Particle Interception as a Function of Reynolds Number and Collector Density

Version 2.0

Jordan Wingenroth

8/28/2020

Contents

Objective(s)

In the interest of documenting our analytic methods, I've decided to make one comprehensive, hopefully lay-readable document including all steps from our raw laboratory data to the model results. I won't be able to include the turbulence analyses in time for our meeting (8/27), but that required Laurel's Matlab script anyways.

Data Tidying

CHANGES FROM VERSION 1 (8/26)

I pared back this section to reduce the page count. Please refer to the original version to see visualizations and more detailed steps and notes.

Time-decay runs:

We entered data into Google Sheets as we processed samples in the lab. I downloaded .csv versions of these data in 2019, then made a few modifications as I explored confounders. Today, I added data from runs done in my absence to this dataset. Field names, date syntax, and other conventions weren't kept consistent, so some data wrangling is necessary. I also removed the blank data at the beginning of some files manually.

```r
library(tidyverse)
```

```r
pumpfiles <- list.files("../data/peristaltic pumps/")
trapfiles <- list.files("../data/sediment traps/")
```

```r
pumpdate <- str_sub(pumpfiles,0,6)
trapdate <- str_sub(trapfiles,0,6)
```

In all, five runs are missing sediment trap data, meaning they won't be used in our final, published analyses.

Let's turn our attention to the suspended concentration data (i.e., pump data).

```r
pumpdata <- lapply(pumpfiles, function(x) read_csv(paste0("../data/peristaltic pumps/",x)))

names(pumpdata) <- pumpdate

x <- pumpdata

tidypump <- lapply(seq_along(x), function(i) {
  select(x[[i]],
         loc = Location,
         ht = Height,
         t = `time series`,
         mvc = contains("(ppm)")) %>%
    filter(t < 21) %>% #filter a few timepoints outside the normal window
    mutate(t = (t-min(t)+1)*300, #convert from timestep to seconds
           mvc = as.numeric(mvc),
           date = as.numeric(names(x)[[i]])) %>%
    filter(mvc<80, mvc>8) #outliers were removed based on the residual graph
  }
  )

pump <- bind_rows(tidypump)
```

So now we have a long table of our pump data (time × concentration) stratified by run (i.e., treatment), height, and upstream/downstream location.

Aside from a few runs where **part** or **all** of the data don't follow the decay pattern, the results look pretty good! If we repeated runs as necessary to replace those with erroneous data, that shouldn't be a problem. Here I'll narrow down our dataset by knocking out runs with clear issues.

```r
pump <- pump %>%

#first 3 runs had starting sediment mass of 100g rather than 200g

    filter(date > 181005) %>%

#we can't use runs without sediment mass

    filter(as.character(date) %in% trapdate)
```

So this leaves 20 rows. Let's join this to our run metadata table.

```r
metadata <- read_csv("../data/run_metadata.csv")

pump <- left_join(pump, metadata, by = "date")
```

Now then, let's remove the biofouled runs and a couple runs with other issues, which won't be used in assessing our primary hypothesis about Reynolds number and collector density, and we'll see what we're left with.

```r
pumpfinal <- pump %>%
  filter(growth_days==0, date != 190417, date != 190802, # leaving out old control
         dowel_density != "0232")
```

There are certainly no trends jumping out immediately, which isn't necessarily worrisome since exponential decay is hard to compare by eye. More on that later though.

For the sediment traps, data import and wrangling follows a similar structure to the pump data.

```r
trapdata <- lapply(trapfiles, function(x) read_csv(paste0("../data/sediment traps/",x)))

names(trapdata) <- trapdate

x <- trapdata

tidytrap <- lapply(seq_along(x), function(i) {
  select(x[[i]], station = 1, pre = contains("pre"), post = contains("post"), sed = contains("sed")) %>
  mutate(date = names(x)[i]) %>%
  mutate_at(vars(pre,post,sed,date), as.numeric)
  }
  )

trap <- bind_rows(tidytrap) %>%
  filter(!is.na(sed))
```

So, across 21 runs, we had 186 samples. We were set up to collect 9 per run, but traps broke, filters slipped and spilled sediment, et cetera. $186/21 = 8.8571429$, so our success rate was actually pretty good, although some of those measurements were probably erroneous despite being measured.

```r
trapfinal <- left_join(trap, metadata, by = "date") %>%
  filter(date %in% pumpfinal$date)
```

Velocity experiment:

We estimated Reynolds number from a flow velocity experiment in the open channel conducted on Jan 28, 2019. The Vectrino data is in commit `e78357` and the R code for the regression in commit `17c8af` in the esdlflume Github repo (Thanks Candace!).

There's an image in version 1 (and in /pics) but the linear regression is simply $u = 0.00081282 + 0.00195723f$, where u is flow velocity (m/s) and f is pump frequency (Hz). Hence our velocities at 10, 20, and 30 Hz, were 0.0203851, 0.0399574, and 0.0595297 respectively.

It should be noted that the back-of-envelope estimate of $v = f/500$ is very close for our range, probably by design. For the regression, $R^2 > .99$.

Flume volume experiment:

The test section is modeled as a rectangular prism, making calculation of volume trivial. However, the water in the flume is in an irregular form. In order to correct for the time water spends flowing outside the test section, where collectors are not acting on particles, we must estimate the total volume of water. We did so using a simple integration of the volumetric flow rate at the drain. We ran an experiment previously in 2018 or 2019, but my physics common sense lapsed: We picked the drain hose up off the ground to take measurements, then stuck it back down in the drain for most of the flow time. This led to an underestimate of about 5-10%. I re-did the analysis. 1st through 4th degree polynomial regression all yield the same volume +/- <0.5% (code in this repo, under /code)

I rounded off to 2.43 m^3. The test section has been measured as $1.95 \times .6 \times .4 = 0.468$ m^3. Hence the corrective factor is 5.1923077.

Modelling for ECE%

So at this point, we really only need `pumpfinal`, `trapfinal`, maybe `metadata` for good measure, the frequency-to-velocity regression coefficients, and individual constants about the physical setup such as dowel diameter, flume volume, starting sediment mass, dynamic viscosity etc.

I'll also mention here that our model is based on the general equation:

$$k_{tot} = k_s + k_c + k_{bg}$$

where the k's represent total decay rate and portions of it due to settling, collection, and (background) settling in the rest of the flume outside the test section, respectively.

Uncertainty propogation function: mutate_with_error

Here is a function found on StackExchange.com, which performs error propogation in R using derivatives. Error values should be named "dx", with x being the corresponding variable name.

```
mutate_with_error = function(.data, f) {
  exprs = list(
    # expression to compute new variable values
    deparse(f[[3]]),

    # expression to compute new variable errors
    sapply(all.vars(f[[3]]), function(v) {
      dfdp = deparse(D(f[[3]], v))
      sprintf('(d%s*(%s))^2', v, dfdp)
    }) %>%
      paste(collapse='+') %>%
      sprintf('sqrt(%s)', .)
  )
  names(exprs) = c(
    deparse(f[[2]]),
    sprintf('d%s', deparse(f[[2]]))
  )

  .data %>%
    # the standard evaluation alternative of mutate()
    mutate_(.dots=exprs)
}
```

k_tot (total time-decay rate)

We estimate `k_tot` from our pump data by linear regression of `log(mvc)`:

