# Analyzing data

Mike Blazanin

2023-01-22

## Contents



## Workflow

1. Introduction
2. Importing & transforming data
3. Incorporating design information
4. Pre-processing and plotting your data
5. Processing your data
6. **Analyzing your data**
7. Statistics, merging other data, and other resources

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 x 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
```

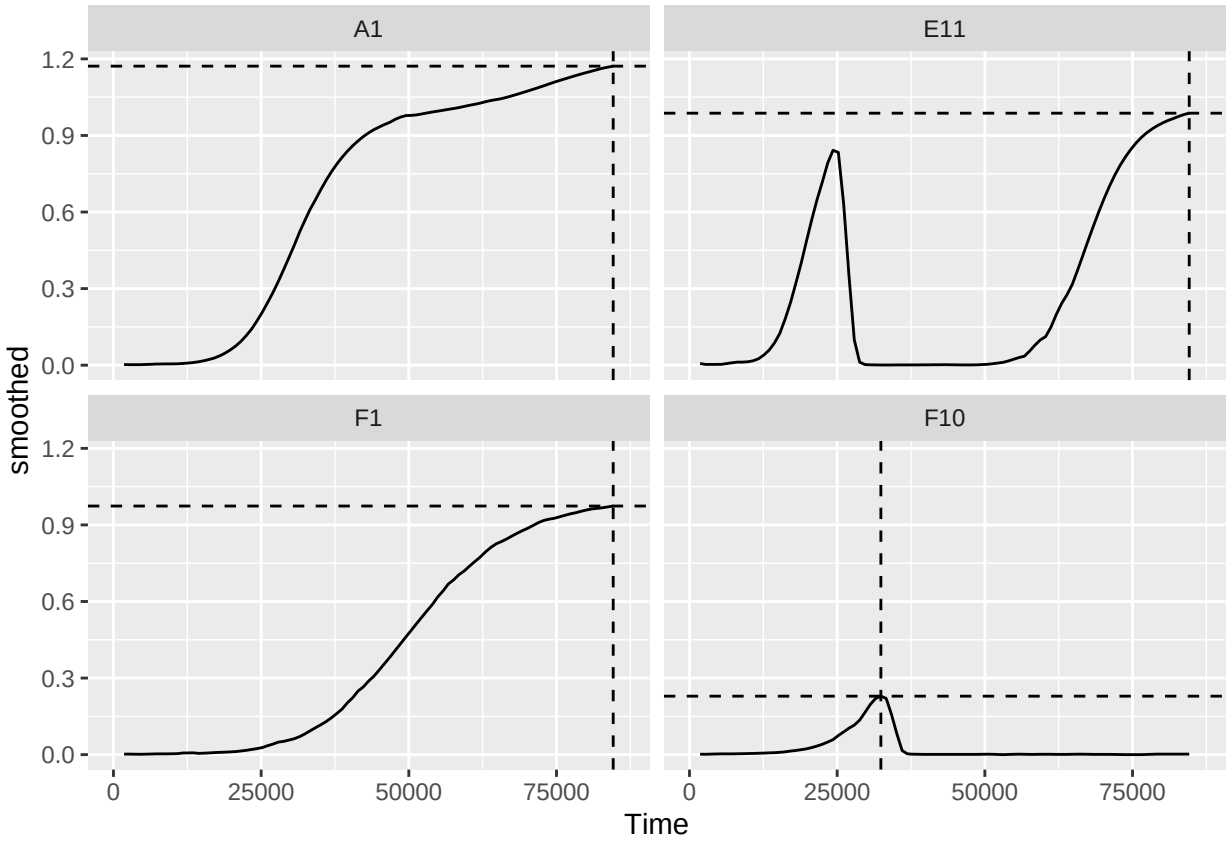
If you want additional characteristics, you simply add them to the summarize. For instance, if we want the time when the maximum density occurs, you just add that as a second argument. In this case, we use the `which.max` function, which returns the index of the maximum value, to get the index of the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

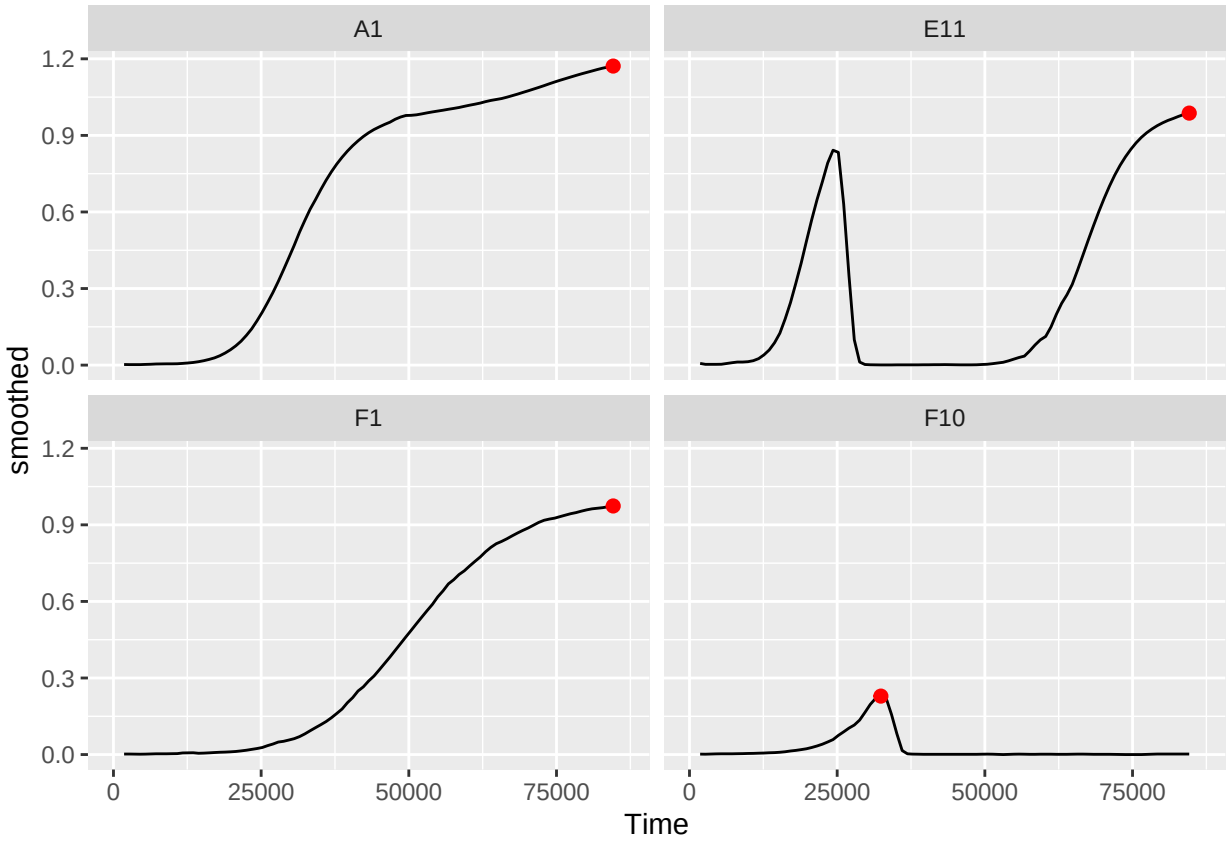
```
ggplot(data = dplyr::filter(ex_dat_mrg, Well %in% sample_wells),
       aes(x = Time, y = smoothed)) +
  geom_line() +
  facet_wrap(~Well) +
  geom_hline(data = dplyr::filter(ex_dat_mrg_sum, Well %in% sample_wells),
             aes(yintercept = max_dens), lty = 2) +
  geom_vline(data = dplyr::filter(ex_dat_mrg_sum, 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 x 4
#> # Groups:   Bacteria_strain, Phage [6]
#>   Bacteria_strain Phage      Well     auc
#>   <chr>           <chr>      <fct>  <dbl>
#> 1 Strain 1        No Phage   A1    55223.
```

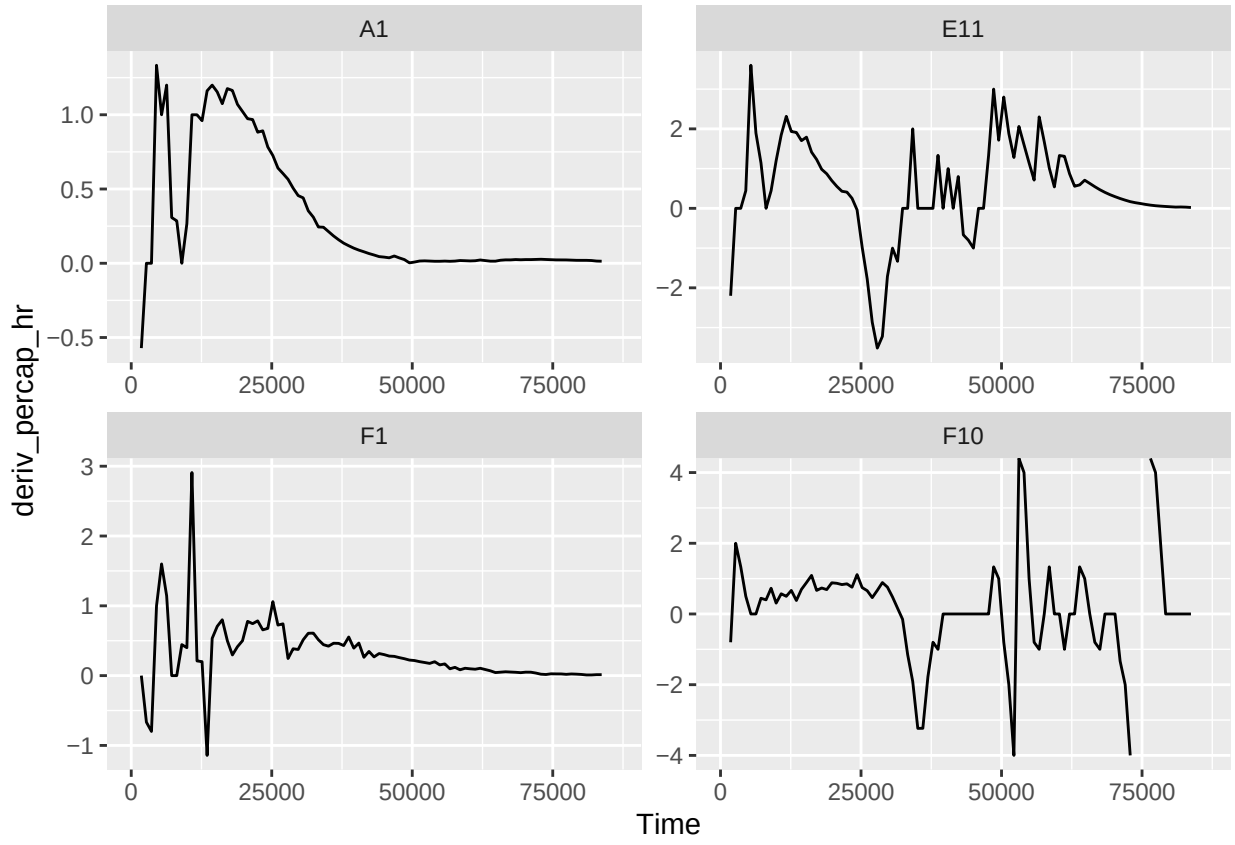
```
#> 2 Strain 1        Phage Added A7     3846
#> 3 Strain 10       No Phage    B4    71393.
#> 4 Strain 10       Phage Added B10   20743.
#> 5 Strain 11       No Phage    B5    73257
#> 6 Strain 11       Phage Added B11   26149.
```

## 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

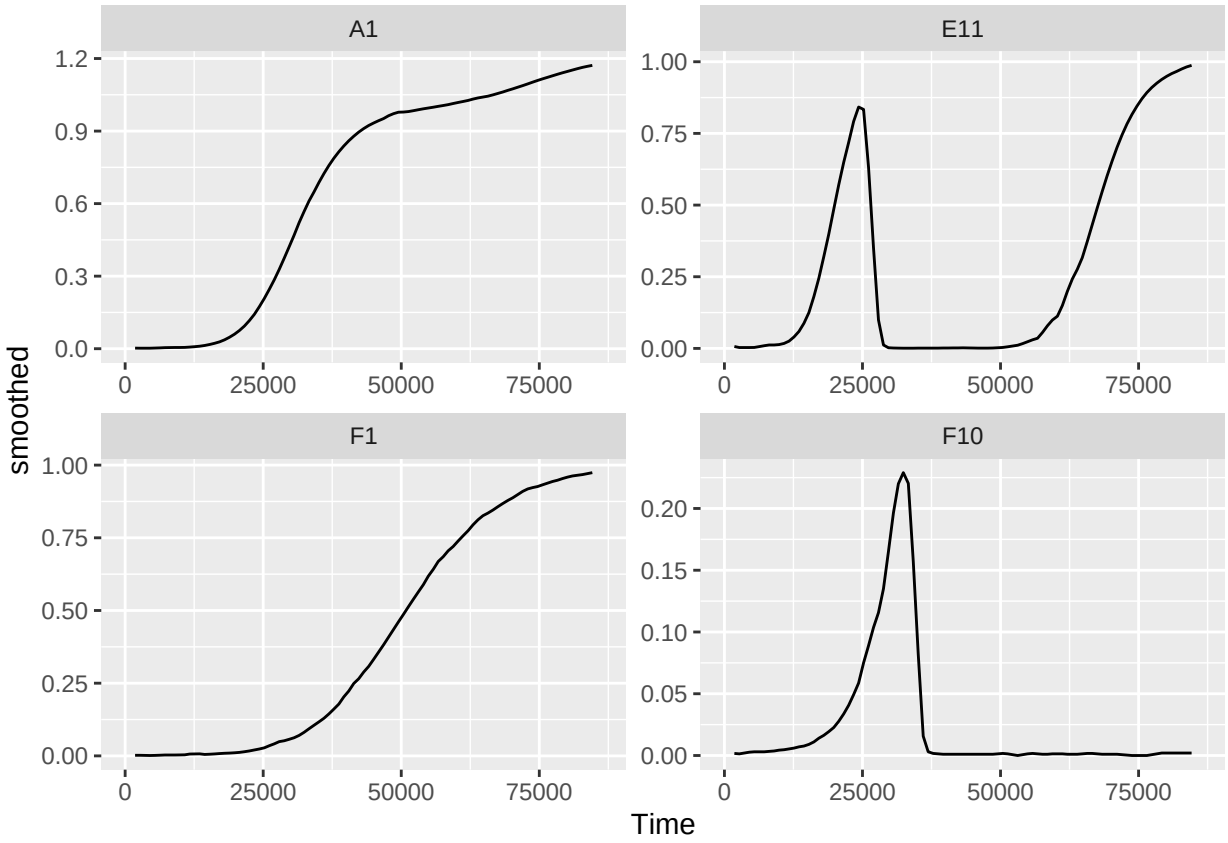
Let's look again at our smoothed per-capita growth rates:

