction?

#### Peak density

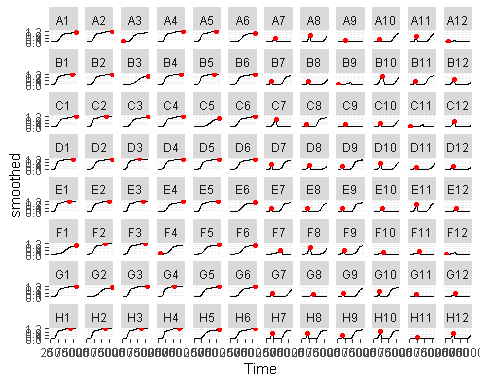
Let’s start with the former example: finding the peak of density.

To identify the first peak, we can use the gcplyr function first\_peak. first\_peak simply requires the y data you want to identify the peak in. In this case, that’s smoothed. We also need to specify whether we want the function to return the index of the first peak, the x value of the peak, or the y value of the peak. We’ll get the x and y values, saving them in columns first\_peak\_x and first\_peak\_y, respectively. (Note that if you want the x-value, you have to provide the x values to first\_peak). As usual, first\_peak needs to be used inside of a summarize command on data that has already been grouped.

ex\_dat\_mrg\_sum <-  
 summarize(grouped\_ex\_dat\_mrg,  
 first\_peak\_x = first\_peak(x = Time, y = smoothed, return = "x"),  
 first\_peak\_y = first\_peak(y = smoothed, return = "y"))  
#> `summarise()` has grouped output by 'Bacteria\_strain', 'Phage'. You can override using the  
#> `.groups` argument.  
  
head(ex\_dat\_mrg\_sum)  
#> # A tibble: 6 × 5  
#> # Groups: Bacteria\_strain, Phage [6]  
#> Bacteria\_strain Phage Well first\_peak\_x first\_peak\_y  
#> <chr> <chr> <fct> <dbl> <dbl>  
#> 1 Strain 1 No Phage A1 84600 1.17   
#> 2 Strain 1 Phage Added A7 30600 0.453  
#> 3 Strain 10 No Phage B4 78300 1.21   
#> 4 Strain 10 Phage Added B10 30600 0.959  
#> 5 Strain 11 No Phage B5 65700 1.22   
#> 6 Strain 11 Phage Added B11 18900 0.348

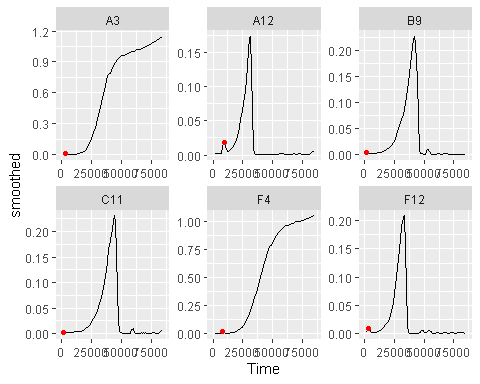
Let’s plot these points on all the wells to confirm they are what we expect:

ggplot(data = ex\_dat\_mrg, aes(x = Time, y = smoothed)) +  
 geom\_line() +  
 facet\_wrap(~Well, nrow = 8, ncol = 12) +  
 geom\_point(data = ex\_dat\_mrg\_sum,   
 aes(x = first\_peak\_x, y = first\_peak\_y),   
 color = "red", size = 1.5)  
#> Warning: Removed 4 row(s) containing missing values (geom\_path).



Hmmm, in most of the wells first\_peak worked perfectly well. However, a few of the wells aren’t quite what we’d expect. Let’s take a closer look at them:

wells\_tocheck <- c("A3", "A12", "B9", "C11", "F4", "F12")  
ggplot(data = dplyr::filter(ex\_dat\_mrg, Well %in% wells\_tocheck),   
 aes(x = Time, y = smoothed)) +  
 geom\_line() +  
 facet\_wrap(~Well, scales = "free") +  
 geom\_point(data = dplyr::filter(ex\_dat\_mrg\_sum, Well %in% wells\_tocheck),   
 aes(x = first\_peak\_x, y = first\_peak\_y),   
 color = "red", size = 1.5)  
#> Warning: Removed 4 row(s) containing missing values (geom\_path).