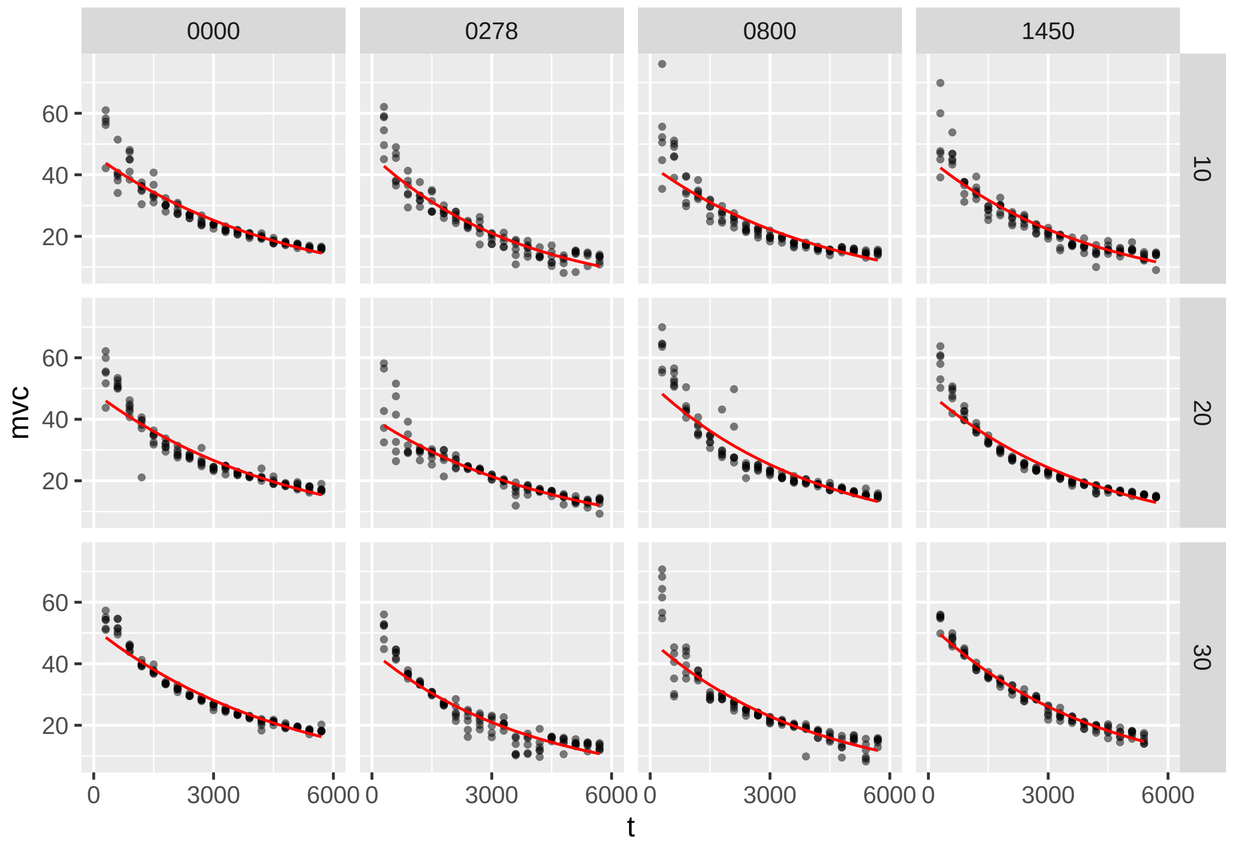
```
library(lme4)

fits <- lmList(data = pumpfinal, log(mvc)~t | date)

summary(fits)
```

```
## Call:
##   Model: log(mvc) ~ t | NULL
##    Data: pumpfinal
##
## Coefficients:
##    (Intercept)
##        Estimate Std. Error  t value Pr(>|t|)
## 181115 3.973846 0.02576910 154.2097        0
## 190321 3.943840 0.02502525 157.5944        0
## 190506 3.785685 0.02502525 151.2746        0
## 190612 3.867590 0.02502525 154.5475        0
## 190729 3.840790 0.02544371 150.9525        0
## 190905 3.836769 0.02521758 152.1466        0
## 190926 3.701444 0.02544066 145.4932        0
## 191029 3.767874 0.02503661 150.4946        0
## 200211 3.948348 0.02502525 157.7746        0
## 200303 3.815678 0.02502525 152.4731        0
## 200313 3.889987 0.02502525 155.4425        0
## 200902 3.888671 0.02502525 155.3899        0
##    t
##             Estimate   Std. Error   t value      Pr(>|t|)
## 181115 -0.0002381660 7.935532e-06 -30.01260 1.591279e-151
## 190321 -0.0002025636 7.316199e-06 -27.68700 1.364456e-133
## 190506 -0.0002477993 7.316199e-06 -33.86995 8.349205e-182
## 190612 -0.0002455980 7.316199e-06 -33.56907 1.987260e-179
## 190729 -0.0002045307 7.409333e-06 -27.60447 5.819498e-133
## 190905 -0.0002643807 7.525614e-06 -35.13078 8.871278e-192
## 190926 -0.0002130646 7.405203e-06 -28.77229 6.415321e-142
## 191029 -0.0002221626 7.317278e-06 -30.36137 3.064143e-154
## 200211 -0.0002397320 7.316199e-06 -32.76729 4.194466e-173
## 200303 -0.0002382292 7.316199e-06 -32.56189 1.739496e-171
## 200313 -0.0002333927 7.316199e-06 -31.90081 2.748612e-166
## 200902 -0.0002022880 7.316199e-06 -27.64933 2.645878e-133
##
## Residual standard error: 0.1283571 on 1329 degrees of freedom
```

The `t` coeffient estimates are the `k_tot` values in units s^{-1}, no transformation needed aside from a sign change ($\log(\bar{\phi}) = \log(\phi_0 e^{-kt}) = log(\phi_0) - kt$).

All the t values are high, though the coefficient t-values are relative to 0, meaning they're really just expressing that we're very certain we did indeed add sediment (phew!). But let's see what the fits look like with the data:

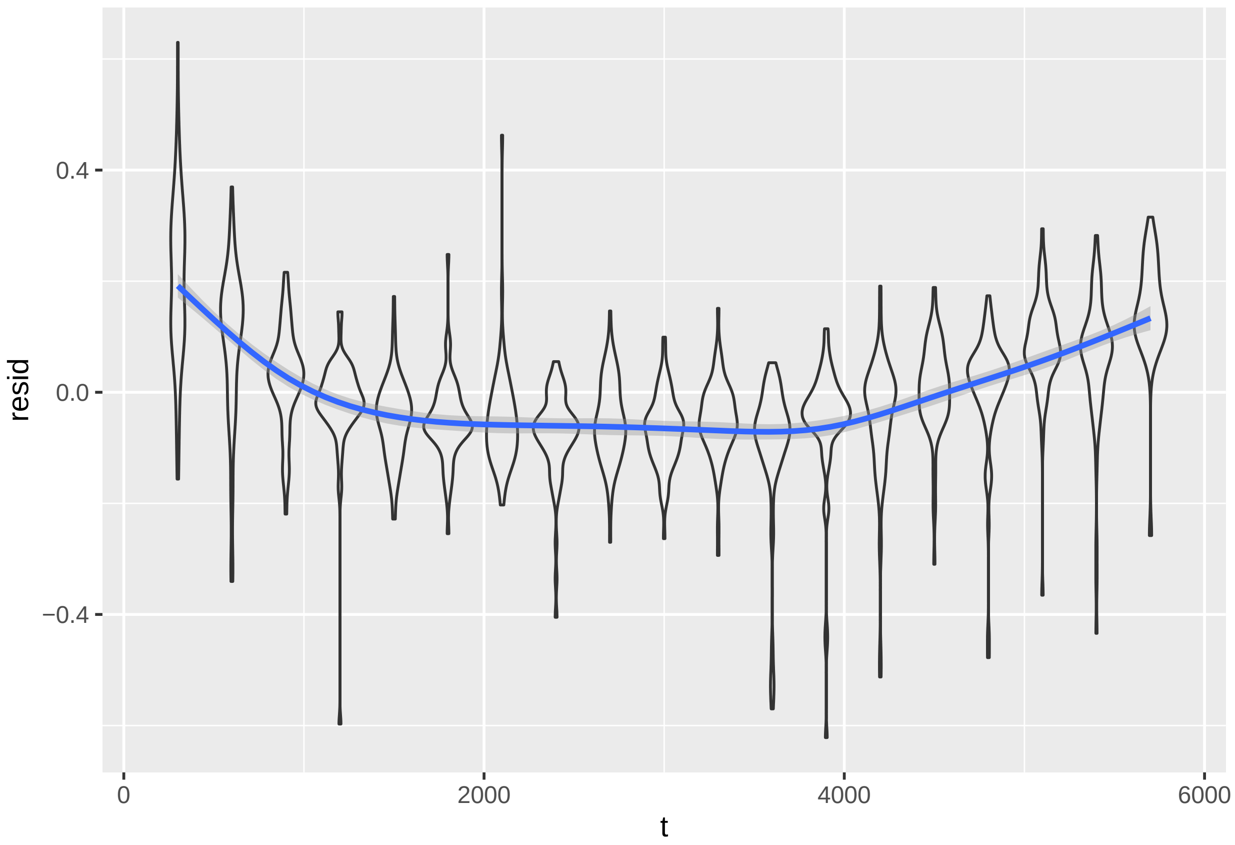
```
cbind(pumpfinal, pred = predict(fits)) %>%
  ggplot() +
  geom_point(aes(x = t, y = mvc), size = .75, alpha = .5) +
  geom_line(aes(x = t, y = exp(pred)), color = "red") +
  facet_grid(pump_freq~dowel_density) +
  scale_x_continuous(breaks = c(0,3000,6000), limits = c(0,6000))
```