```
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).
```



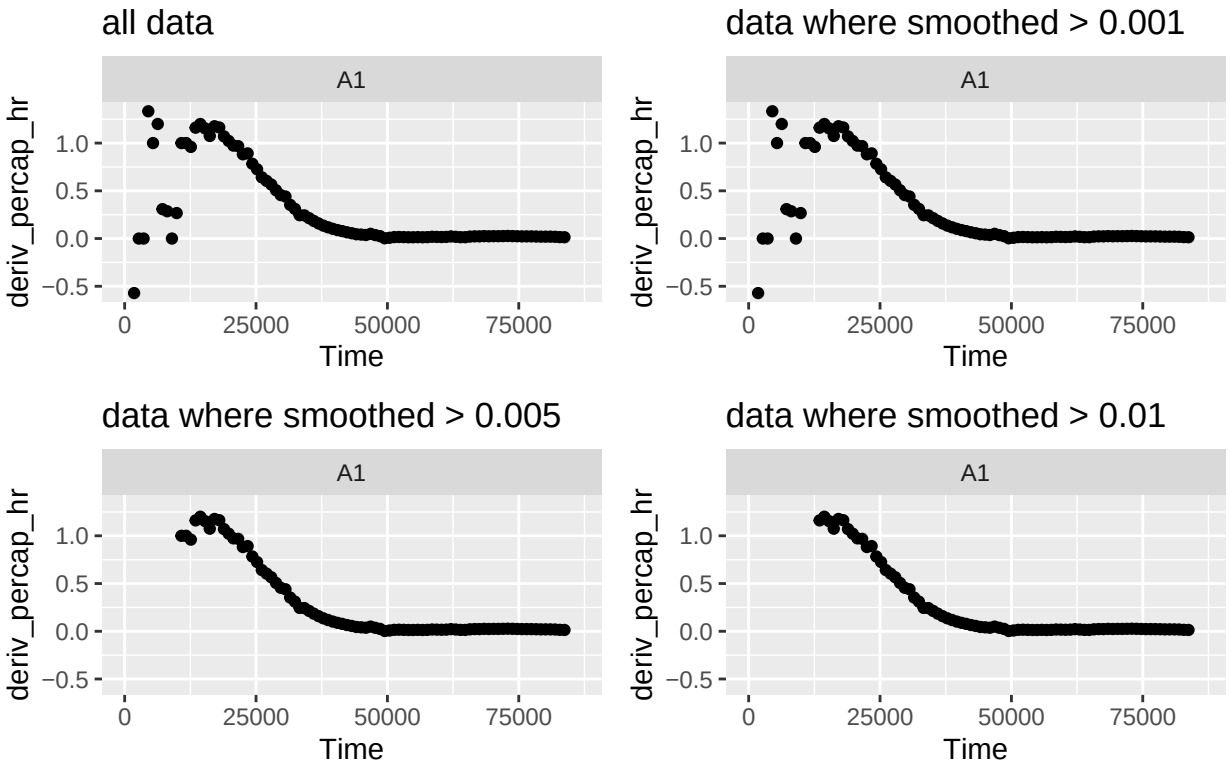

If we compare these plots with the previous ones, we can begin to see that most of the noise is arising when the bacterial populations are very small. Indeed, **this is common with per-capita growth rates, which are very sensitive to noise at low densities**. What can we do about it? We can simply exclude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

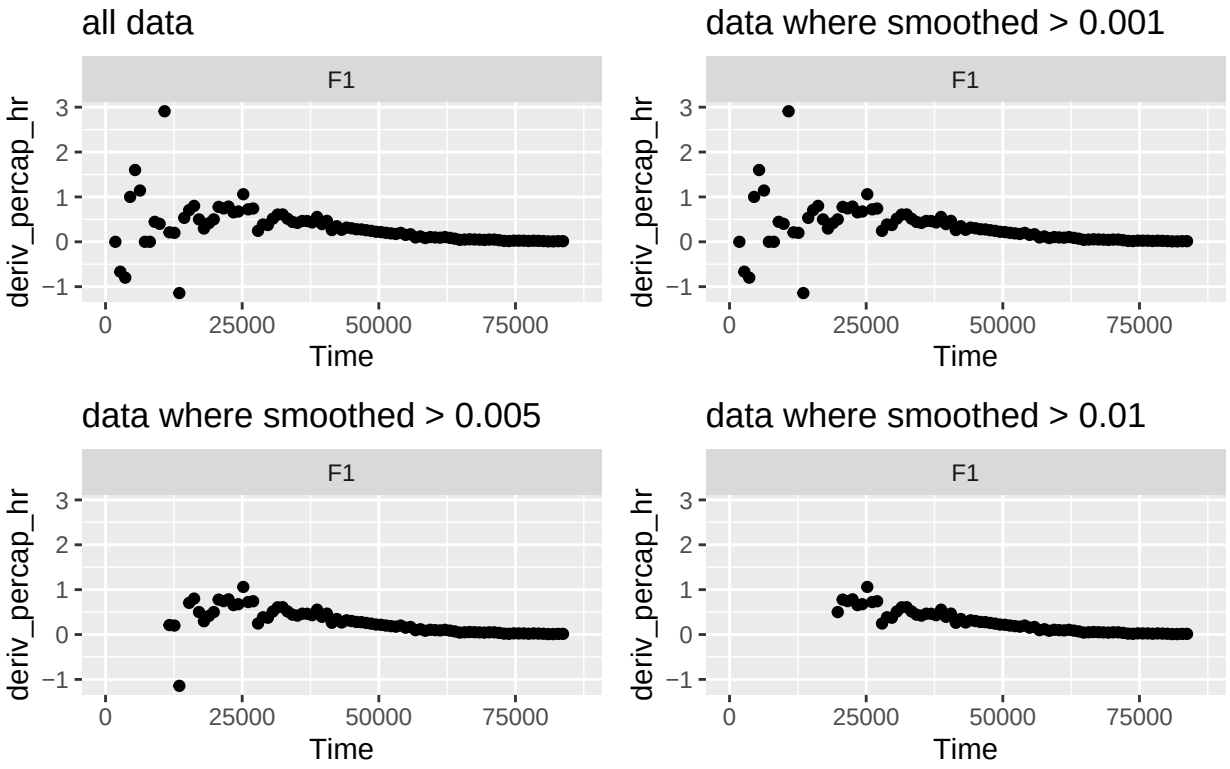
```
            max(xdat[is.finite(xdat)], na.rm = TRUE))
  ylims <- c(min(ydat[is.finite(ydat)], na.rm = TRUE),
             max(ydat[is.finite(ydat)], na.rm = TRUE))

