# Introduction to using gcplyr

Mike Blazanin

## Contents

## Getting started

`gcplyr` is a package that implements a number of functions to make it easier to import, manipulate, and analyze microbial growth from data collected in multiwell plate readers ("growth curves"). Without `gcplyr`, importing and analyzing plate reader data can be a complicated process that has to be tailored for each experiment, requiring many lines of code. With `gcplyr` many of those steps are now just a single line of code.

This document gives an introduction of how to use `gcplyr` for each step of a growth curve analysis.

To get started, you need your growth curve data file saved to your computer (.csv, .xls, .xlsx, or any other format that can be read by `read.table`).

Users often want to combine their data with some information on the experimental design of their plate(s). You can save this information into a tabular file as well, or you can just keep it handy to enter directly in `R` (see `vignette("gc03_incorporate_designs")`).

Let's get started by loading `gcplyr`. We're also going to load a couple other packages we'll need.

```r
library(gcplyr)

library(dplyr)
library(ggplot2)
```

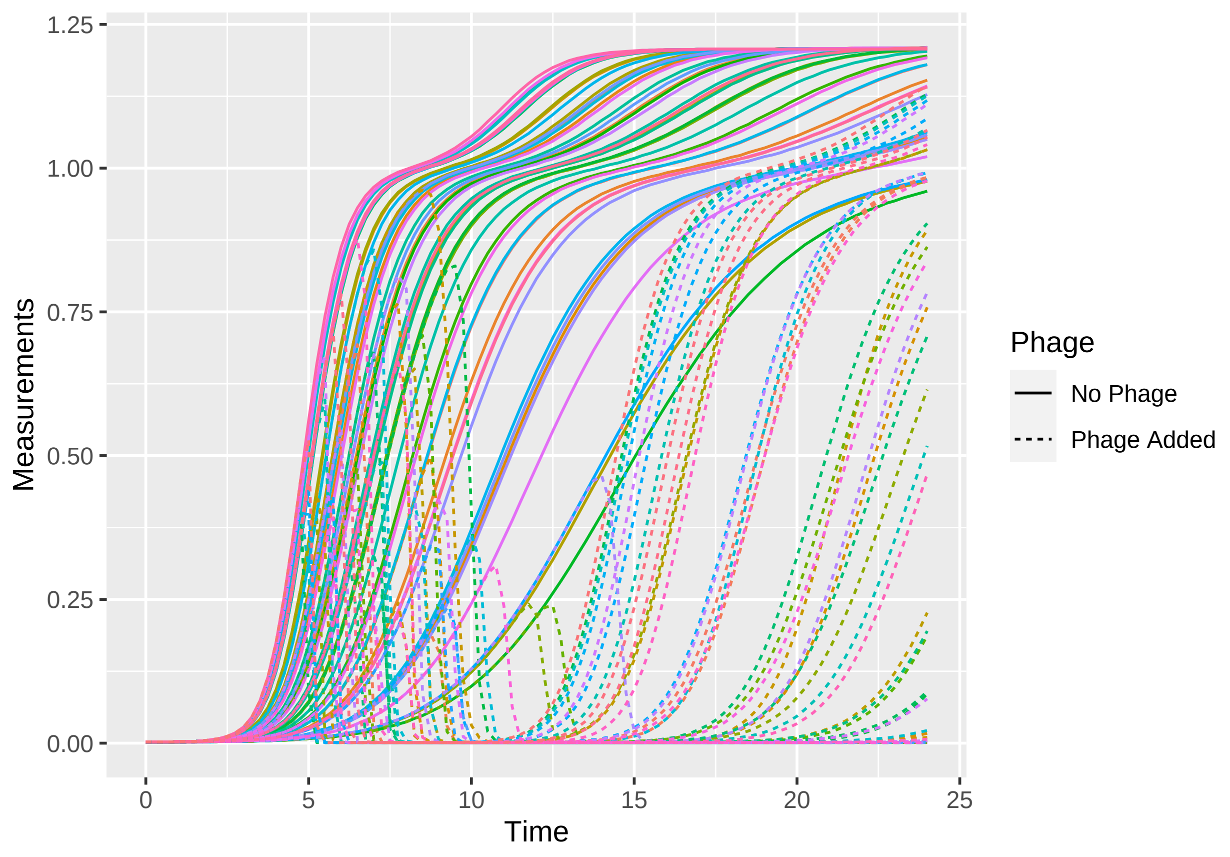
## A quick demo of `gcplyr`

Before digging into the details, here's a simple demonstration of what a final `gcplyr` script can look like. This script:

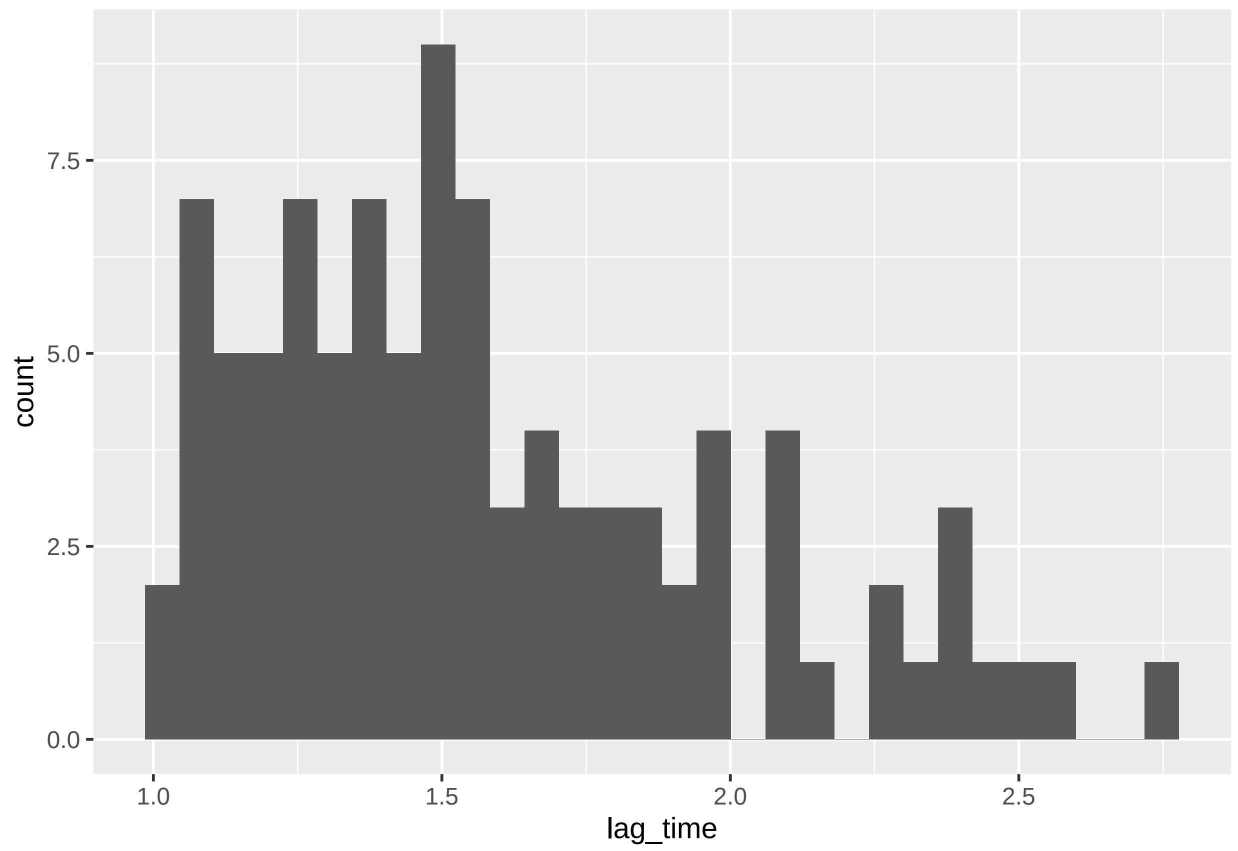
1. imports data from files created by a plate reader
2. combines it with design files created by the user
3. calculates the lag time, maximum growth rate, maximum density, and area-under-the-curve

**Don't worry about understanding all the details of how the code works right now.** Each of these steps is explained in depth in later articles.