As I noticed back in early 2019, these curves seem to hint at a U-shaped pattern in the residuals.

```r
cbind(pumpfinal, resid = residuals(fits)) %>%
  ggplot(aes(x = t, y = resid, group = t)) +
  geom_violin(alpha = .05) +
  geom_smooth(inherit.aes = F, aes(x = t, y = resid))
```

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



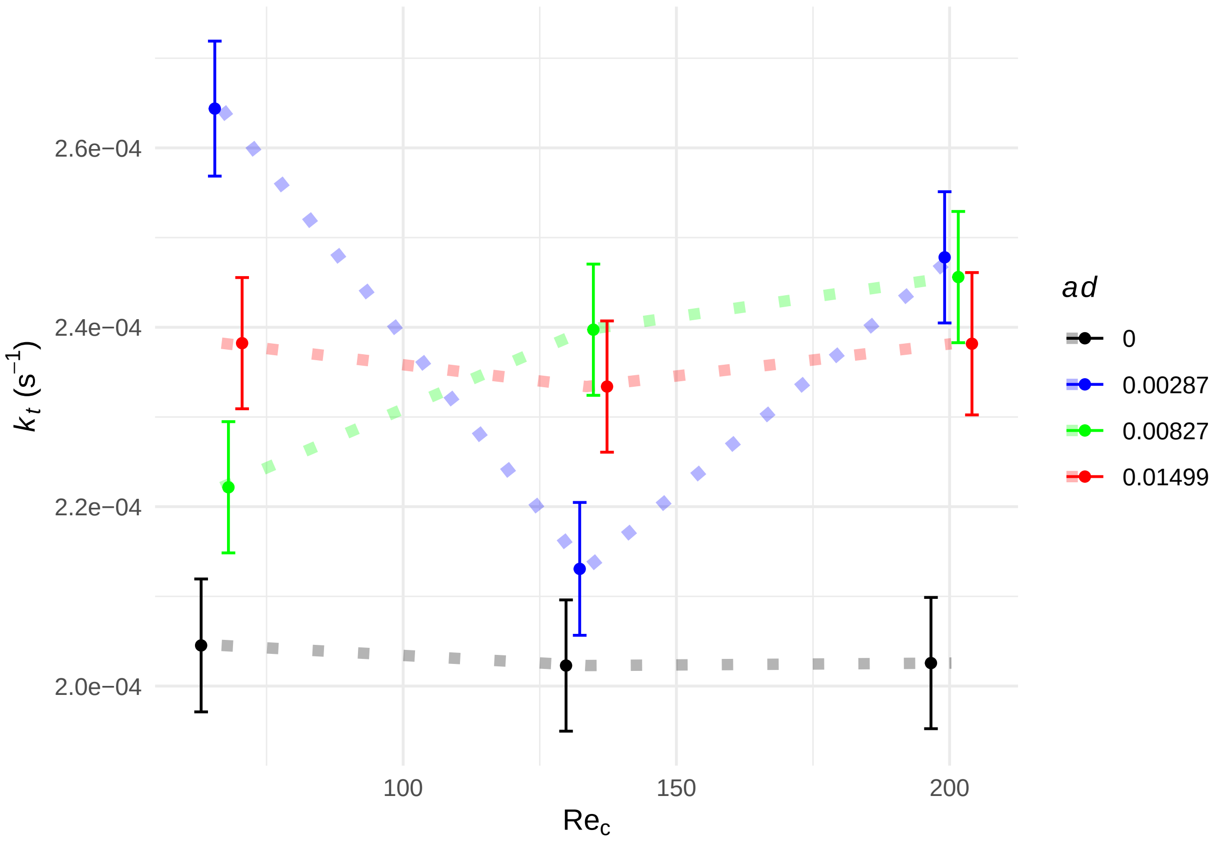
I think this is simply an artifact of nonlinear processes not accounted for in the exponential model. As an example of what I mean, let's imagine there were a hidden variable called "stickiness" differentiating walnut shell particles from one another. If by way of accellerated settling due to floculation, or accellerated collection, those particles were disproportionately removed from suspension early in the timespan of the experiment, the average physical properties of the suspended sediment would change over its course. It also seems likely to me that as sediment adheres to collectors, their efficiency decreases on account of their having less open surface area.

Whatever the explanation is, we (and other researchers using the exponential model) assume it is uniform across our independent variables. I don't know how safe that assumption is, but that'll have to wait for a future paper. It might be testable from our data but `n = 12` seems a bit of a small sample. Then again, we've probably done more like `n = 100` runs over the entire history of the project.

Aside from that U-curve, the fit seems reasonably good. **This residual plot was used to remove the most obvious outliers**, though those in the bottom right (say, those satisfying $(y + 0.4) < (0.1/1000)(x - 3500)$) might reasonably be considered outliers too. We might be in the gray area in terms of data-massaging at that point. Any formal outlier removal included in the manuscript will have to follow a *de rigueur*, citable method of course.

So then, k_tot as a function of our independent vars:

```
fitdata <- coefficients(fits) %>%
  mutate(k_t = -t) %>%
  cbind(dk_t = summary(fits)[[4]][,,'t'][,2], date = as.numeric(row.names(coefficients(fits)))) %>%
  left_join(metadata, by = "date") %>%
  mutate(ad = as.numeric(dowel_density)*.003175^2*2/1.95, Re = pump_freq/500*.003175/9.509e-7) # DOWEL

fitdata %>%
  ggplot(aes(x = Re, y = k_t, ymin = k_t - dk_t, ymax = k_t + dk_t, color = as.character(round(ad,5))))
  geom_line(lty = "dotted", size = 2, alpha = .3) +
  geom_point(position = position_dodge(width = 10)) +
  geom_errorbar(width = 10, position = "dodge") +
  scale_color_manual(values = c("black","blue","green","red")) +
  scale_y_continuous(labels = function(x) format(x, scientific = TRUE)) +
  labs(y = expression(paste(italic(k[t])," (",s^{-1},")")), x = expression(Re[c]), color = expression(i
  theme_minimal()
```



k_s (time-decay due to settling)