  #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).
```

## Well A1




```
#> Warning: Removed 8 rows containing missing values (geom_point).
#> Removed 1 rows containing missing values (geom_point).
```

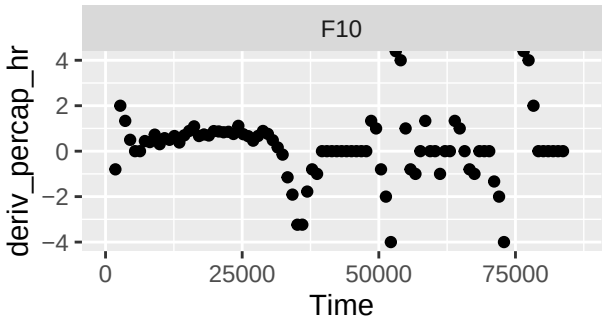
## Well F1



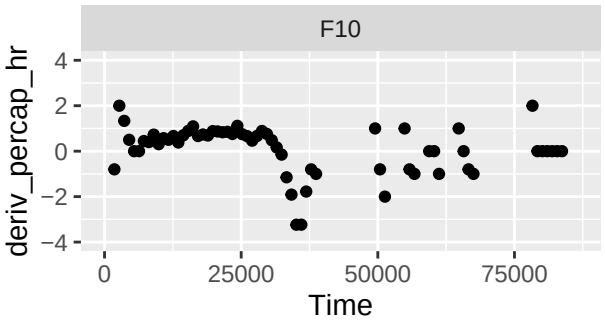