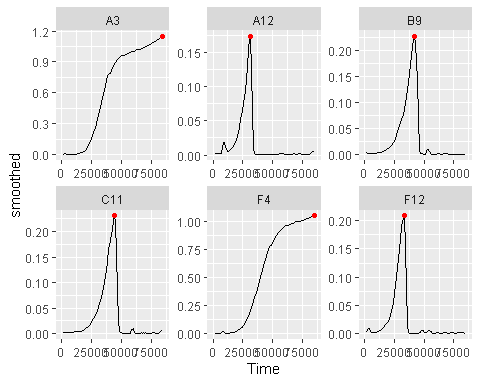


Now we can see what’s going on. In these wells, first\_peak seems to have ‘gotten stuck’ on some earlier smaller peaks. Just like in smoothing, **peak-finding also has tuning parameters**. For first\_peak and find\_local\_extrema, these are window\_width\_n, window\_width and window\_height:

* window\_width determines the width of the window used to search for peaks and valleys, in units of x
* window\_width\_n determines the width of the window, in units of number of data points
* window\_height determines the shortest peak or shallowest valley the window will cross, in units of y

If we want first\_peak to be less sensitive to local peaks, we can increase these parameters (the default setting is window\_width\_n equal to 20% of the length of y, but window\_width is a better approach since it works in units of seconds). Let’s try that:

ex\_dat\_mrg\_sum <-  
 summarize(grouped\_ex\_dat\_mrg,  
 first\_peak\_x = first\_peak(x = Time, y = smoothed, return = "x",  
 window\_width = 35000),  
 first\_peak\_y = first\_peak(x = Time, y = smoothed, return = "y",  
 window\_width = 35000))  
#> `summarise()` has grouped output by 'Bacteria\_strain', 'Phage'. You can override using the  
#> `.groups` argument.  
  
ggplot(data = dplyr::filter(ex\_dat\_mrg, Well %in% wells\_tocheck),   
 aes(x = Time, y = smoothed)) +  
 geom\_line() +  
 facet\_wrap(~Well, scales = "free") +  
 geom\_point(data = dplyr::filter(ex\_dat\_mrg\_sum, Well %in% wells\_tocheck),   
 aes(x = first\_peak\_x, y = first\_peak\_y),   
 color = "red", size = 1.5)  
#> Warning: Removed 4 row(s) containing missing values (geom\_path).

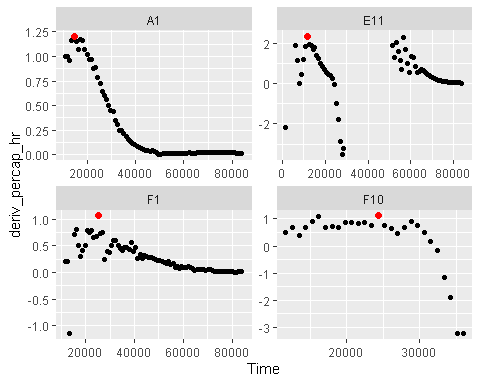


That worked great!

#### Maximum growth rate and lag time

Now let’s look at the other example: using first\_peak to find the first peak in per-capita growth rate to find both the maximum growth rate and the lag time. As we did earlier, we’ll limit our analyses to data where smoothed > 0.005, and visualize using points (even though this is smoothed):

ex\_dat\_mrg\_sum <-  
 summarize(grouped\_ex\_dat\_mrg,  
 max\_growth\_rate = first\_peak(x = Time[smoothed > 0.005],   
 y = deriv\_percap\_hr[smoothed > 0.005],   
 return = "y", window\_width = 35000),  
 lag\_time = first\_peak(x = Time[smoothed > 0.005],   
 y = deriv\_percap\_hr[smoothed > 0.005],   
 return = "x", window\_width = 35000))  
#> `summarise()` has grouped output by 'Bacteria\_strain', 'Phage'. You can override using the  
#> `.groups` argument.  
  