$$m_s = \frac{k_s}{k_{tot}}(1 - e^{-kT})m_0$$

Or, expressed for k_s,

$$k_s = \frac{m_s}{m_0}(\frac{1}{1 - e^{-kT}})k_{tot}$$

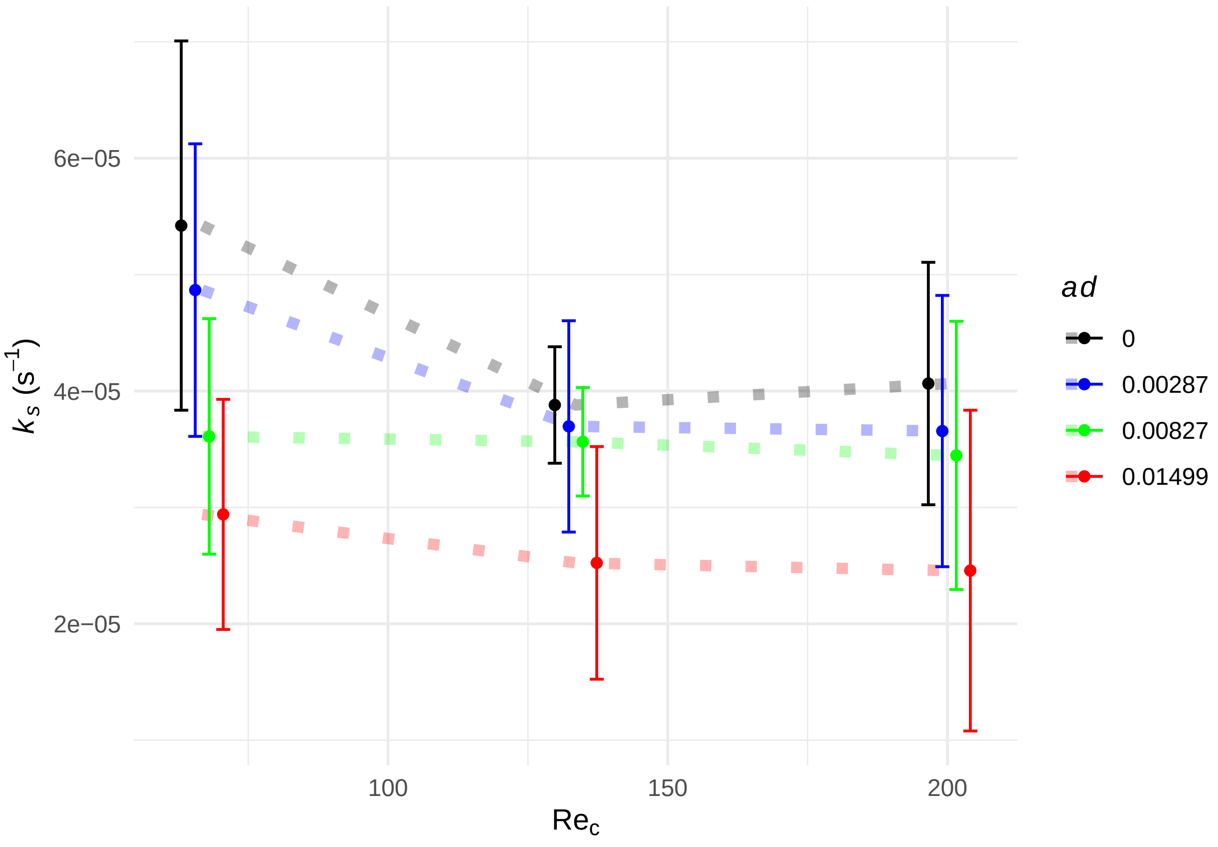
where m_s is mass settled in the test section (estimated from sediment traps), m_0 is starting sediment mass (200 g for this study), and T is the total exposure time of the sediment traps (100 min, i.e., 6000 s, for this study).

```
final <- trapfinal %>%
  group_by(date) %>%
  summarise(m_trap = mean(sed)/1000,
            dm_trap = sd(sed)/1000) %>% #average sediment in trap converted from mg to g
  mutate_with_error(m_s ~ m_trap*1.95*.6/(3.14159*.0127^2)) %>% #*A_TS/a_trap
  left_join(fitdata, by = "date") %>%
  mutate_with_error(c_s ~ m_s/200/(1-exp(-k_t*6000))) %>% #intermediate step to squash error
  mutate_with_error(k_s ~ c_s*k_t)
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## Warning: `mutate_()` is deprecated as of dplyr 0.7.0.
## Please use `mutate()` instead.
## See vignette('programming') for more help
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```
final %>%
  ggplot(aes(x = Re, y = k_s, ymin = k_s - dk_s, ymax = k_s + dk_s, color = as.character(round(ad,5)))) 
  geom_line(lty = "dotted", size = 2, alpha = .3) +
  geom_point(position = position_dodge(width = 10)) +
  geom_errorbar(width = 10, position = "dodge") +
  scale_color_manual(values = c("black","blue","green","red")) +
  scale_y_continuous(labels = function(x) format(x, scientific = TRUE)) +
  labs(y = expression(paste(italic(k[s])," (",s^{-1},")")), x = expression(Re[c]), color = expression(i
  theme_minimal()
```



k_s seems negatively related to both dowel density and flow velocity.

k_bg (time-decay due to settling in the rest of the flume)

In control runs, $k_c = 0$, so we can estimate the decay rate in the rest of the flume as $k_{bg} = k_{tot} - k_s$

```
bgvals <- final %>%
  filter(dowel_density=="0000") %>%
  mutate_with_error(k_bg ~ k_t - k_s) %>%
  select(pump_freq, k_bg, dk_bg) %>%
  arrange(pump_freq)

bgvals
```

```
## # A tibble: 3 x 3
##   pump_freq     k_bg      dk_bg
##       <dbl>    <dbl>      <dbl>
## 1        10 0.000150 0.0000175
## 2        20 0.000163 0.00000886
## 3        30 0.000162 0.0000127
```

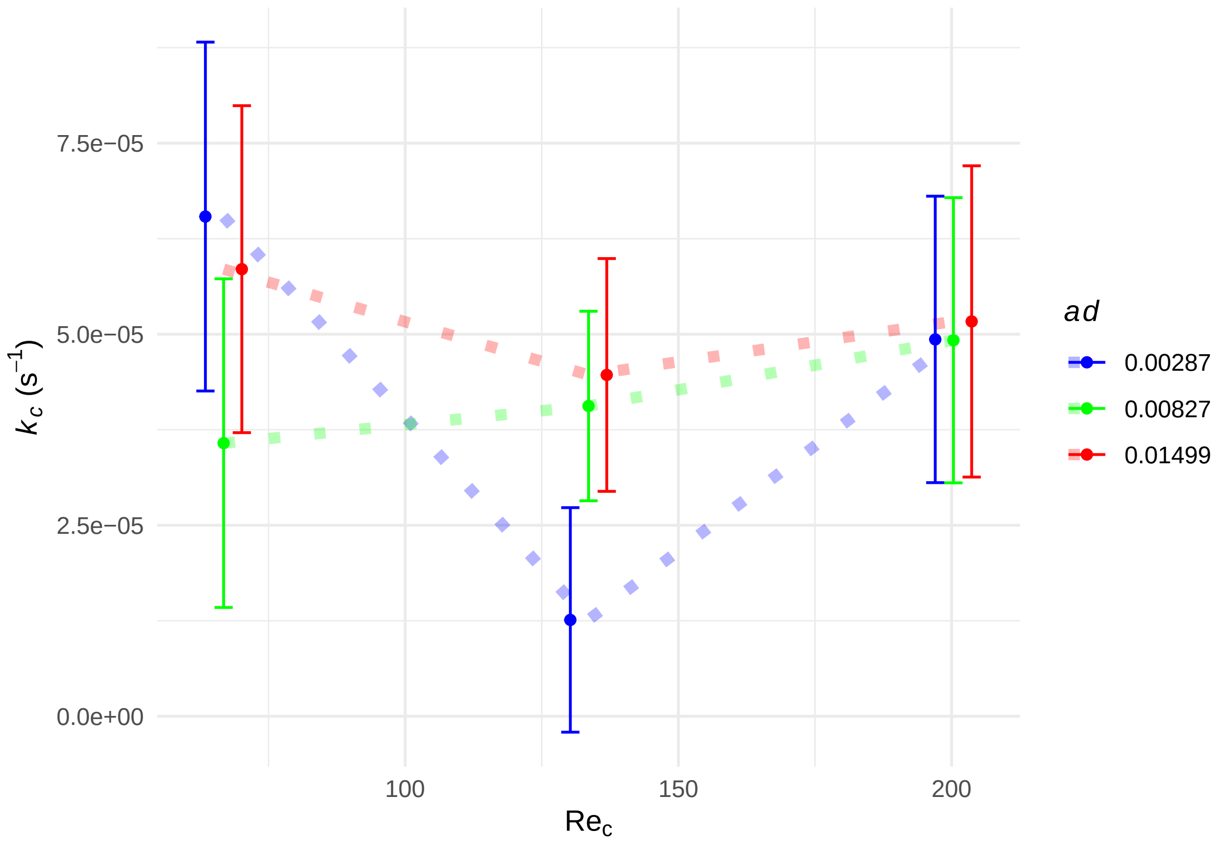
To add it to the table:

```
final <- left_join(final, bgvals, by = "pump_freq")
```

k_c (time-decay due to collection)

So, now that we have our values in the table, k_c is just a subtraction away:

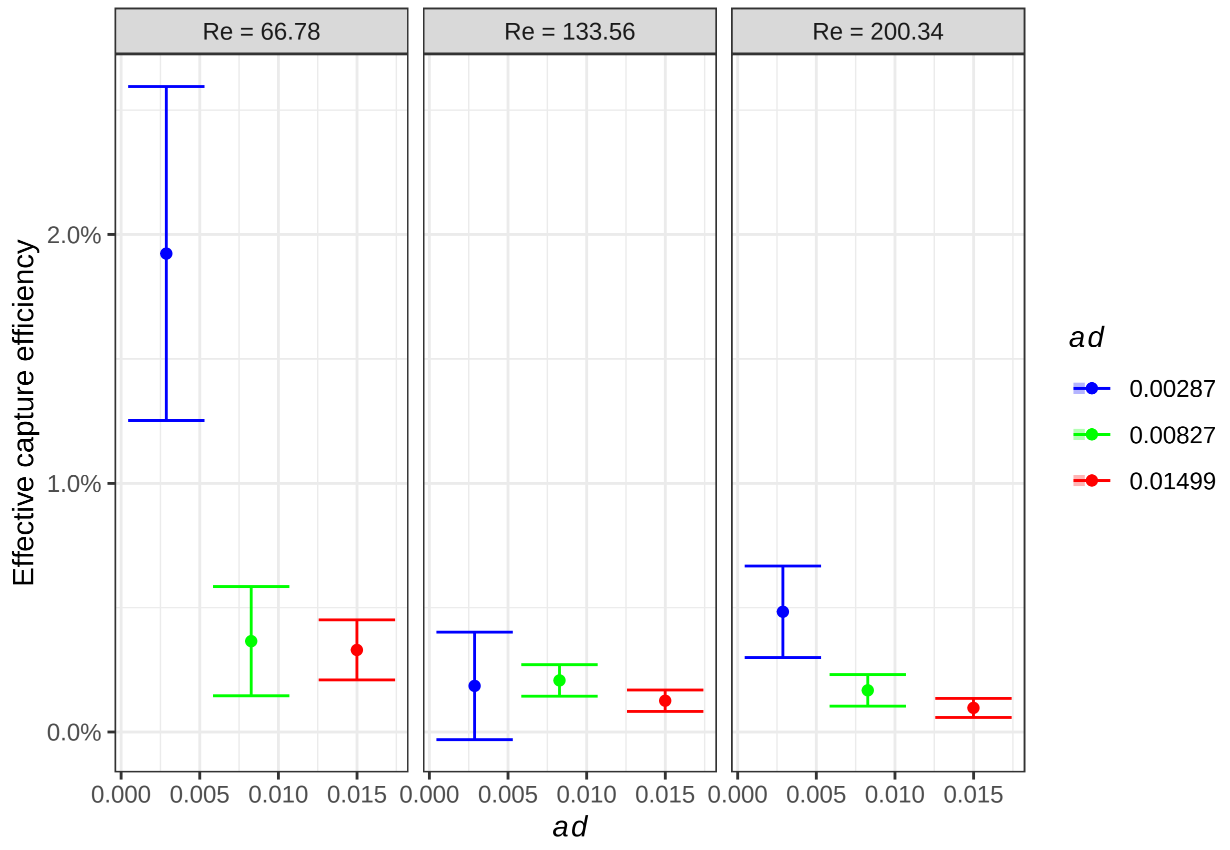
```
final <- final %>%
  mutate_with_error(k_c ~ k_t - k_s - k_bg)

final %>%
  filter(dowel_density != "0000") %>%
  ggplot(aes(x = Re, y = k_c, ymin = k_c - dk_c, ymax = k_c + dk_c, color = as.character(round(ad,5))))
  geom_line(lty = "dotted", size = 2, alpha = .3) +
  geom_point(position = position_dodge(width = 10)) +
  geom_errorbar(width = 10, position = "dodge") +
  scale_color_manual(values = c("blue","green","red")) +
  scale_y_continuous(labels = function(x) format(x, scientific = TRUE)) +
  labs(y = expression(paste(italic(k[c])," (",s^{-1},")")), x = expression(Re[c]), color = expression(i
  theme_minimal()
```



eta

```
#temp in flume measured at 22.2C with a calibrated thermometer
#plugged into https://www.engineeringtoolbox.com/water-dynamic-kinematic-viscosity-d_596.html
visc = 9.509e-7 #kinematic viscosity, m2/s

d = .003175 #dowel diameter

final <- final %>%
  mutate(frontal_area = as.numeric(dowel_density)*.003175,
           u = as.numeric(pump_freq)/500, #velocity
           Re = u*.003175/visc, #Reynolds #
           k_t = -t,
           k_c, k_s, k_bg) %>%
  mutate(eta = k_c/u/frontal_area,
         deta = dk_c/u/frontal_area) %>%
  mutate(eta = eta * 2.43/(1.95*.4*.6),
         deta = deta * 2.43/(1.95*.4*.6)) # don't forget to correct for time out of test section!

final %>%
  filter(dowel_density != "0000") %>%
  mutate(Refac = factor(Re, labels = c("Re = 66.78", "Re = 133.56", "Re = 200.34"))) %>%
  ggplot(aes(x = ad, y = eta, ymin = eta - deta, ymax = eta + deta, color = as.character(round(ad,5)))) +
  geom_line(lty = "dotted", size = 2, alpha = .3) +
  geom_point() +
  geom_errorbar() +
  scale_color_manual(values = c("blue","green","red")) +
  scale_y_continuous(labels = scales::percent) +
  facet_wrap(~Refac) +
  labs(y = "Effective capture efficiency", x = expression(italic(ad)), color = expression(italic(ad)))
  theme_bw()
```

```
## geom_path: Each group consists of only one observation. Do you need to adjust
## the group aesthetic?
## geom_path: Each group consists of only one observation. Do you need to adjust
## the group aesthetic?
## geom_path: Each group consists of only one observation. Do you need to adjust
## the group aesthetic?
```



Addressing Laurel's comments on the outline

Most of Laurel's comments addressed the structure and written content of the paper. However, several recommended new analyses to include.

1: Finding C for each level of ad, comparing to Fauria

COMMENT:

To better flesh out the role of stem density, you could consider figuring out what C would be for each stem density, using the same functional form of the equation as Fauria. C from our study and Fauria's study could be plotted against dimensionless stem density (i.e., ad or solid volume fraction) to determine the nature of the relationship.

ADDRESSED:

Fauria's functional form was $CRe_c^{-1.14}R^{0.65}$. Excluding runs with biofilm, they got C values of 1.53 and 3.18 for stem densities of 7209 (high) and 2724 (low) per meter, respectively.

Let's dig into stem density, frontal area, ad, etc.

Fauria didn't use cylinders but they give dimensions for their synthetic grass: average x-y plane cross-section was 3mm x 1mm, and it seems that they set water height to match stem height roughly. Given that their stems faced randomy in all directions, average *frontal* diameter per stem assuming a 3mm x 1mm rectangular prism (to avoid a multiple integral) would be approximated by, in mm,

$$\frac{\int_0^{\pi/2}[3\sin(\theta)+\cos(\theta)]\,\mathrm{d}\theta}{\pi/2}=\frac{(1)-(-3)}{\pi/2}=\frac{8}{\pi}\,.$$

This, converted to meters, times stems per unit bed area gives frontal area per unit volume: 6.93 m^{-1} (low) and 18.36 m^{-1} (high).

ad is frontal area per unit volume times diameter, making it dimensionless. Since it's unclear which d_c to use for non-cylindrical objects, I'll just use that same frontal d_c again: $8/\pi$. This yields *ad* values of 0.0177 and 0.0467.

Solid volume fraction (ϕ) is actually quite easy for rectangular prisms, and would theoretically be equivalent for the stems' actual, more trapezoidal shapes, if Fauria averaged correctly. It's simply $l \times w \times I_c$. This yields ϕ values of 0.00817 and 0.02163.

As for our values, cylinders make everything quite easy. We already have stem density and *ad* in the table, and can calculate frontal area and ϕ with ease:

```r
final <- final %>%
  mutate(frontal = ad/.003175, phi = ad*pi/4)

final %>%
  select(dowel_density, frontal, ad, phi) %>%
  distinct() %>%
  arrange(ad) %>%
  filter(ad>0)
```

```
## # A tibble: 3 x 4
##   dowel_density frontal      ad     phi
##   <chr>           <dbl>   <dbl>   <dbl>
## 1 0278            0.905 0.00287 0.00226
## 2 0800            2.61  0.00827 0.00650
## 3 1450            4.72  0.0150  0.0118
```

Next, particle ratios: Fauria ran models across a range, but without LISST data, we only know η for our particles altogether. Given our constant d_c of 0.3175 cm and our median particle size of 25.2 microns, we'd get a particle-to-stem ratio of 0.007937. This raised to the .65 power gives a corrective term of 0.04312975.