```
#> Warning: Removed 5 rows containing missing values (geom_point).
#> Removed 1 rows containing missing values (geom_point).
#> Removed 1 rows containing missing values (geom_point).
#> Removed 1 rows containing missing values (geom_point).
```

## Well F10

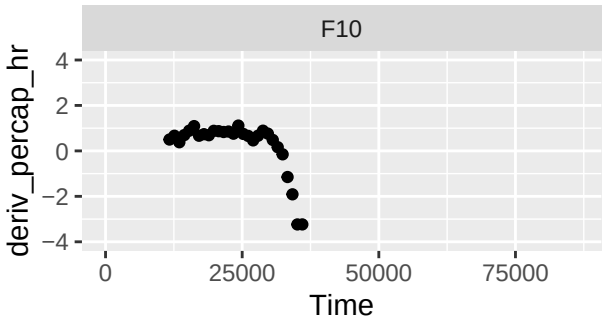
all data



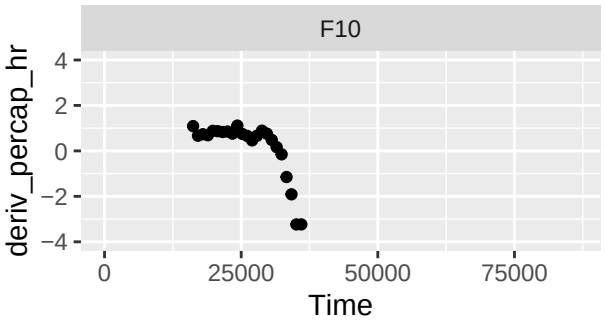

data where smoothed > 0.001



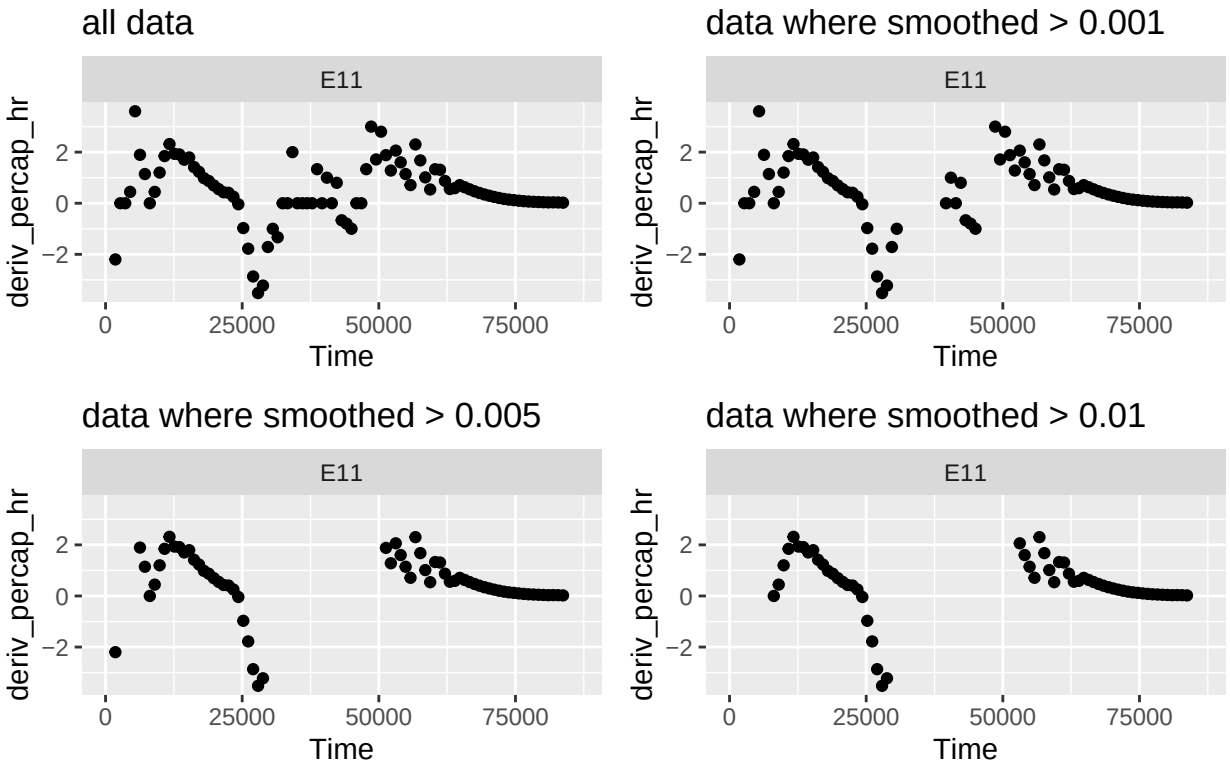

data where smoothed > 0.005




data where smoothed > 0.01



## Well E11

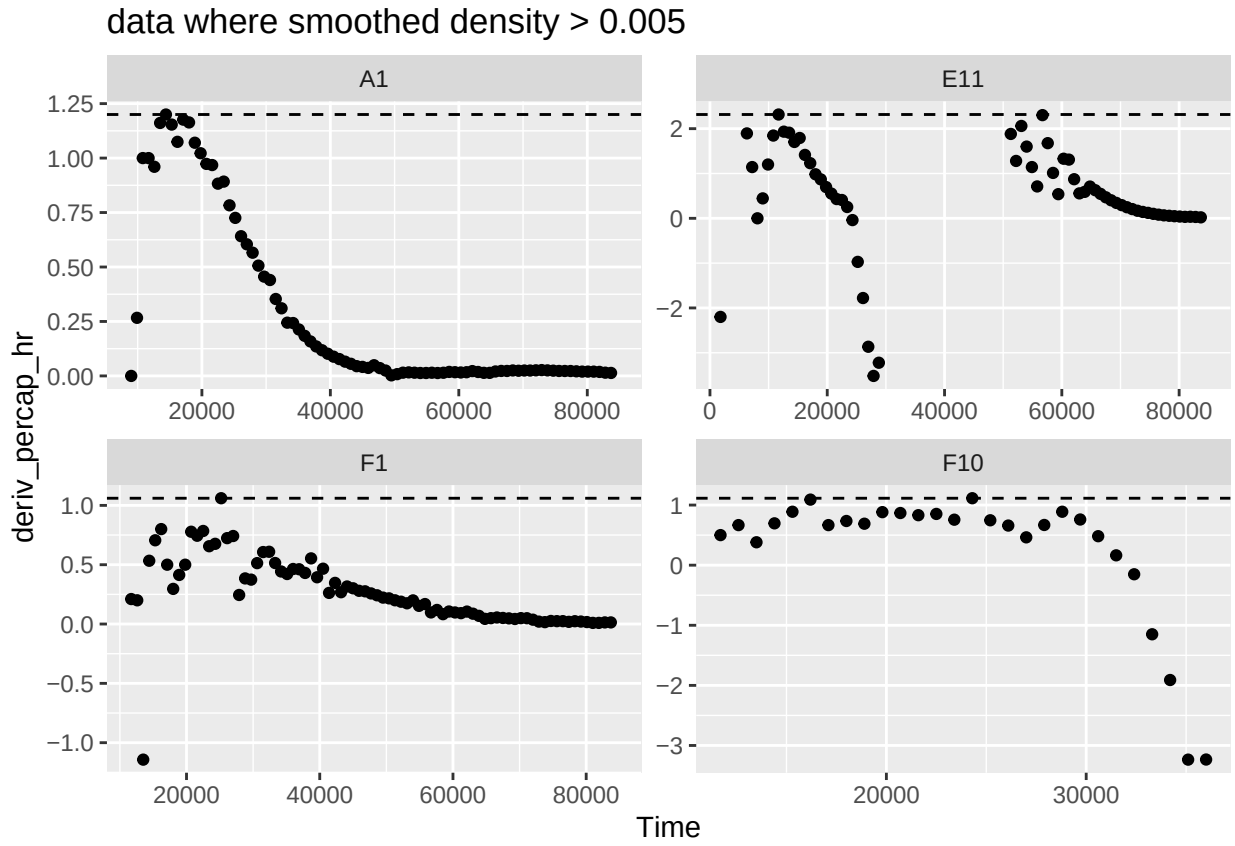



We can see with a cutoff of 0.001, much of the noise still remains. However, once we use a cutoff of 0.005, they are all basically gone, and no high growth rate values are affected by 0.005 vs 0.01. If we checked this pattern for all the wells (as you should in your own analyses), we would see a similar 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 x 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?

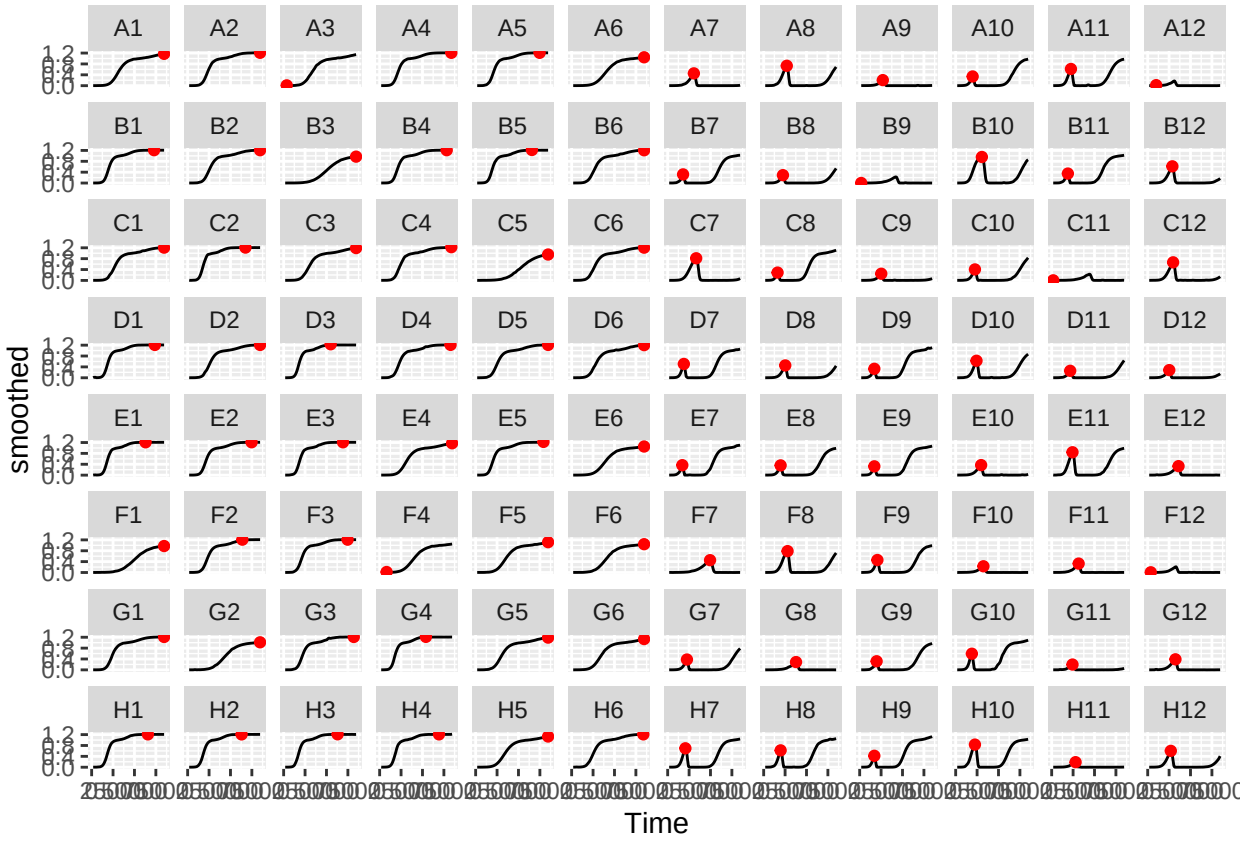
**Peak density** Let's start with the former example: finding the peak of density.

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 x 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
```

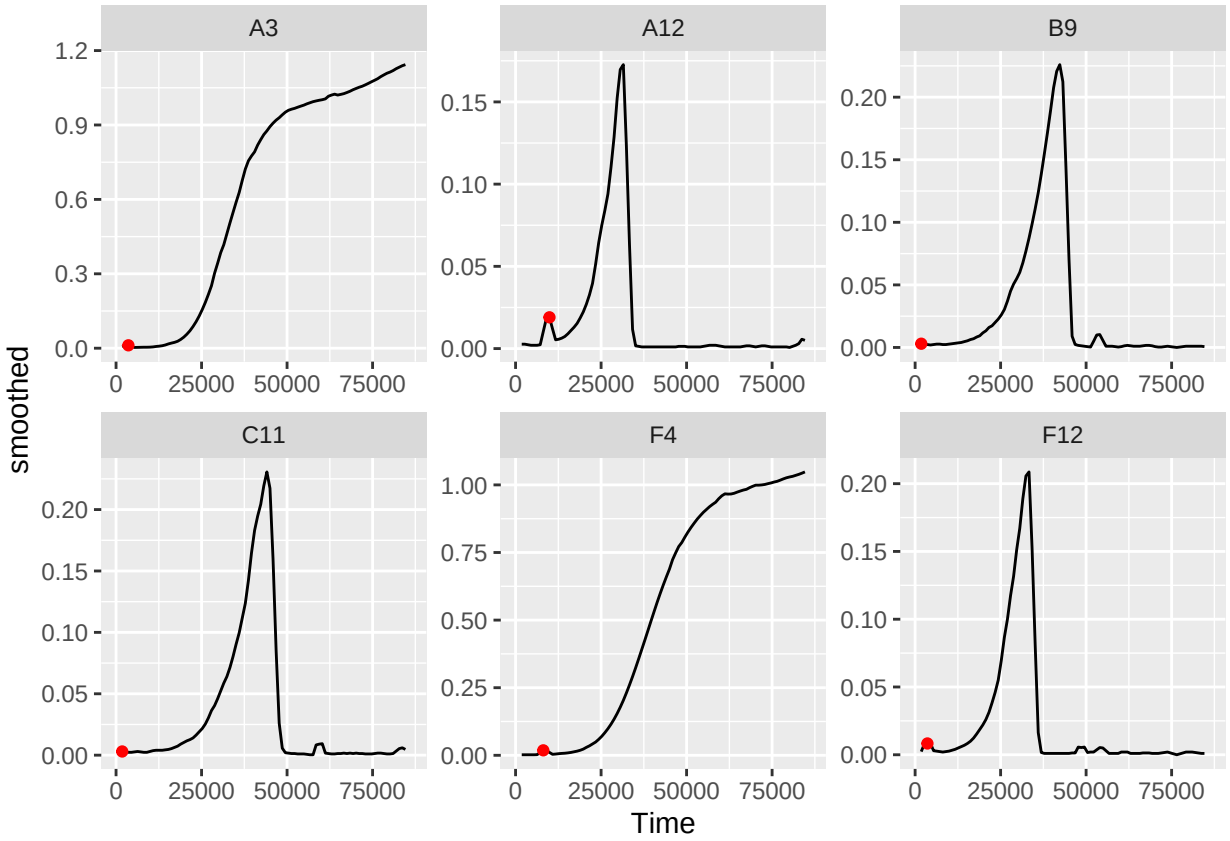
Let's plot these points on all the wells to confirm they are what we expect:

```
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).
```