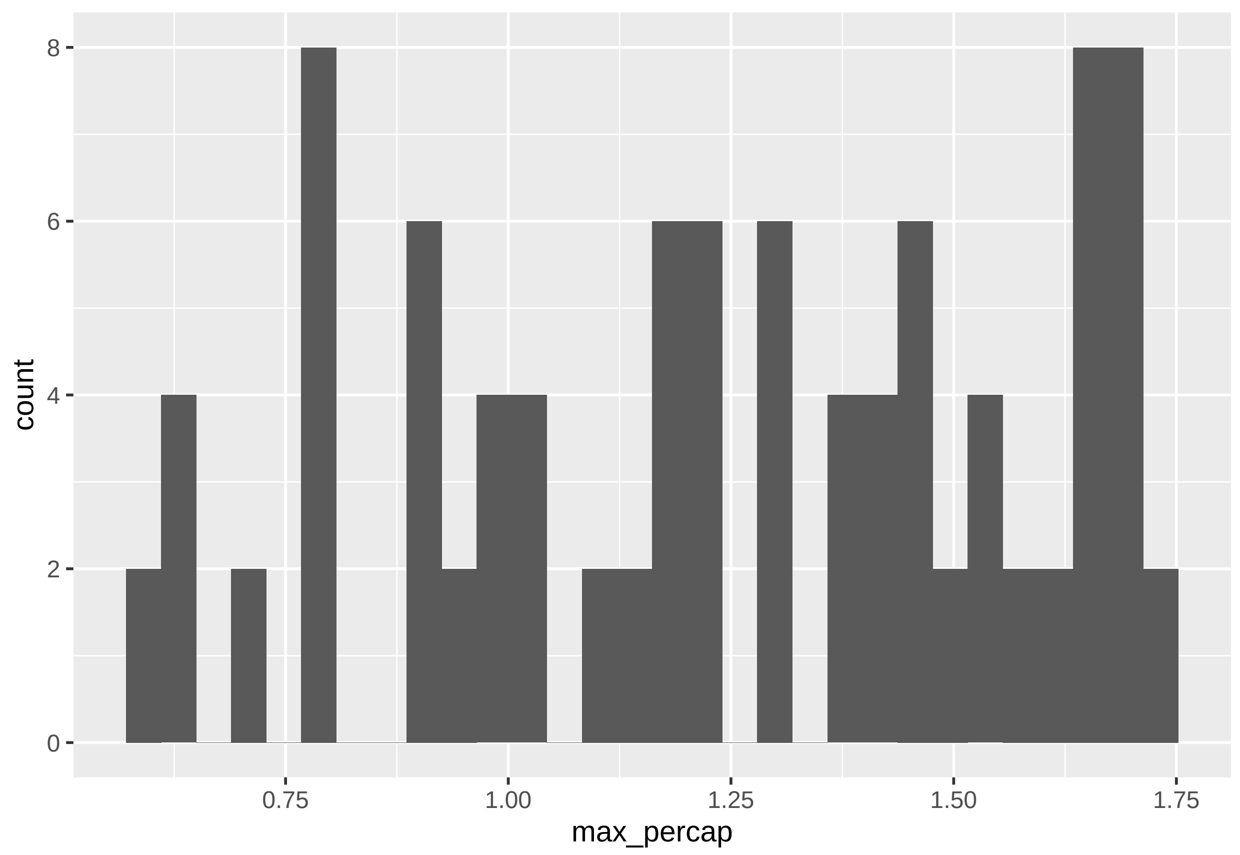
```
#For the purposes of this demo, we have to create our example data and
# design files. Normally, the data would be created by a plate reader, and
# the design would be created by you, the user

#Generate our example data file, widedata.csv
make_example(vignette = 1, example = 1)
#> Files have been written
#> [1] "./widedata.csv"

#Generate our example design files, Bacteria_strain.csv and Phage.csv
make_example(vignette = 1, example = 2)
#> Files have been written
#> [1] "./Bacteria_strain.csv" "./Phage.csv"

# Read in our data
data_wide <- read_wides(files = "widedata.csv")

# Transform our data to be tidy-shaped
data_tidy <-
  trans_wide_to_tidy(wides = data_wide, id_cols = c("file", "Time"))

# Convert our time into hours
data_tidy$Time <- as.numeric(data_tidy$Time)/3600

# Import our designs
designs <- import_blockdesigns(files = c("Bacteria_strain.csv", "Phage.csv"))

# Merge our designs and data
data_merged <- merge_dfs(data_tidy, designs)
#> Joining with `by = join_by(Well)`

# Plot the data
ggplot(data = data_merged,
       aes(x = Time, y = Measurements, color = Well)) +
  geom_line(aes(lty = Phage)) +
  guides(color = "none")
```