head(ex\_dat\_mrg\_sum)  
#> # A tibble: 6 × 5  
#> # Groups: Bacteria\_strain, Phage [6]  
#> Bacteria\_strain Phage Well max\_growth\_rate lag\_time  
#> <chr> <chr> <fct> <dbl> <dbl>  
#> 1 Strain 1 No Phage A1 1.2 14400  
#> 2 Strain 1 Phage Added A7 1.76 16200  
#> 3 Strain 10 No Phage B4 2.84 10800  
#> 4 Strain 10 Phage Added B10 2.14 10800  
#> 5 Strain 11 No Phage B5 2.22 11700  
#> 6 Strain 11 Phage Added B11 3.36 9000  
  
ggplot(data = dplyr::filter(ex\_dat\_mrg,  
 Well %in% sample\_wells, smoothed > 0.005),  
 aes(x = Time, y = deriv\_percap\_hr)) +  
 geom\_point() +  
 facet\_wrap(~Well, scales = "free") +  
 geom\_point(data = dplyr::filter(ex\_dat\_mrg\_sum, Well %in% sample\_wells),  
 aes(x = lag\_time, y = max\_growth\_rate),  
 color = "red", size = 2)  
#> Warning: Removed 3 rows containing missing values (geom\_point).



Here we can see that in Well E11, first\_peak has identified the peak growth rate at the beginning of the dynamics, and not the one that occurs later on. This means that our lag\_time value will actually reflect what we want it to.

But what if you want to find an extrema that’s *not* the first peak? In the next section, we’ll learn how to use find\_local\_extrema to identify all kinds of local extrema.

### Finding any kind of local extrema: diauxic shifts

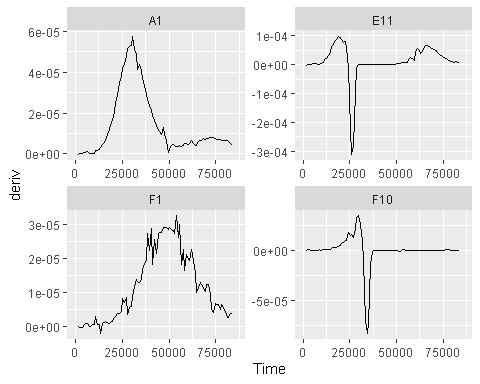
We’ve seen how first\_peak can be used to identify the first peak in data. But what about other kinds of local extrema? The first minimum? The *second* peak?

In order to identify these kinds of extrema, we can use the more-general function find\_local\_extrema. In fact, first\_peak is really just a special case of find\_local\_extrema. Just like first\_peak, find\_local\_extrema only requires a vector of y data in which to find the local extrema, and can return the index, x value, or y of the extrema it finds.

Unlike first\_peak, find\_local\_extrema returns a vector containing *all* of the local extrema found under the given settings. Users can alter which kinds of local extrema are reported using the arguments return\_maxima, return\_minima, and return\_endpoints. However, find\_local\_extrema will always return a vector of all the extrema found, so users must use brackets to select which one they want summarize to save.

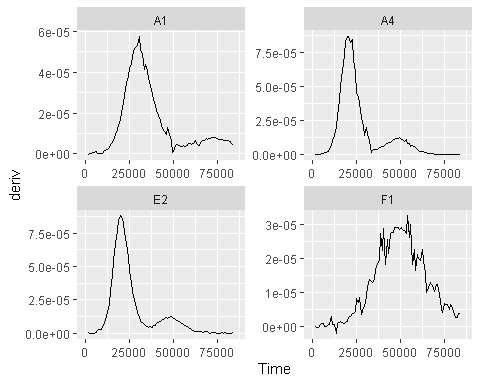
Let’s dig into an example: identifying diauxic shifts. To refresh your memory on what we saw in the section [A simple derivative], here’s a plot of the derivative of some of the wells over time.

ggplot(data = dplyr::filter(ex\_dat\_mrg, Well %in% sample\_wells),  
 aes(x = Time, y = deriv)) +  
 geom\_line() +  
 facet\_wrap(~Well, scales = "free")  
#> Warning: Removed 5 row(s) containing missing values (geom\_path).