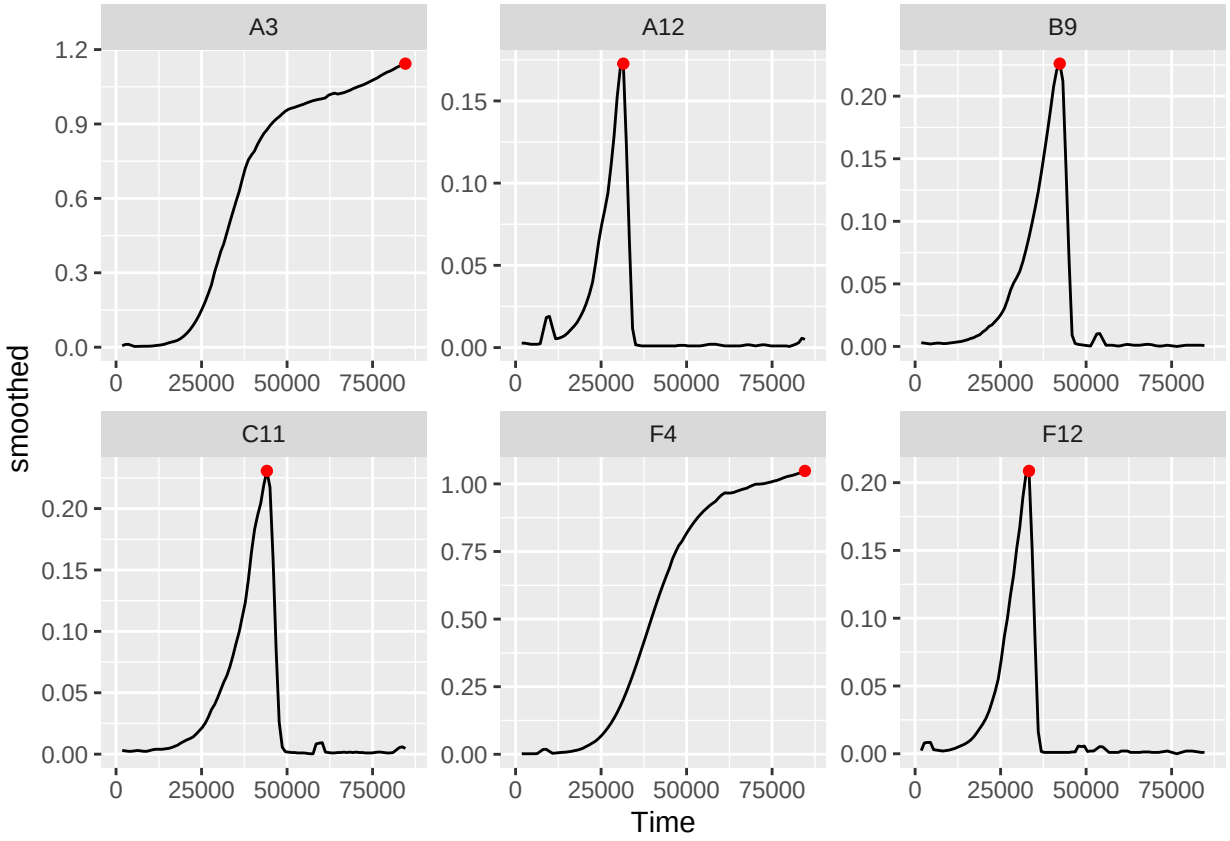

Now we can see what's going on. In these wells, `first_peak` seems to have 'gotten stuck' on some earlier smaller peaks. Just like in smoothing, **peak-finding also has tuning parameters**. For `first_peak` and `find_local_extrema`, these are `window_width_n`, `window_width` and `window_height`:

- `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).
```



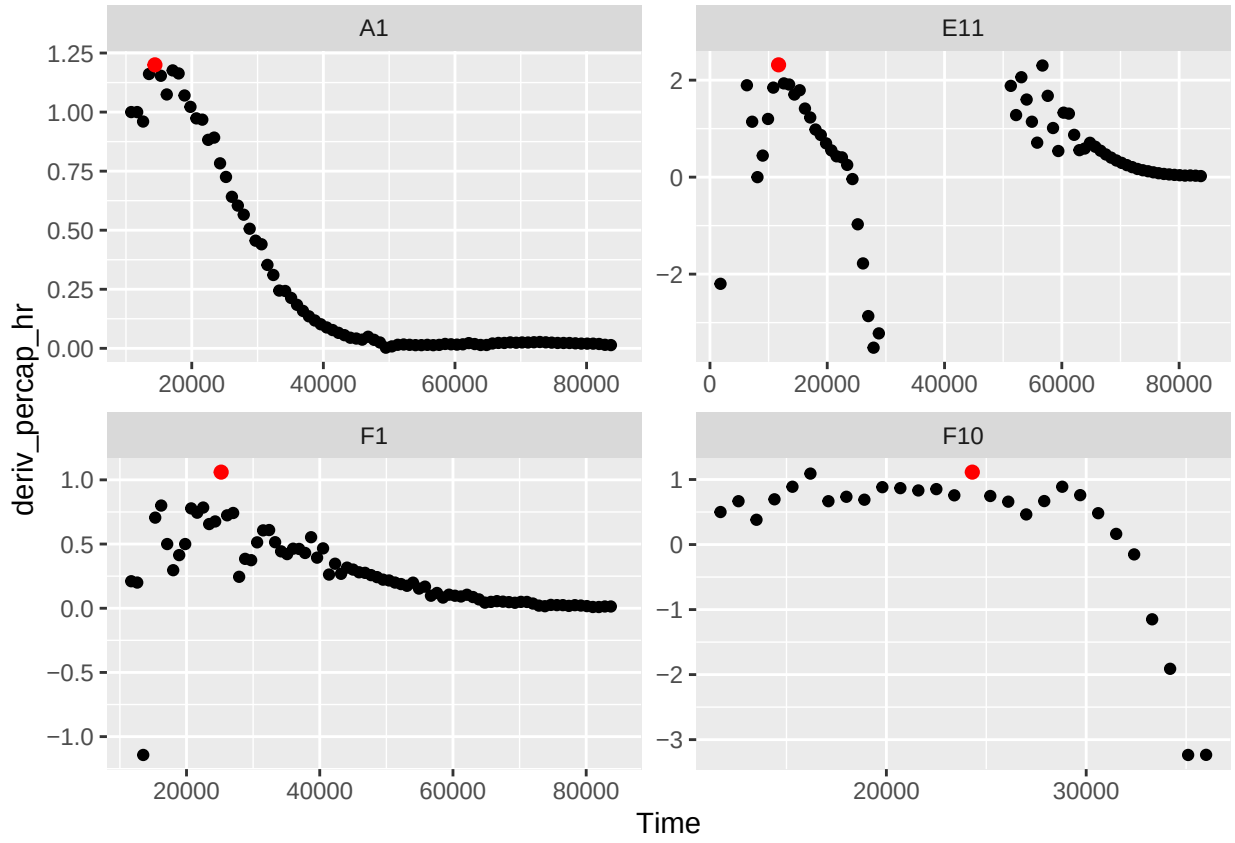

That worked great!

**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 x 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

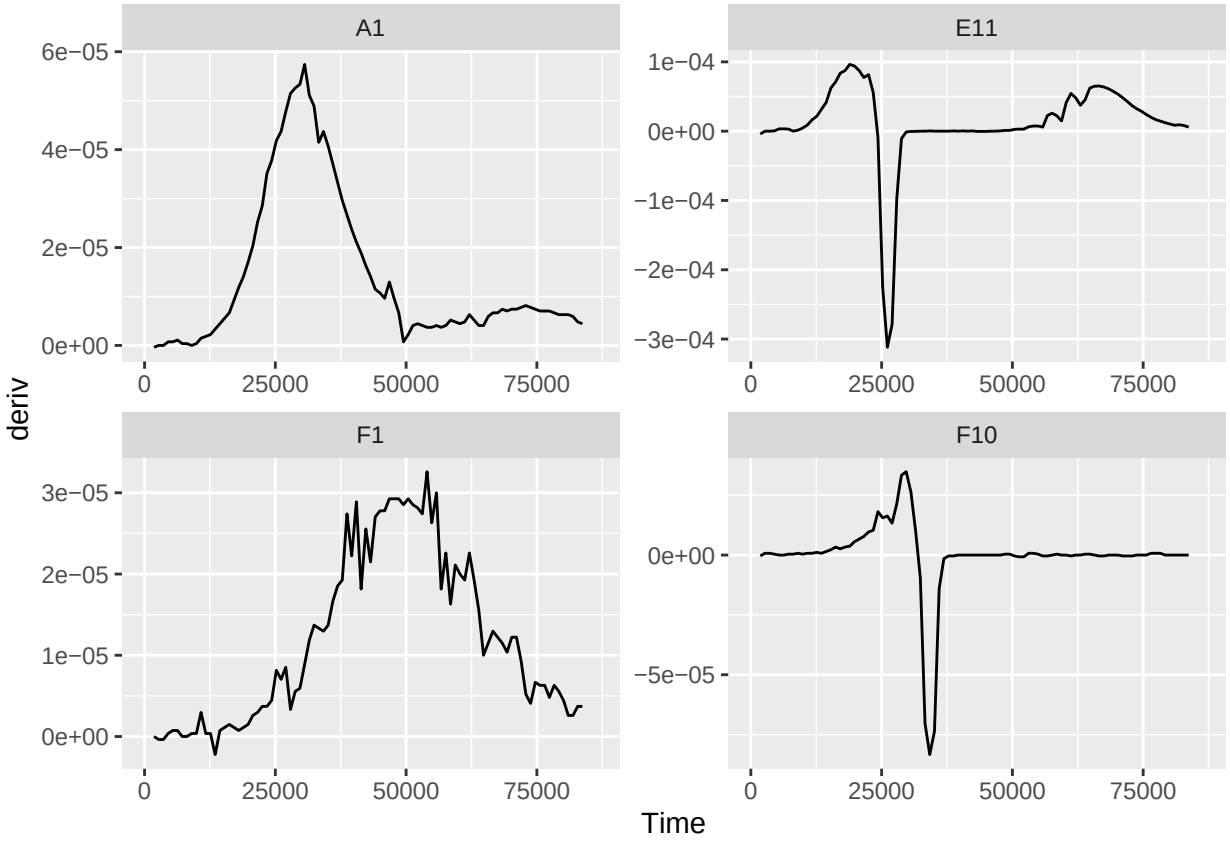
We've seen how `first_peak` can be used to identify the first peak in data. But what about other kinds of local extrema? The first minimum? The *second* peak?

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.

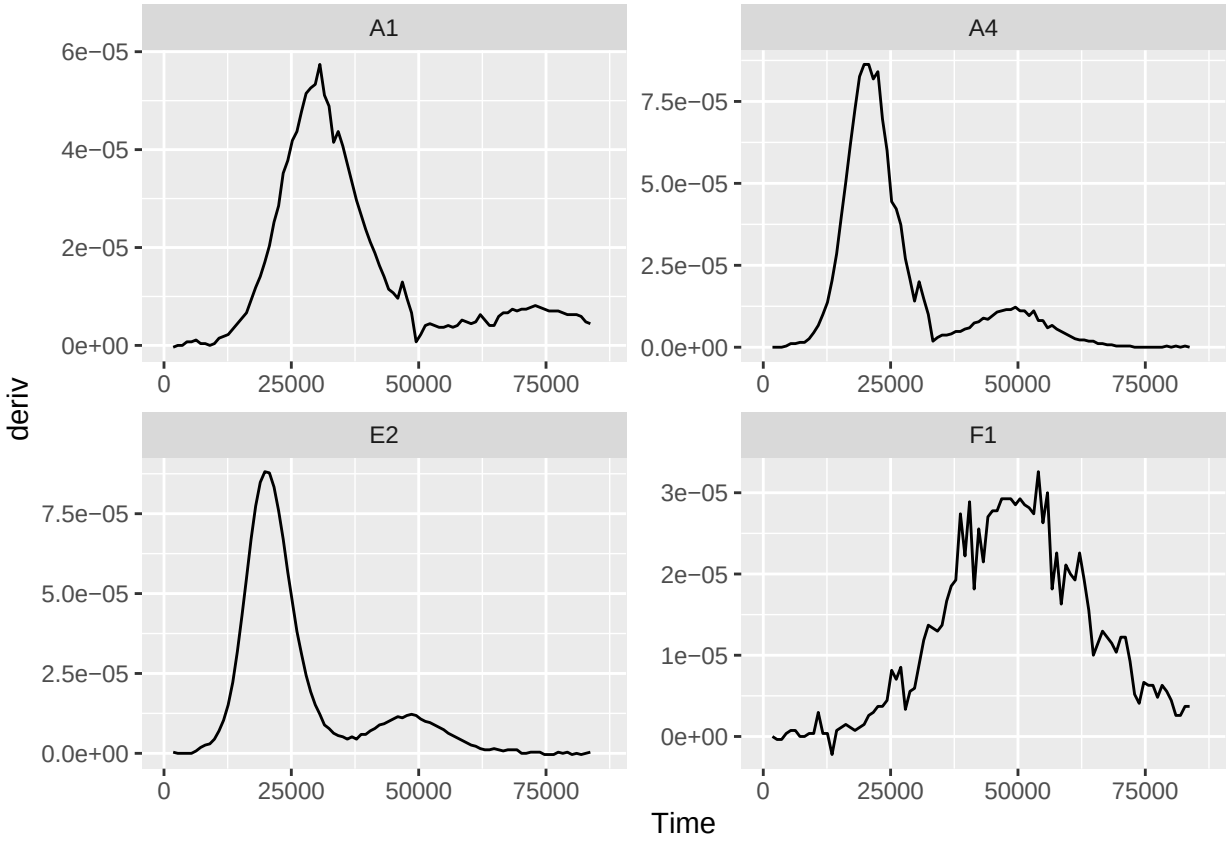
Let's dig into an example: identifying diauxic shifts. To refresh your memory on what we saw in the section [A simple derivative], here's a plot of the derivative of some of the wells over time.

```
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).
```

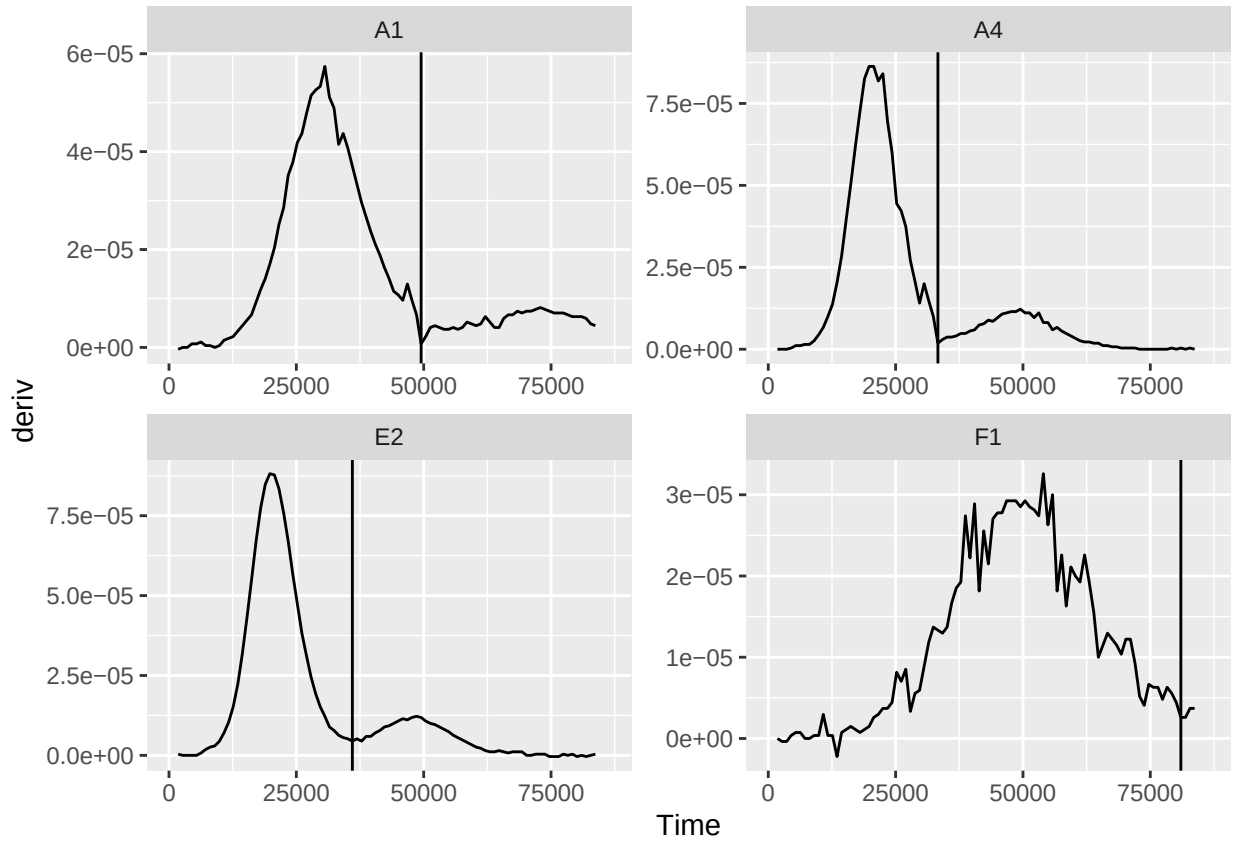



This second, slower, burst of growth after the first wave of growth is common in bacterial growth curves and is called *diauxic growth*.

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)) +
```