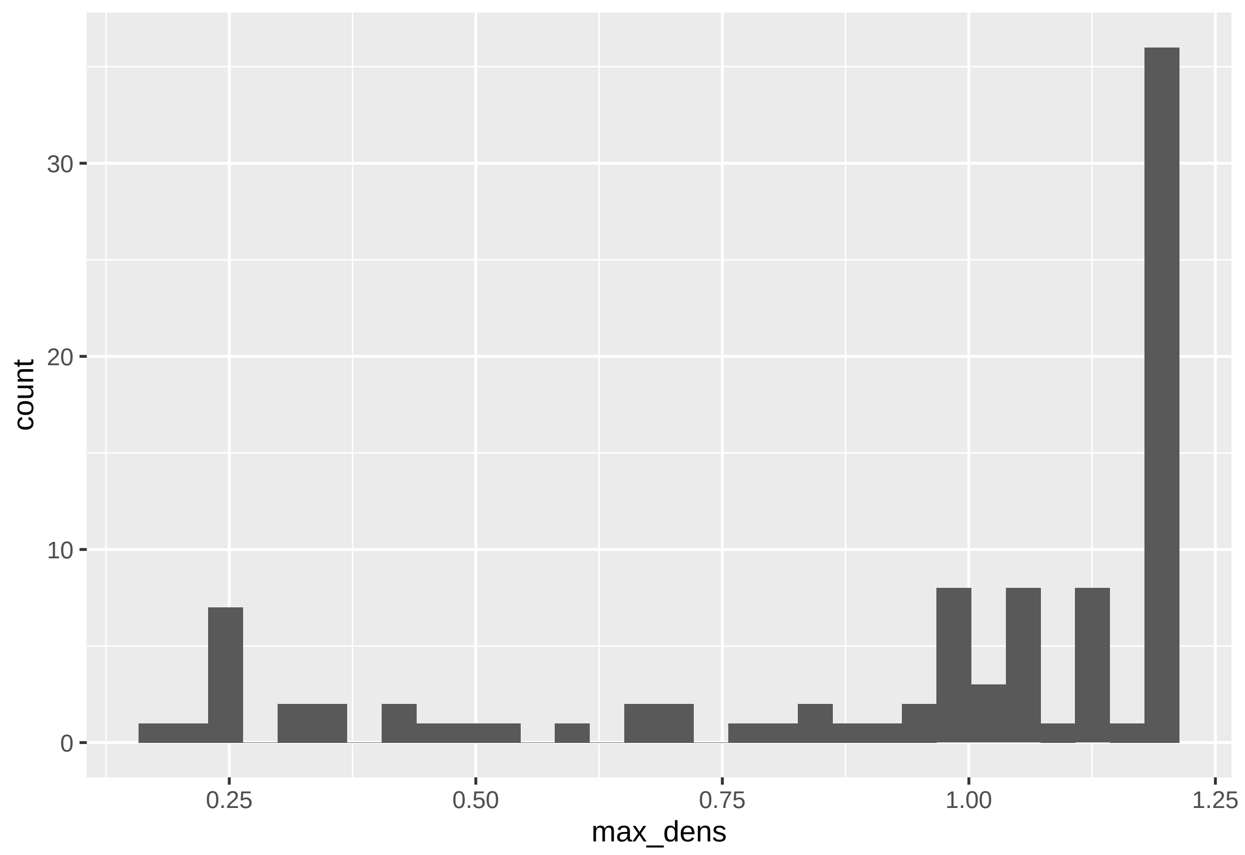
```
# Voila! 8 lines of code and all your data is imported & plotted!

# Calculate the per-capita growth rate over time in each well
data_merged <- mutate(
  group_by(data_merged, Well),
  percap_deriv = calc_deriv(y = Measurements, x = Time, percapita = TRUE,
                            blank = 0, window_width_n = 5))

# Calculate four common metrics of bacterial growth:
#  the lag time, saving it to a column named lag_time
#  the maximum growth rate, saving it to a column named max_percap
#  the maximum density, saving it to a column named max_dens
#  the area-under-the-curve, saving it to a column named 'auc'
data_sum <- summarize(
  group_by(data_merged, Well, Bacteria_strain, Phage),
  lag_time = lag_time(x = Time, y = Measurements, deriv = percap_deriv),
  max_percap = max(percap_deriv, na.rm = TRUE),
  max_dens = max(Measurements),
  auc = auc(y = Measurements, x = as.numeric(Time)))
#> `summarise()` has grouped output by 'Well', 'Bacteria_strain'. You can override using
#> the `.groups` argument.

# Print some of the values
head(data_sum)