In fact, if we look at some more of the wells with no phage added, we’ll see a similar pattern repeatedly.

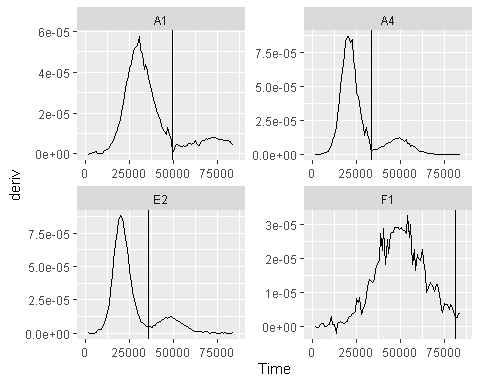
sample\_wells <- c("A1", "A4", "E2", "F1")  
ggplot(data = dplyr::filter(ex\_dat\_mrg, Well %in% sample\_wells),  
 aes(x = Time, y = deriv)) +  
 geom\_line() +  
 facet\_wrap(~Well, scales = "free")  
#> Warning: Removed 5 row(s) containing missing values (geom\_path).



This second, slower, burst of growth after the first wave of growth is common in bacterial growth curves and is called *diauxic growth*.

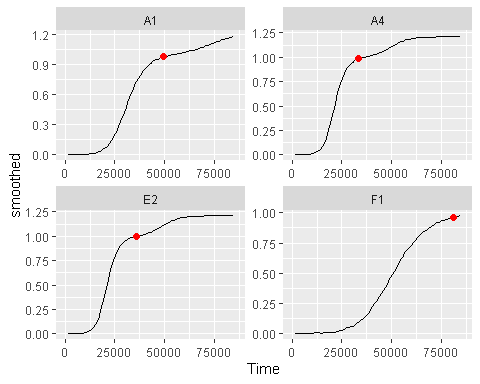
How could we identify the time when the bacteria switch from their first burst of growth to their second? We can find the first minima (that isn’t just the start) in the deriv values. To do so, we specify to find\_local\_extrema that we want return = "x" and we don’t want maxima returned:

ex\_dat\_mrg\_sum <-  
 summarize(  
 grouped\_ex\_dat\_mrg,  
 diauxie\_time = find\_local\_extrema(x = Time, y = deriv, return = "x",  
 return\_maxima = FALSE, return\_minima = TRUE,  
 window\_width\_n = 39)[2])  
#> `summarise()` has grouped output by 'Bacteria\_strain', 'Phage'. You can override using the  
#> `.groups` argument.  
  
#Plot data with vertical line at detected diauxie  
ggplot(data = dplyr::filter(ex\_dat\_mrg, Well %in% sample\_wells),  
 aes(x = Time, y = deriv)) +  
 geom\_line() +  
 facet\_wrap(~Well, scales = "free") +  
 geom\_vline(data = dplyr::filter(ex\_dat\_mrg\_sum, Well %in% sample\_wells),   
 aes(xintercept = diauxie\_time))  
#> Warning: Removed 5 row(s) containing missing values (geom\_path).



Now that we’ve found the point where the bacteria switch, we could quite easily find the density where that happens. To make it easier to follow, we’ll save the *index* where the diauxic shift occurs to a column titled diaxuie\_idx. To get that, we simply run find\_local\_extrema with return = "index". Then, we can get the smoothed value at that index:

ex\_dat\_mrg\_sum <-  
 summarize(  
 grouped\_ex\_dat\_mrg,  
 diauxie\_time = find\_local\_extrema(x = Time, y = deriv, return = "x",  
 return\_maxima = FALSE, return\_minima = TRUE,  
 window\_width\_n = 39)[2],  
 diauxie\_idx = find\_local\_extrema(x = Time, y = deriv, return = "index",  
 return\_maxima = FALSE, return\_minima = TRUE,  
 window\_width\_n = 39)[2],  
 diauxie\_dens = smoothed[diauxie\_idx])  
#> `summarise()` has grouped output by 'Bacteria\_strain', 'Phage'. You can override using the  
#> `.groups` argument.  
  