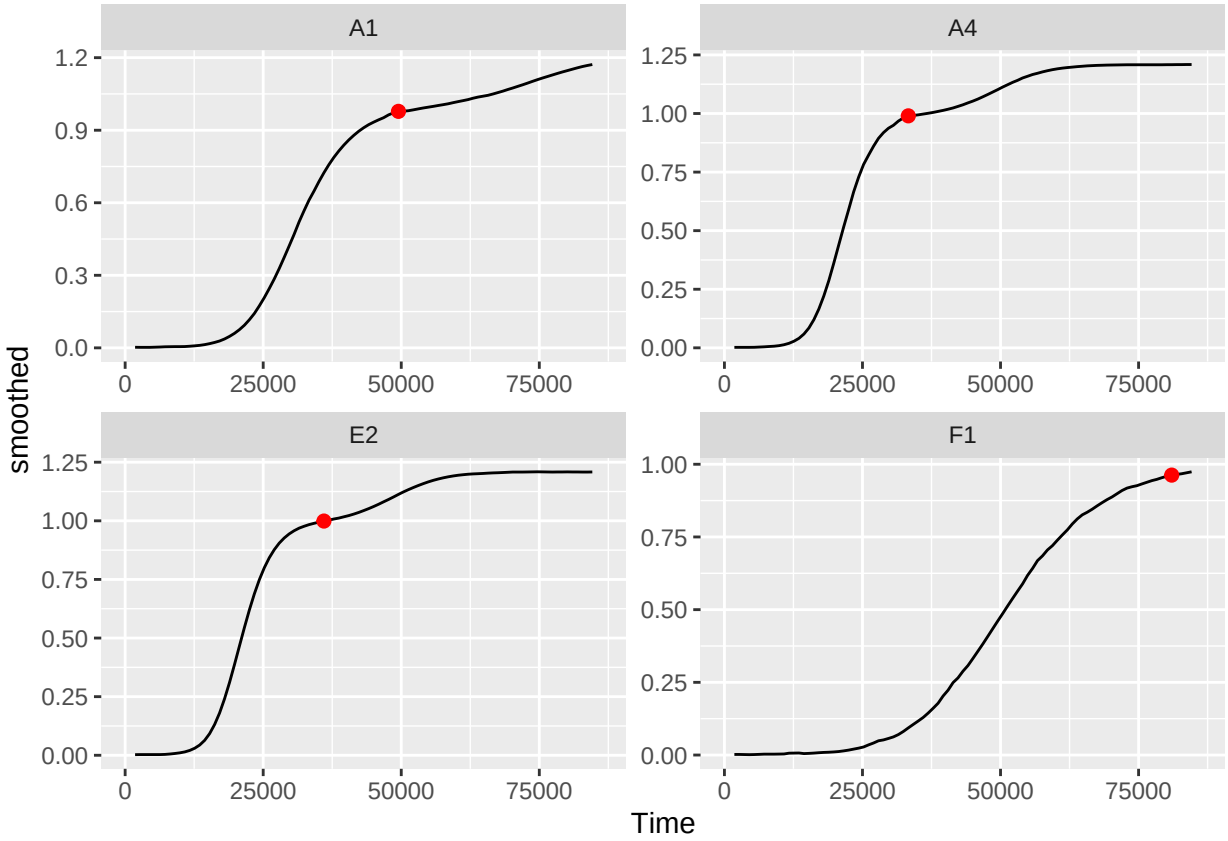
```
  geom_line() +
  facet_wrap(~Well, scales = "free") +
  geom_vline(data = dplyr::filter(ex_dat_mrg_sum, Well %in% sample_wells),
             aes(xintercept = diauxie_time))
#> Warning: Removed 5 row(s) containing missing values (geom_path).
```