#> # A tibble: 6 x 7
#> # Groups:   Well, Bacteria_strain [6]
#>   Well  Bacteria_strain Phage       lag_time max_percap max_dens    auc
#>   <chr> <chr>           <chr>          <dbl>      <dbl>    <dbl>  <dbl>
#> 1 A1    Strain 1        No Phage        2.11      1.00     1.18  15.9
#> 2 A10   Strain 4        Phage Added     1.19      1.43     0.984  5.57
#> 3 A11   Strain 5        Phage Added     1.20      1.47     0.984  5.99
#> 4 A12   Strain 6        Phage Added     1.54      0.789    0.19   0.395
#> 5 A2    Strain 2        No Phage        1.74      1.31     1.21  19.3
#> 6 A3    Strain 3        No Phage        2.14      0.915    1.15  15.1

# Plot the results for each of the metrics
ggplot(data = data_sum, aes(x = lag_time)) +
  geom_histogram()
#> `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



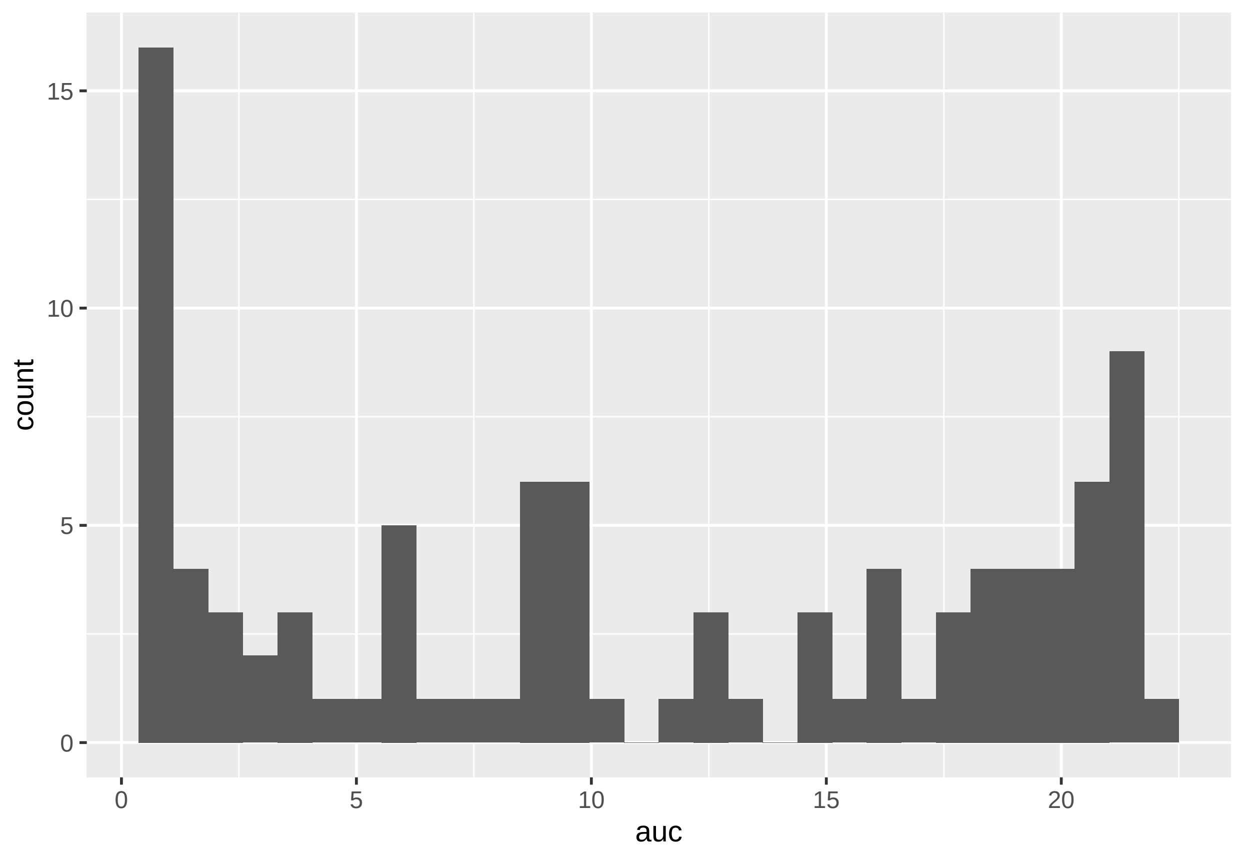
```
ggplot(data = data_sum, aes(x = max_percap)) +
  geom_histogram()
#> `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```
ggplot(data = data_sum, aes(x = max_dens)) +
  geom_histogram()
#> `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```r
ggplot(data = data_sum, aes(x = auc)) +
  geom_histogram()
#> `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



## What's next?

Generally, working with `gcplyr` will follow a number of steps, each of which is likely to be only one or a few lines of code in your final script. We've explained each of these steps in a page linked below. To start, we'll learn how to import our data into `R` and transform it into a convenient format.

1. Introduction: `vignette("gc01_gcplyr")`
2. **Importing and reshaping data: `vignette("gc02_import_reshape")`**
3. Incorporating experimental designs: `vignette("gc03_incorporate_designs")`
4. Pre-processing and plotting your data: `vignette("gc04_preprocess_plot")`
5. Processing your data: `vignette("gc05_process")`
6. Analyzing your data: `vignette("gc06_analyze")`
7. Dealing with noise: `vignette("gc07_noise")`
8. Best practices and other tips: `vignette("gc08_conclusion")`
9. Working with multiple plates: `vignette("gc09_multiple_plates")`
10. Using make_design to generate experimental designs: `vignette("gc10_using_make_design")`