OK, so now for the C value models:

```r
cmods <- final %>%
  mutate(coeff = 0.04312975*Re^-1.14) %>%
  lmList(eta ~ 0 + coeff|phi, data = .)
```

```
## Warning: Unknown or uninitialised column: `(weights)`.

## Warning: Unknown or uninitialised column: `(offset)`.

## Warning: Unknown or uninitialised column: `(weights)`.
## Warning: Unknown or uninitialised column: `(offset)`.
```

```
## Warning: Unknown or uninitialised column: `(weights)`.
```

```
## Warning: Unknown or uninitialised column: `(offset)`.
```

```
## Warning: Unknown or uninitialised column: `(weights)`.
```

```
## Warning: Unknown or uninitialised column: `(offset)`.
```

```
## Warning in lmList(eta ~ 0 + coeff | phi, data = .): 1 error caught in lm.fit(x,
## y, offset = offset, singular.ok = singular.ok, ...): 0 (non-NA) cases
```

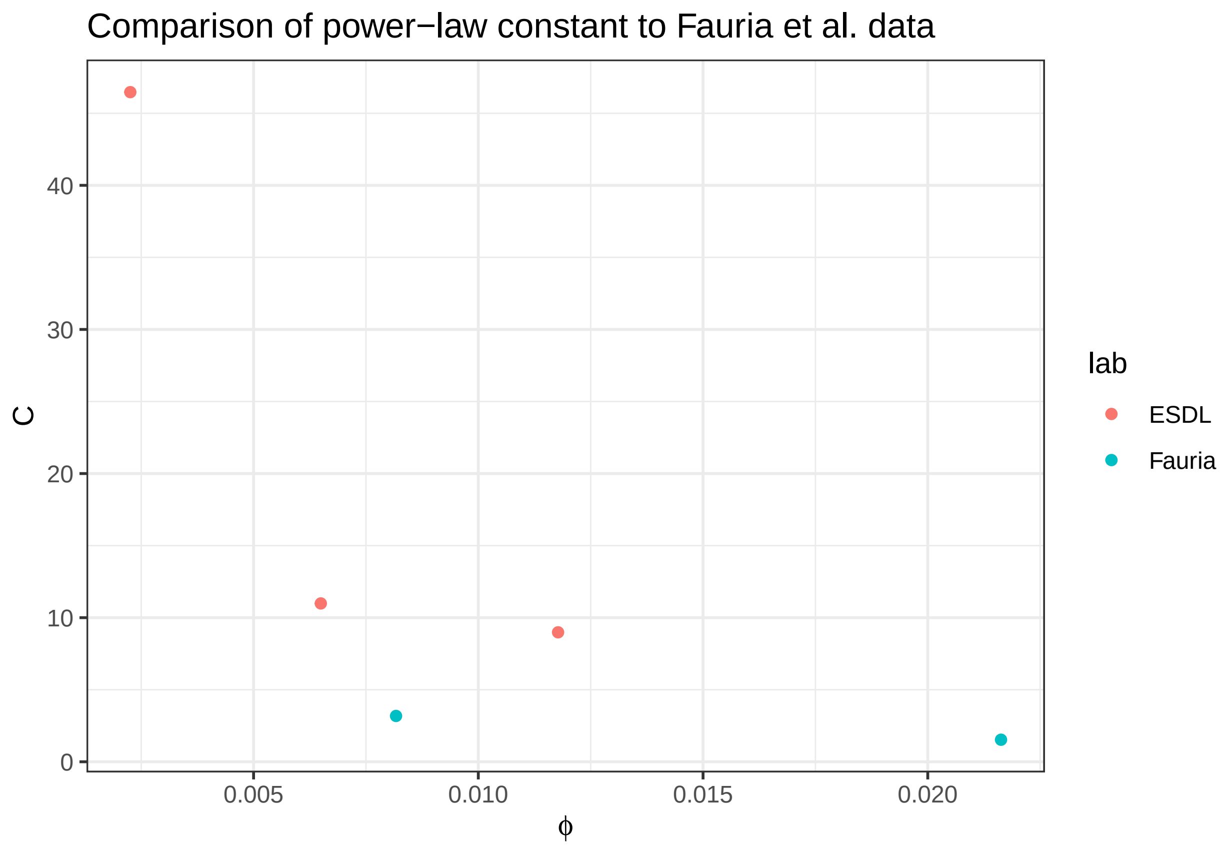
So then, to put our results and Fauria's side by side:

```
c_over_phi <- cbind(rownames(coef(cmods))[2:4],coef(cmods)[2:4,]) %>%
  as.tibble() %>%
  mutate_all(as.numeric) %>%
  mutate(lab = "ESDL") %>%
  rename(phi = V1, C = V2)
```

```
## Warning: `as.tibble()` is deprecated as of tibble 2.0.0.
## Please use `as_tibble()` instead.
## The signature and semantics have changed, see `?as_tibble`.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```
## Warning: The `x` argument of `as_tibble.matrix()` must have unique column names if `.name_repair` is
## Using compatibility `.name_repair`.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```
fauria <- tibble(phi = c(.00817,.02163), C = c(3.18,1.53), lab = "Fauria")

c_over_phi <- rbind(c_over_phi, fauria)

c_over_phi %>%
  ggplot(aes(x = phi, y = C, color = lab)) +
  geom_point() +
  theme_bw() +
  labs(title = "Comparison of power-law constant to Fauria et al. data",
       x = expression(phi))
```



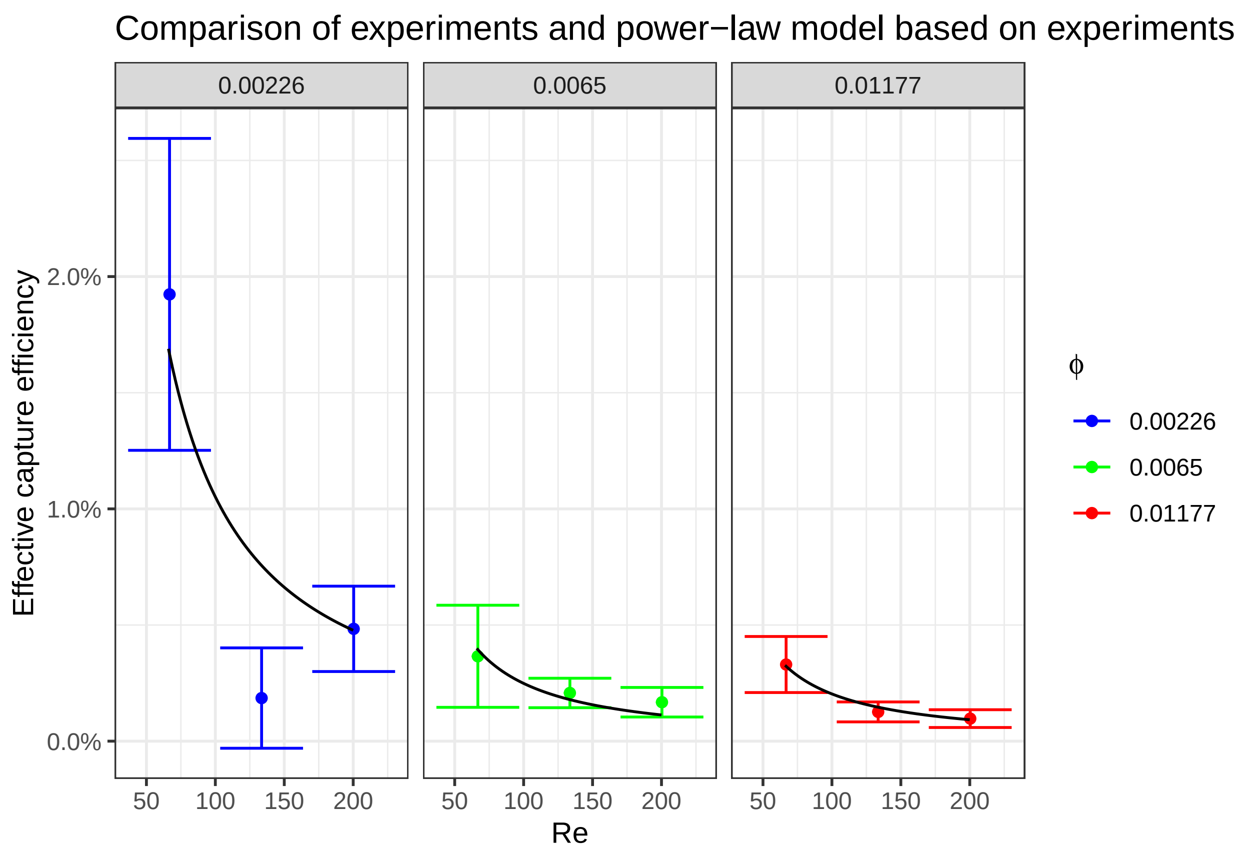
2: Comparing Fauria model predictions to our observations

COMMENT:

If you use the same exponents as Fauria but treat C as a free parameter for each stem density configuration, how do predictions compare to the observations? This would be a plot either of Re vs eta for experimental data (points with error bars) and model predictions (lines) or of predicted vs. observed eta (with error bars for observed).