#Plot data with a point at the moment of diauxic shift  
ggplot(data = dplyr::filter(ex\_dat\_mrg, Well %in% sample\_wells),  
 aes(x = Time, y = smoothed)) +  
 geom\_line() +  
 facet\_wrap(~Well, scales = "free") +  
 geom\_point(data = dplyr::filter(ex\_dat\_mrg\_sum, Well %in% sample\_wells),   
 aes(x = diauxie\_time, y = diauxie\_dens),  
 size = 2, color = "red")  
#> Warning: Removed 4 row(s) containing missing values (geom\_path).

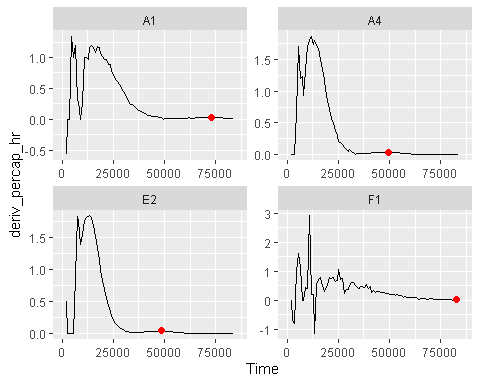


Something that was hard to see on the density plot has now been easily quantified and can be visualized exactly where the shift occurs.

### Combining subsets and local extrema: diauxic growth rate

In the previous section we identified when the bacteria shifted into their second burst of growth. Can we find out what the peak per-capita growth rate was during that second burst? Yes, we just have to put together some of the things we’ve learned already. In particular, we’re going to combine our use of find\_local\_extrema, max, and subsets to find the max(deriv\_percap\_hr) during the times after the diauxic shift:

ex\_dat\_mrg\_sum <-  
 summarize(  
 grouped\_ex\_dat\_mrg,  
 diauxie\_time = find\_local\_extrema(x = Time, y = deriv, return = "x",  
 return\_maxima = FALSE, return\_minima = TRUE,  
 window\_width\_n = 39)[2],  
 diauxie\_percap = max(deriv\_percap\_hr[Time >= diauxie\_time], na.rm = TRUE),  
 diauxie\_percap\_time =   
 Time[Time >= diauxie\_time][  
 which.max(deriv\_percap\_hr[Time >= diauxie\_time])]  
 )  
#> Warning in max(deriv\_percap\_hr[Time >= diauxie\_time], na.rm = TRUE): no non-missing  
#> arguments to max; returning -Inf  
#> `summarise()` has grouped output by 'Bacteria\_strain', 'Phage', 'Well'. You can override  
#> using the `.groups` argument.  
  
#Plot data with a point at the moment of peak diauxic growth rate  
ggplot(data = dplyr::filter(ex\_dat\_mrg, Well %in% sample\_wells),  
 aes(x = Time, y = deriv\_percap\_hr)) +  
 geom\_line() +  
 facet\_wrap(~Well, scales = "free") +  
 geom\_point(data = dplyr::filter(ex\_dat\_mrg\_sum, Well %in% sample\_wells),   
 aes(x = diauxie\_percap\_time, y = diauxie\_percap),  
 size = 2, color = "red")  
#> Warning: Removed 5 row(s) containing missing values (geom\_path).



## Finding threshold-crossings: extinction time and time to density

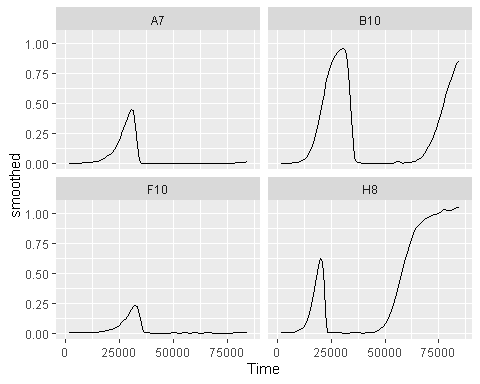
We’ve previously shown how you can find local and global extrema in data, but what if you just want to find when the data passes some threshold value? In this section, we’ll show how you can use the gcplyr functions first\_below and find\_threshold\_crosses to find the points when your data crosses user-defined thresholds.