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).
```



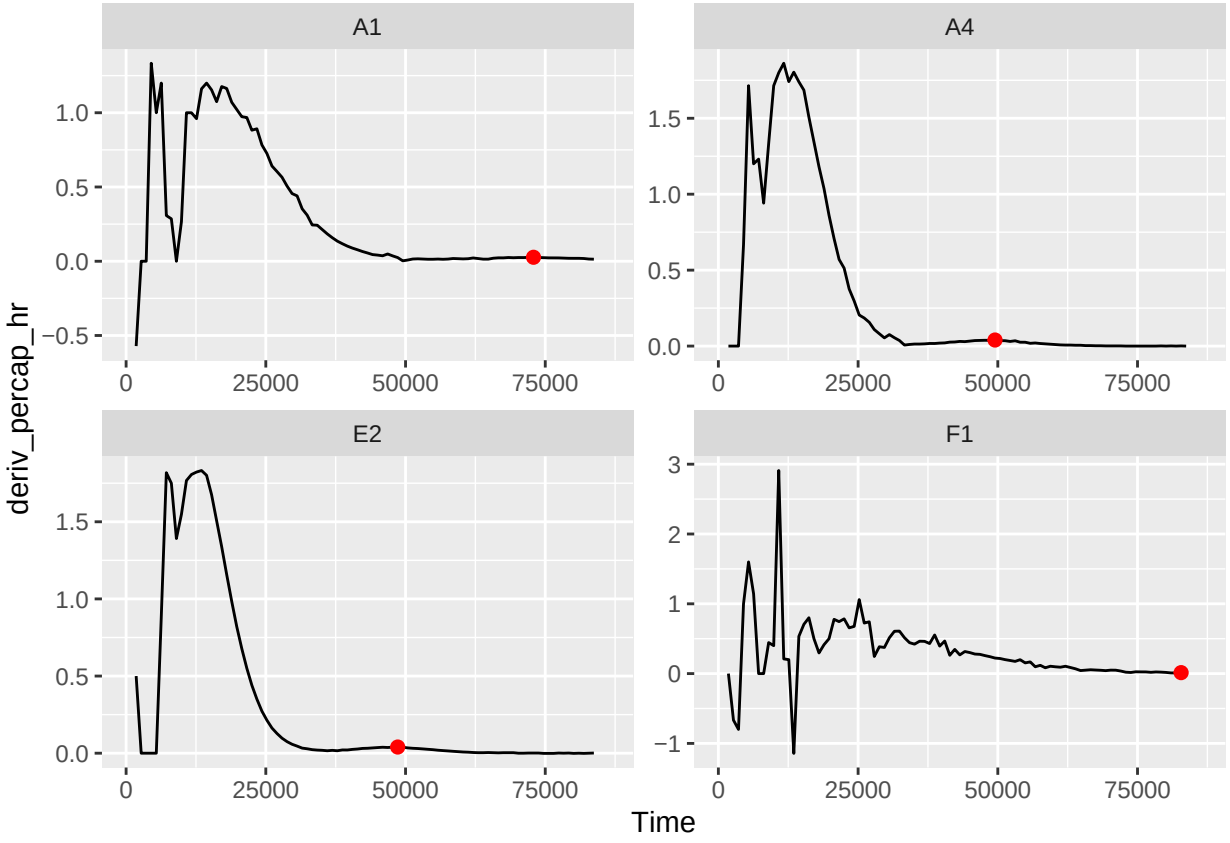

Something that was hard to see on the density plot has now been easily quantified and can be visualized exactly where the shift occurs.

### 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`

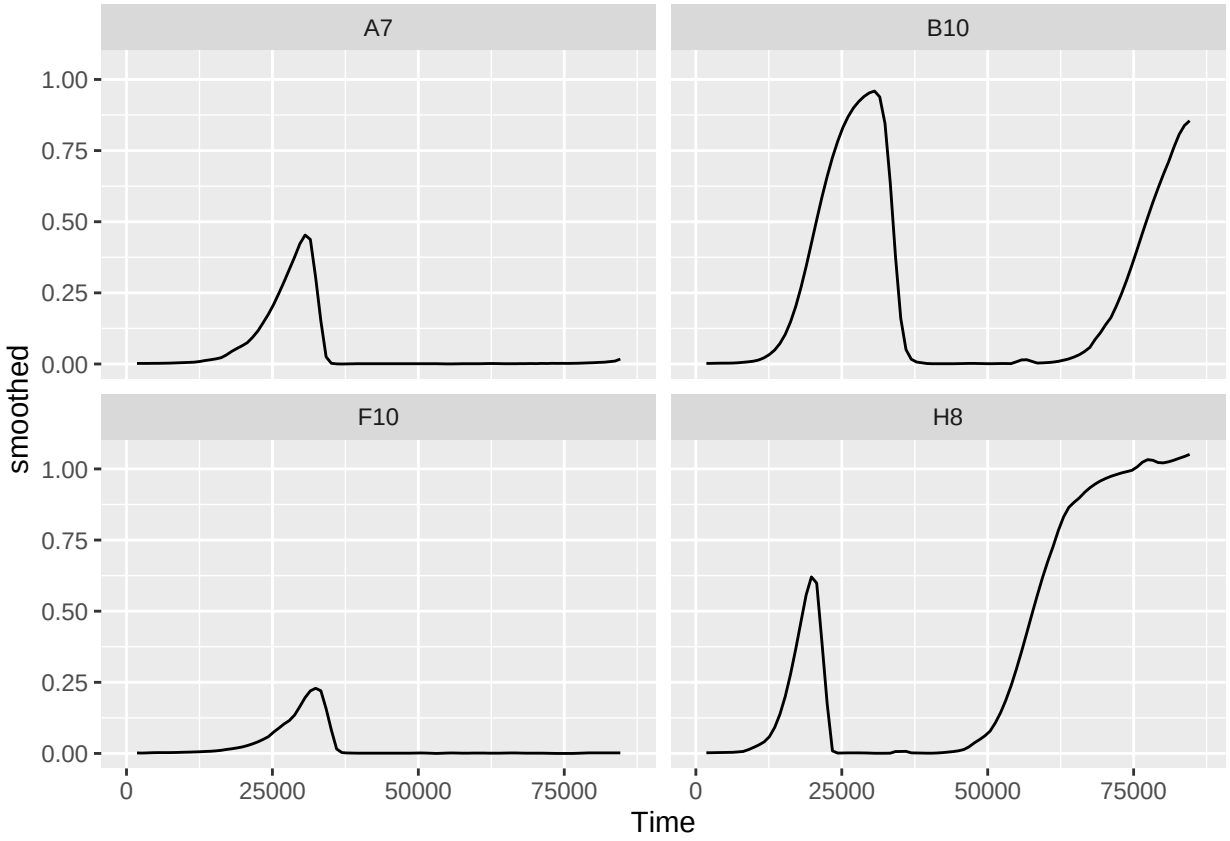
functions `first_below` and `find_threshold_crosses` to find the points when your data crosses user-defined thresholds.

### 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:

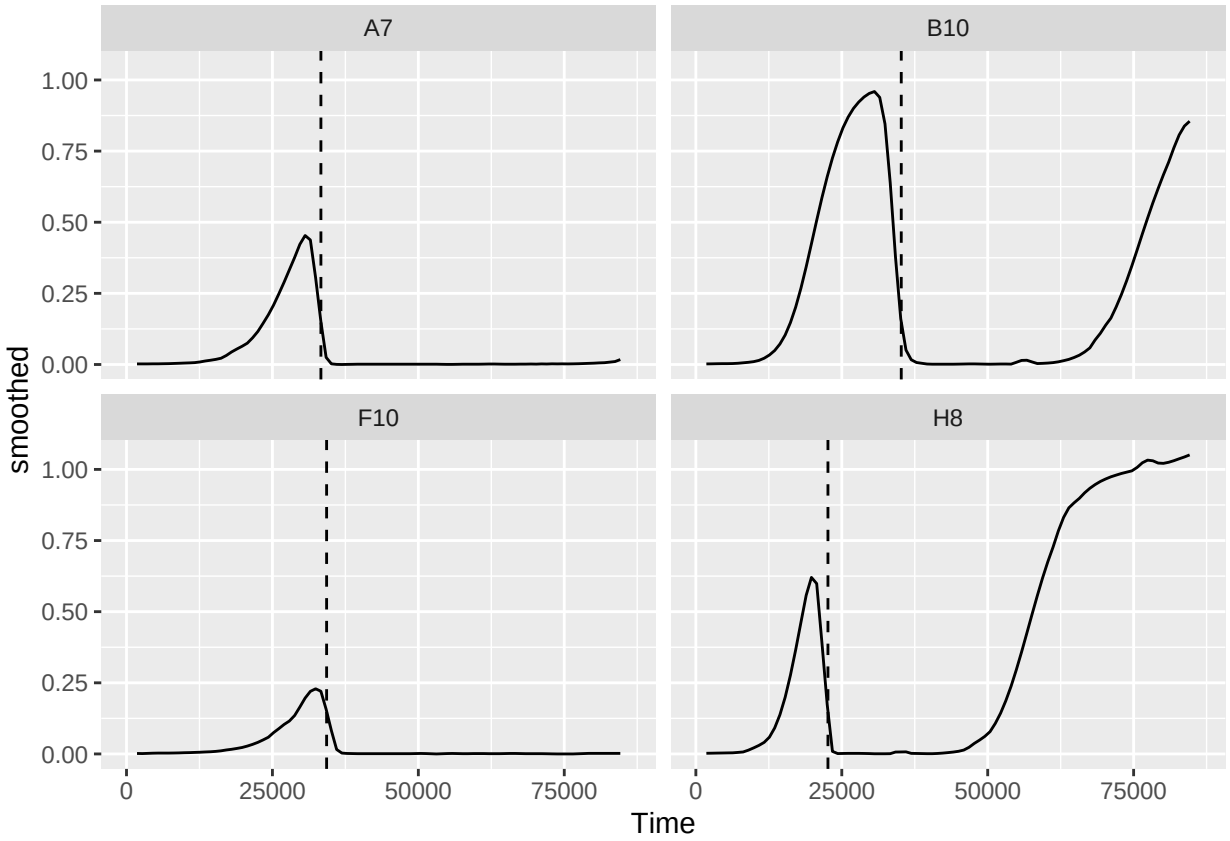
```
sample_wells <- c("A7", "B10", "F10", "H8")

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 x 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?

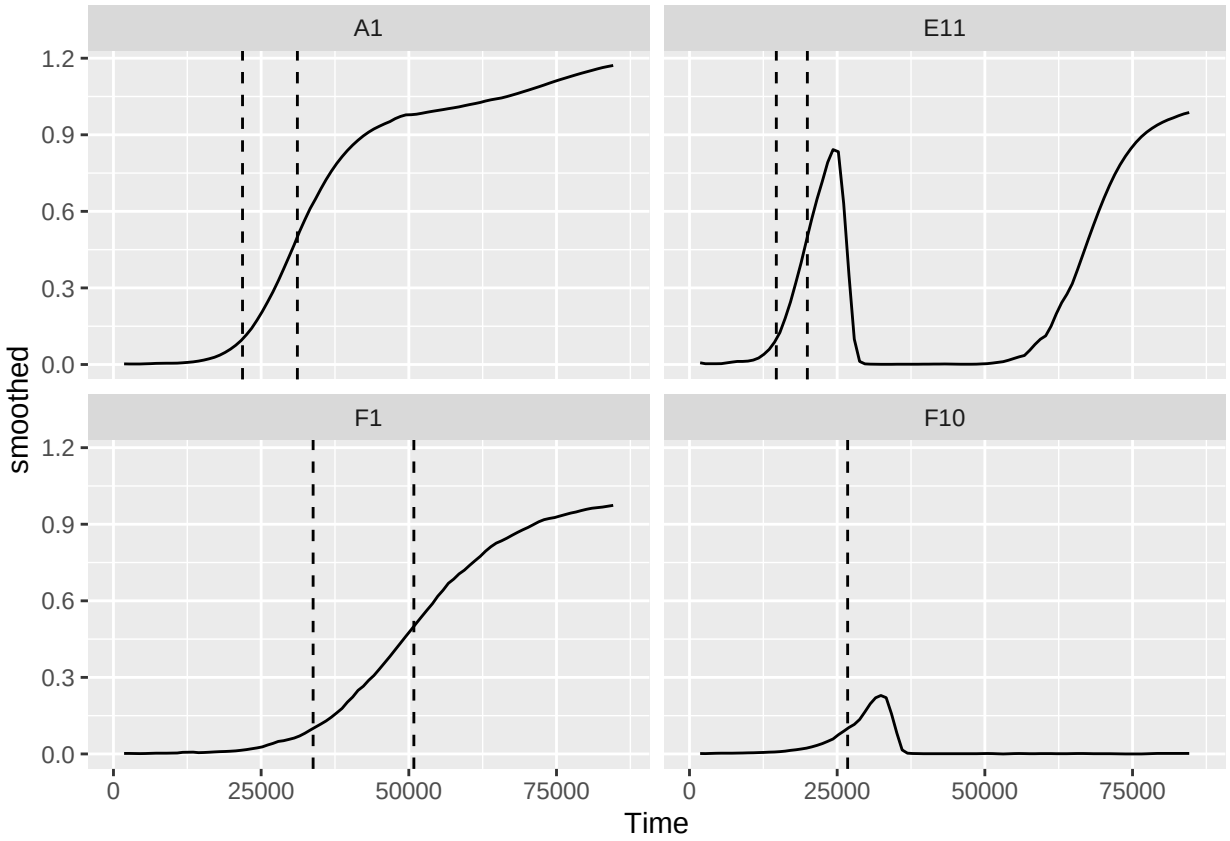
In order to identify these kinds of extrema, we can use the more-general function `find_threshold_crosses`. In fact, `first_below` is really just a special case of `find_threshold_crosses`. Just like `first_below`, `find_threshold_crosses` only requires a `threshold` and a vector of `y` data in which 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 x 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
2. Importing & transforming data
3. Incorporating design information
4. Pre-processing and plotting your data
5. Processing your data
6. **Analyzing your data**
7. Statistics, merging other data, and other resources