ADDRESSED:

```r
# curve data
fauria_pred <- tibble(Re = rep(seq(66,200,length = 1000),3),
       C = c(rep(46.46556,1000),rep(10.987587,1000),rep(8.983344,1000)),
       phi = c(rep(0.002257447,1000),rep(0.006496250,1000),rep(0.011774453,1000))) %>%
  mutate(eta_est = Re^-1.14*C*0.04312975, ad = phi)

final %>%
  filter(dowel_density != "0000") %>%
  mutate(Refac = factor(Re, labels = c("Re = 66.78", "Re = 133.56", "Re = 200.34"))) %>%
  ggplot(aes(x = Re, y = eta, ymin = eta - deta, ymax = eta + deta, color = as.character(round(phi,5)))
  geom_point() +
  geom_errorbar() +
  geom_line(data = fauria_pred, aes(y = eta_est, x = Re), inherit.aes = FALSE) +
  scale_color_manual(values = c("blue","green","red")) +
  scale_y_continuous(labels = scales::percent) +
  facet_wrap(~ as.character(round(phi,5))) +
  labs(y = "Effective capture efficiency", x = expression(Re), color = expression(phi)) +
  theme_bw() +
  ggtitle("Comparison of experiments and power-law model based on experiments")
```



3: Monte Carlo approach

COMMENT:

Each data point has a normal distribution associated with the estimate of eta (defined by the estimate value and the associated standard deviation, i.e., the error bars). For each stem density, you can do Monte Carlo sampling in which you draw a different value of eta from each of the three Re by sampling from the underlying distributions. For each realization of the three eta estimates, you would then do a log transformation, fit a straight line, and save the slope of that line. The result is a distribution of slopes over 1000 or more realizations. If the 95% confidence interval for those slopes includes zero, we cannot say with statistical certainty that there is a negative trend with Re. You would do this for each stem density. Then, you would evaluate whether there is a significant trend in eta with stem density by holding Re constant and doing the same thing for each Re. You could show the results as box plots, where you use some symbol (such as an asterisk) to denote those boxes whose 95% confidence interval does not include zero.

ADDRESSED:

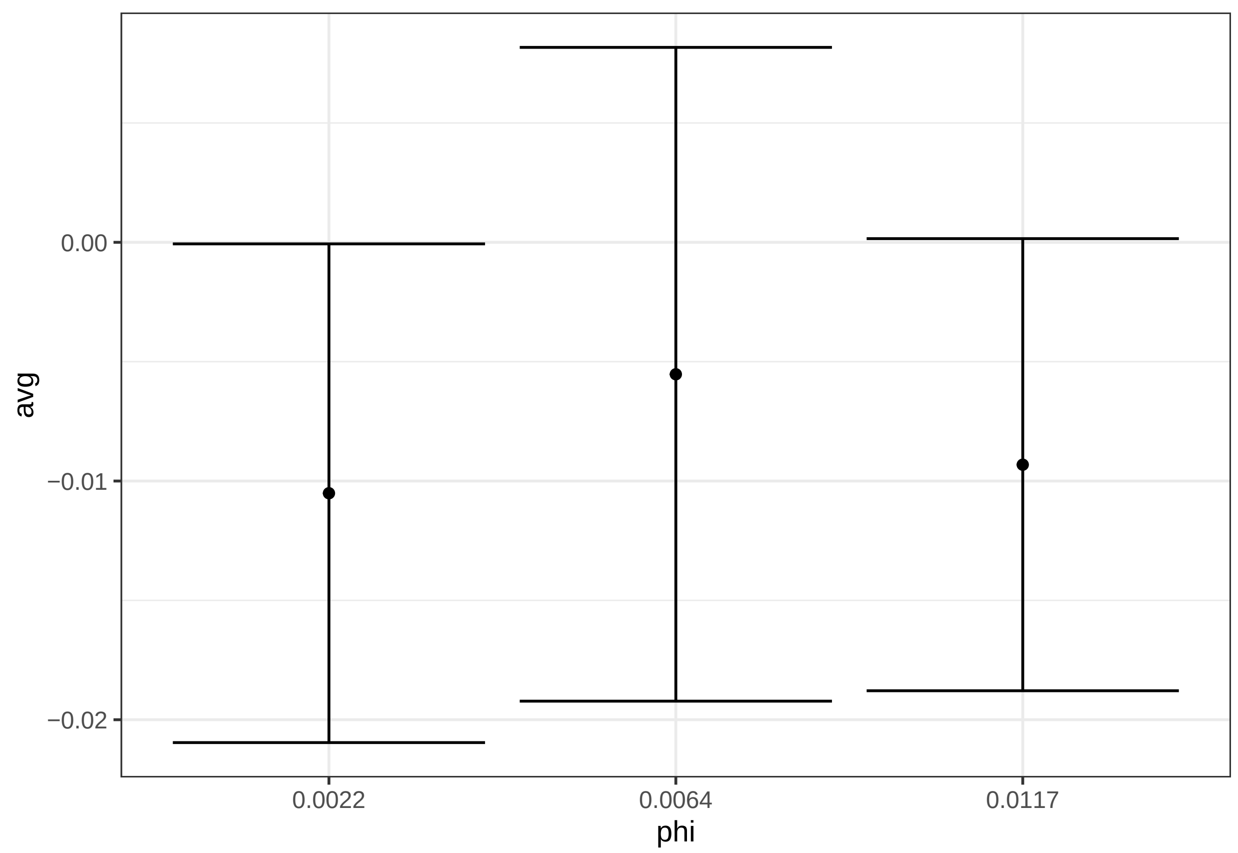
Grouped by stem density:

```
skeleton_eta <- final %>%
  filter(frontal_area > 0) %>%
  select(phi, Re, eta, deta) %>%
  arrange(phi, Re)
```

```r
monte <- list()

for (i in 1:nrow(skeleton_eta)) {
  monte[[i]] <- tibble(phi = skeleton_eta$phi[i],
                       Re = skeleton_eta$Re[i],
                       eta = rnorm(1000, mean = skeleton_eta$eta[i], sd = skeleton_eta$deta[i]))
}

lmsphi <- bind_rows(monte) %>%
  mutate(ID = paste0(phi, 1:1000)) %>%
  lmList(log(eta) ~ Re|ID, data = .) %>%
  coefficients()

lmsphi <- tibble(phi = str_sub(rownames(lmsphi),0,6),lmsphi)


lmsphi <- lmsphi %>%
  group_by(phi) %>%
  summarise(avg = mean(Re, na.rm = TRUE), sd = sd(Re, na.rm = TRUE)) %>%
  mutate(ymax = avg + 1.96*sd)

lmsphi
```

```
## # A tibble: 3 x 4
##   phi          avg      sd       ymax
##   <chr>      <dbl>   <dbl>      <dbl>
## 1 0.0022 -0.0105  0.00533 -0.0000646
## 2 0.0064 -0.00553 0.00699  0.00817
## 3 0.0117 -0.00932 0.00483  0.000155
```

```r
lmsphi %>%
  ggplot(aes(x = phi, y = avg, ymin = avg-1.96*sd, ymax = avg+1.96*sd)) +
  geom_point() +
  geom_errorbar() +
  theme_bw()
```