### Finding the first point below a threshold: extinction time

One common case of threshold-crossing we might be interested in is the first point that our data falls below some threshold density. For instance, when bacteria are grown with phages, the amount of time it takes before the bacterial population falls below some threshold can be a proxy metric for how sensitive the bacteria are to that phage.

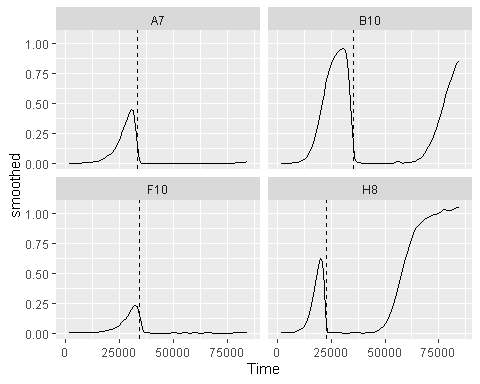
Let’s take a look at the *smoothed* absorbance values in some example wells with both bacteria and phages:

sample\_wells <- c("A7", "B10", "F10", "H8")  
  
ggplot(data = dplyr::filter(ex\_dat\_mrg, Well %in% sample\_wells),  
 aes(x = Time, y = smoothed)) +  
 geom\_line() +  
 facet\_wrap(~Well)  
#> Warning: Removed 4 row(s) containing missing values (geom\_path).



Ok great. Now let’s suppose that I think that an absorbance of 0.15 is a good threshold for extinction in my experiment. How could we use first\_below to calculate the time when that first occurs across all our different wells? Well, primarily, first\_below simply needs our x and y values, the threshold we want to use, as well as whether we want it to return the index of the first point below the threshold, or the x value of that point (since we care about the time it happened here, we’ll do the latter). Additionally, we’ll specify that we don’t care if the startpoint is below the threshold: we only care when the data goes from above to below it.

ex\_dat\_mrg\_sum <-  
 summarize(  
 grouped\_ex\_dat\_mrg,  
 extin\_time = first\_below(x = Time, y = smoothed, threshold = 0.15,  
 return = "x", return\_endpoints = FALSE))  
#> `summarise()` has grouped output by 'Bacteria\_strain', 'Phage'. You can override using the  
#> `.groups` argument.  
head(ex\_dat\_mrg\_sum)  
#> # A tibble: 6 × 4  
#> # Groups: Bacteria\_strain, Phage [6]  
#> Bacteria\_strain Phage Well extin\_time  
#> <chr> <chr> <fct> <dbl>  
#> 1 Strain 1 No Phage A1 NA   
#> 2 Strain 1 Phage Added A7 33307.  
#> 3 Strain 10 No Phage B4 NA   
#> 4 Strain 10 Phage Added B10 35187.  
#> 5 Strain 11 No Phage B5 NA   
#> 6 Strain 11 Phage Added B11 20445.  
  
ggplot(data = dplyr::filter(ex\_dat\_mrg, Well %in% sample\_wells),  
 aes(x = Time, y = smoothed)) +  
 geom\_line() +  
 facet\_wrap(~Well) +  
 geom\_vline(data = dplyr::filter(ex\_dat\_mrg\_sum, Well %in% sample\_wells),  
 aes(xintercept = extin\_time), lty = 2)  
#> Warning: Removed 4 row(s) containing missing values (geom\_path).



All the phage-added wells have a time when the bacteria drop below that threshold, and the plot clearly shows that it’s right where we’d expect it.

### Finding any kind of threshold-crossing: time to density

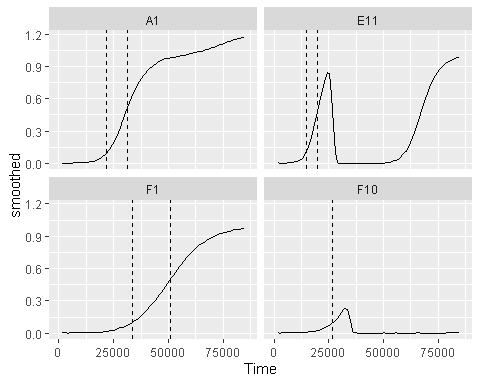
We’ve seen how first\_below can be used to identify the first point some data crosses below a threshold. But what about other kinds of threshold-crossing events? The first point it passes above a threshold? The first point it’s ever below a threshold, including at the start?