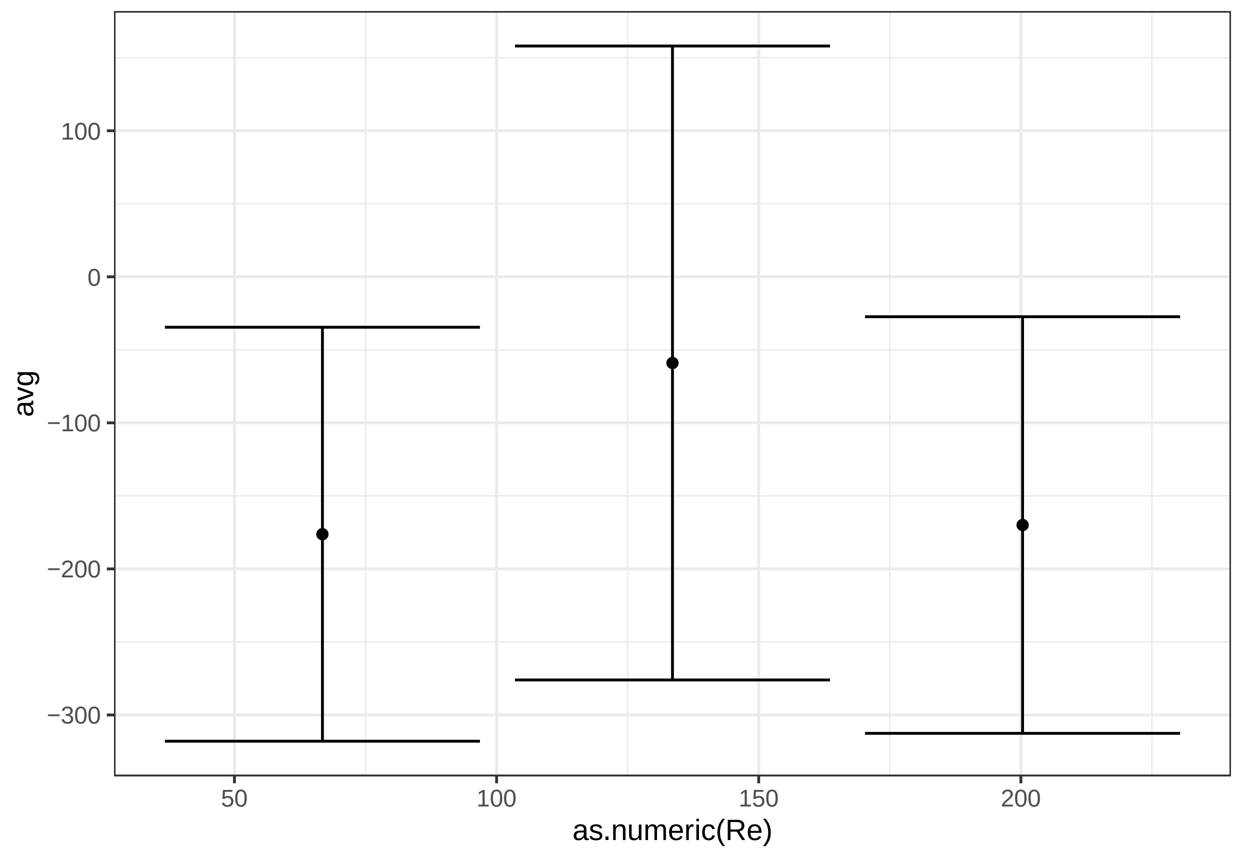


Grouped by Re:

```
monte2 <- list()

for (i in 1:nrow(skeleton_eta)) {
  monte2[[i]] <- tibble(phi = skeleton_eta$phi[i],
                        Re = skeleton_eta$Re[i],
                        eta = rnorm(1000, mean = skeleton_eta$eta[i], sd = skeleton_eta$deta[i]))
}

lmsre <- bind_rows(monte2) %>%
  mutate(ID = paste0(Re, 1:1000)) %>%
  lmList(log(eta) ~ phi|ID, data = .) %>%
  coefficients()

lmsre <- tibble(Re = str_sub(rownames(lmsre),0,6),lmsre)


lmsre <- lmsre %>%
  group_by(Re) %>%
  summarise(avg = mean(phi, na.rm = TRUE), sd = sd(phi, na.rm = TRUE)) %>%
  mutate(ymax = avg + 1.96*sd)

lmsre
```

```
## # A tibble: 3 x 4
##   Re        avg    sd  ymax
##   <chr>   <dbl> <dbl> <dbl>
## 1 133.55  -59.0 111.  158.
## 2 200.33 -170.   72.8 -27.3
## 3 66.778 -176.   72.3 -34.5
```

```
lmsre %>%
  ggplot(aes(x = as.numeric(Re), y = avg, ymin = avg-1.96*sd, ymax = avg+1.96*sd)) +
  geom_point() +
  geom_errorbar() +
  theme_bw()
```



3 addendum: Safety check on null result (assuming no effect of treatments on k_c)

ALL DATA IN THIS SECTION IS IMAGINARY!!!!

Let's see what results we'd get with these same analyses if we detected no effects of Re or ϕ on k_c.

```
imaginary_eta <- final %>%
  filter(frontal_area > 0) %>%
  mutate(imagk_c = mean(k_c),
         imagdk_c = mean(dk_c),
         imageta = imagk_c/u/frontal_area,
         imagdeta = imagdk_c/u/frontal_area) %>%
  mutate(imageta = imageta * 2.43/(1.95*.4*.6),
         imagdeta = imagdeta * 2.43/(1.95*.4*.6)) %>%
  select(phi, Re, eta = imageta, imagdeta)
```

Grouped by stem density:

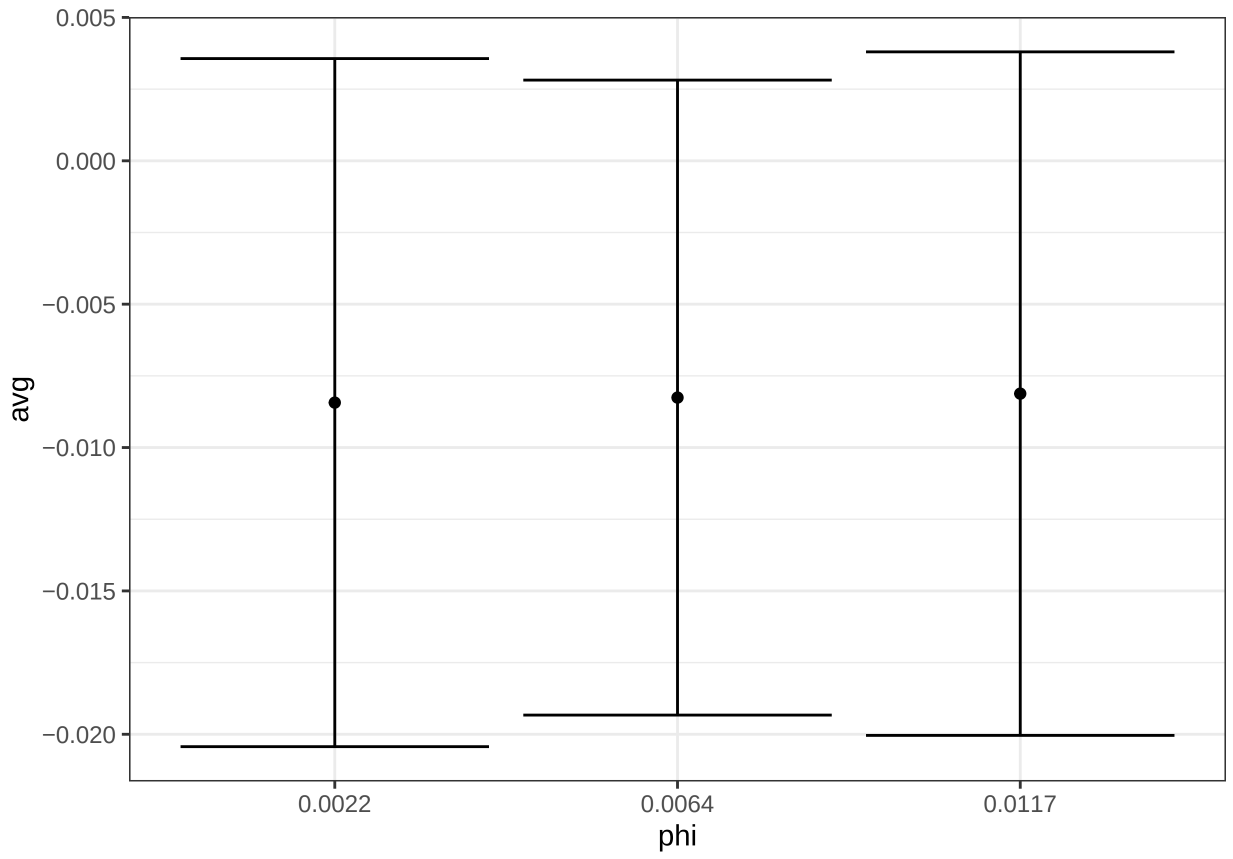
```
monte <- list()

for (i in 1:nrow(imaginary_eta)) {
  monte[[i]] <- tibble(phi = imaginary_eta$phi[i],
                       Re = imaginary_eta$Re[i],
                       eta = rnorm(1000, mean = imaginary_eta$eta[i], sd = imaginary_eta$imagdeta[i]))
}

lmsphi <- bind_rows(monte) %>%
  mutate(ID = paste0(phi, 1:1000)) %>%
  lmList(log(eta) ~ Re|ID, data = .) %>%
  coefficients()

lmsphi <- tibble(phi = str_sub(rownames(lmsphi),0,6),lmsphi)


lmsphi <- lmsphi %>%
  group_by(phi) %>%
  summarise(avg = mean(Re, na.rm = TRUE), sd = sd(Re, na.rm = TRUE)) %>%
  mutate(ymax = avg + 1.96*sd)
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
lmsphi
```

```
## # A tibble: 3 x 4
##   phi         avg      sd    ymax
##   <chr>     <dbl>   <dbl>   <dbl>
## 1 0.0022 -0.00843 0.00612 0.00357
## 2 0.0064 -0.00826 0.00565 0.00281
## 3 0.0117 -0.00812 0.00608 0.00380
```

```
lmsphi %>%
  ggplot(aes(x = phi, y = avg, ymin = avg-1.96*sd, ymax = avg+1.96*sd)) +
  geom_point() +
  geom_errorbar() +
  theme_bw()
```


0.005
0.000
−0.005
−0.010
−0.015
−0.020
avg
0.0022
0.0064
0.0117
phi