In order to identify these kinds of extrema, we can use the more-general function find\_threshold\_crosses. In fact, first\_below is really just a special case of find\_threshold\_crosses. Just like first\_below, find\_threshold\_crosses only requires a threshold and a vector of y data in which to find the threshold crosses, and can return the index or x value of the crossing events it finds.

However, unlike first\_below, find\_threshold\_crosses returns a vector containing *all* of the threshold crossings found under the given settings. Users can alter which kinds of threshold crossings are reported using the arguments return\_rising, return\_falling, and return\_endpoints. However, find\_threshold\_crosses will always return a vector of all the extrema found, so users must use brackets to select which one they want summarize to save.

Let’s dig into an example: identifying the first time the bacteria reach some density, including if they start at that density

sample\_wells <- c("A1", "F1", "F10", "E11")  
ex\_dat\_mrg\_sum <-  
 summarize(  
 grouped\_ex\_dat\_mrg,  
 time\_to\_01 = find\_threshold\_crosses(x = Time, y = smoothed,   
 threshold = 0.1, return = "x",   
 return\_endpoints = TRUE,   
 return\_falling = FALSE)[1],  
 time\_to\_05 = find\_threshold\_crosses(x = Time, y = smoothed,   
 threshold = 0.5, return = "x",   
 return\_endpoints = TRUE,   
 return\_falling = FALSE)[1])  
#> `summarise()` has grouped output by 'Bacteria\_strain', 'Phage'. You can override using the  
#> `.groups` argument.  
head(ex\_dat\_mrg\_sum)  
#> # A tibble: 6 × 5  
#> # Groups: Bacteria\_strain, Phage [6]  
#> Bacteria\_strain Phage Well time\_to\_01 time\_to\_05  
#> <chr> <chr> <fct> <dbl> <dbl>  
#> 1 Strain 1 No Phage A1 21851. 31134.  
#> 2 Strain 1 Phage Added A7 21855. NA   
#> 3 Strain 10 No Phage B4 15178. 20629.  
#> 4 Strain 10 Phage Added B10 15196. 20627.  
#> 5 Strain 11 No Phage B5 14434. 19326.  
#> 6 Strain 11 Phage Added B11 14440. 59796.  
  
ggplot(data = dplyr::filter(ex\_dat\_mrg, Well %in% sample\_wells),  
 aes(x = Time, y = smoothed)) +  
 geom\_line() +  
 facet\_wrap(~Well) +  
 geom\_vline(data = dplyr::filter(ex\_dat\_mrg\_sum, Well %in% sample\_wells),  
 aes(xintercept = time\_to\_01), lty = 2) +  
 geom\_vline(data = dplyr::filter(ex\_dat\_mrg\_sum, Well %in% sample\_wells),  
 aes(xintercept = time\_to\_05), lty = 2)  
#> Warning: Removed 4 row(s) containing missing values (geom\_path).  
#> Warning: Removed 1 rows containing missing values (geom\_vline).

 As we can see, find\_threshold\_crosses has returned the times when the bacteria reached those densities. We can see that some bacteria (e.g. those in Wells A7 and F10) never reached 0.5, so they have an NA value for time\_to\_05. By comparing the times it took each strain to reach an absorbance of 0.1, we could learn something about how soon the bacteria started growing and how quickly they grew.

# What’s next?

Now that you’ve analyzed your data, there’s just a few more notes on best practices for running statistics, merging growth curve analyses with other data, and additional resources for analyzing growth curves.

1. Introduction: vignette("gcplyr")
2. Importing and transforming data: vignette("import\_transform")
3. Incorporating design information: vignette("incorporate\_designs")
4. Pre-processing and plotting your data: vignette("preprocess\_plot")
5. Processing your data: vignette("process")
6. Analyzing your data: vignette("analyze") **7. Statistics, merging other data, and other resources: vignette("conclusion")**