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, 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.

NEW ADDITION: Uncertainty propogation

Here is a pretty swell function found on StackExchange.com, which performs error propogation in R using derivatives:

```
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)
```

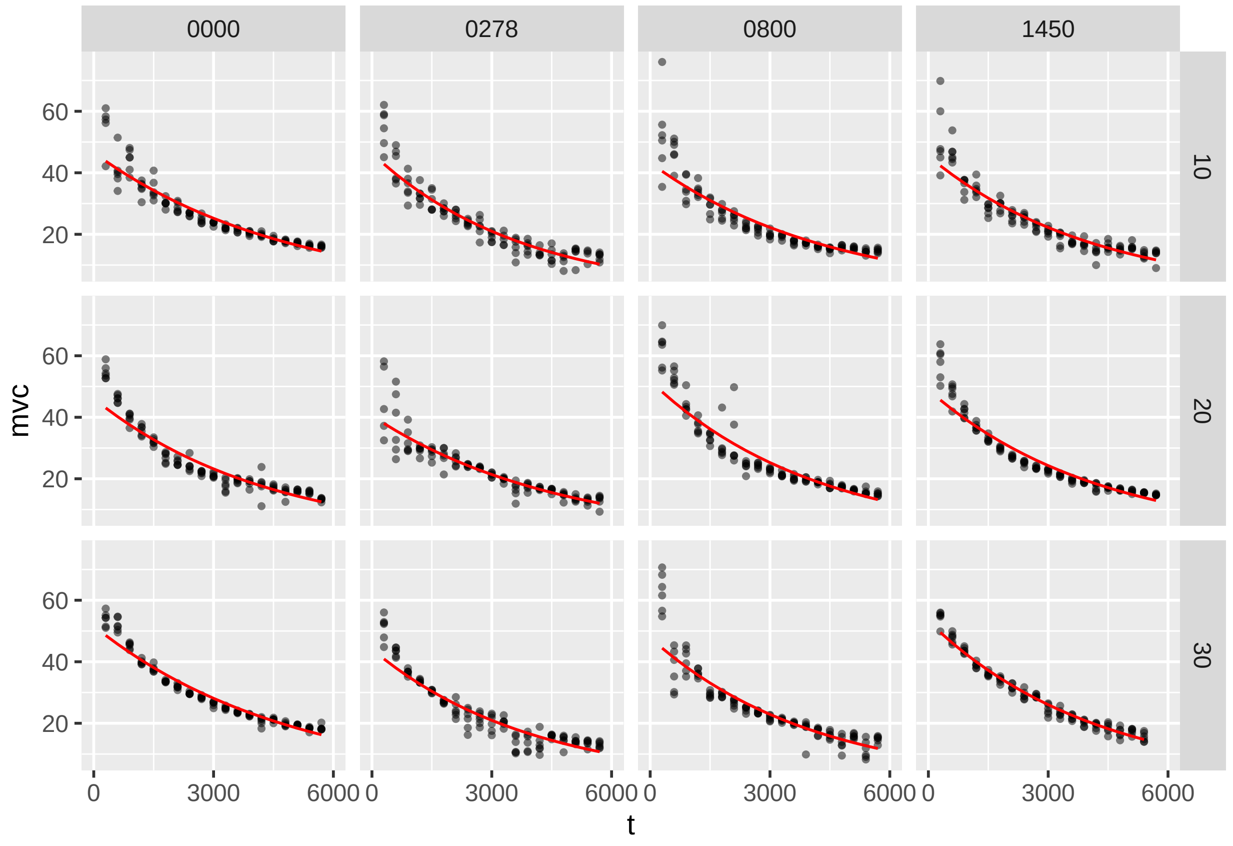
```
## Call:
##   Model: log(mvc) ~ t | NULL
##    Data: pumpfinal
```

```
##
## Coefficients:
##    (Intercept)
##        Estimate Std. Error  t value Pr(>|t|)
## 181115 3.973846 0.02612434 152.1128        0
## 190321 3.943840 0.02537023 155.4515        0
## 190506 3.785685 0.02537023 149.2176        0
## 190612 3.867590 0.02537023 152.4460        0
## 190729 3.840790 0.02579446 148.8998        0
## 190802 3.830722 0.02537023 150.9928        0
## 190905 3.836769 0.02556521 150.0777        0
## 190926 3.701444 0.02579137 143.5148        0
## 191029 3.767874 0.02538174 148.4482        0
## 200211 3.948348 0.02537023 155.6292        0
## 200303 3.815678 0.02537023 150.3998        0
## 200313 3.889987 0.02537023 153.3288        0
##    t
##             Estimate   Std. Error   t value      Pr(>|t|)
## 181115 -0.0002381660 8.044925e-06 -29.60450 2.484909e-148
## 190321 -0.0002025636 7.417054e-06 -27.31052 1.054619e-130
## 190506 -0.0002477993 7.417054e-06 -33.40939 3.910959e-178
## 190612 -0.0002455980 7.417054e-06 -33.11261 8.572682e-176
## 190729 -0.0002045307 7.511472e-06 -27.22911 4.384970e-130
## 190802 -0.0002286986 7.426920e-06 -30.79320 1.390124e-157
## 190905 -0.0002643807 7.629356e-06 -34.65308 5.831753e-188
## 190926 -0.0002130646 7.507285e-06 -28.38105 6.908735e-139
## 191029 -0.0002221626 7.418149e-06 -29.94852 5.303858e-151
## 200211 -0.0002397320 7.417054e-06 -32.32173 1.448716e-169
## 200303 -0.0002382292 7.417054e-06 -32.11912 5.671691e-168
## 200313 -0.0002333927 7.417054e-06 -31.46703 7.432157e-163
##
## Residual standard error: 0.1301265 on 1328 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:

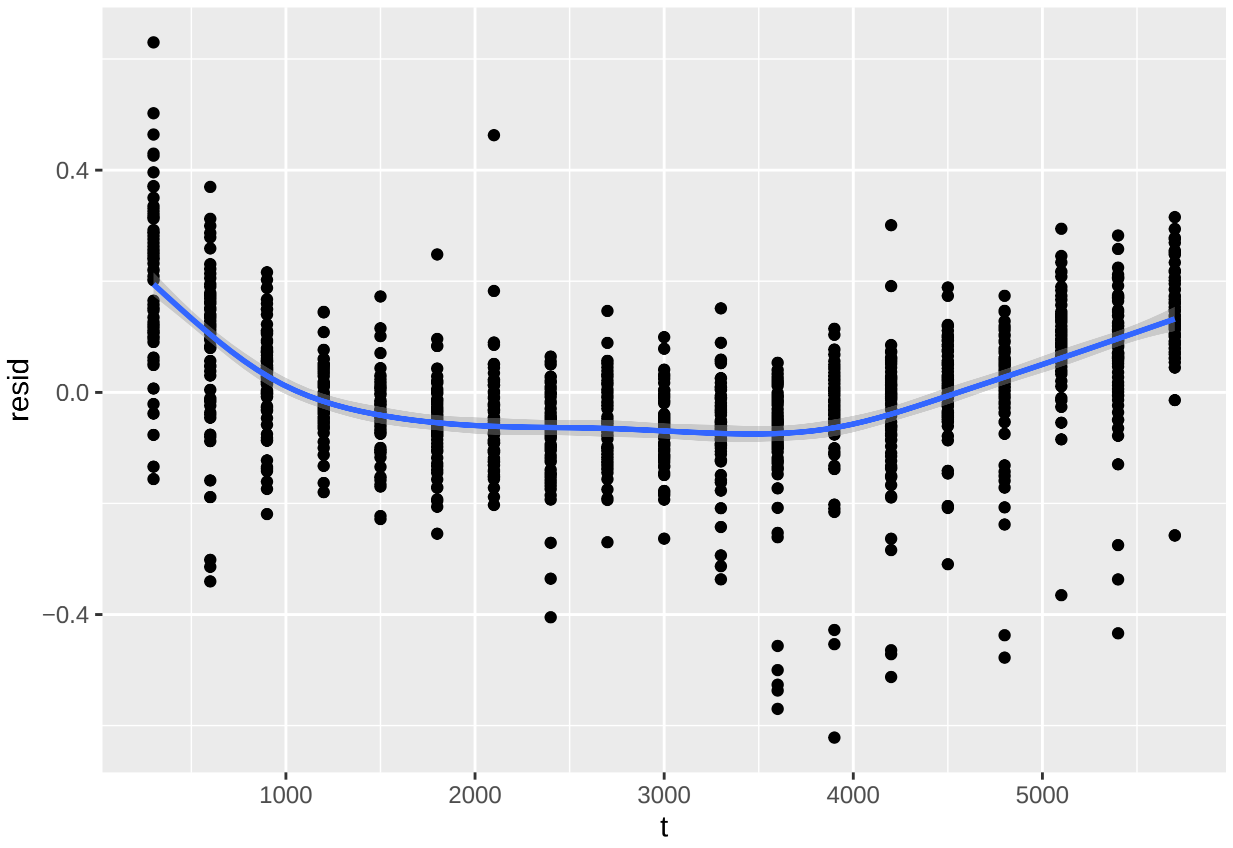
```r
cbind(pumpfinal, pred = predict(fits)) %>%
  ggplot() +
  geom_point(aes(x = t, y = mvc), size = .75, alpha = .5) +
  geom_line(aes(x = t, y = exp(pred)), color = "red") +
  facet_grid(pump_freq~dowel_density) +
  scale_x_continuous(breaks = c(0,3000,6000), limits = c(0,6000))
```



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)) +
  geom_point() +
  geom_smooth()
```

```
## `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.

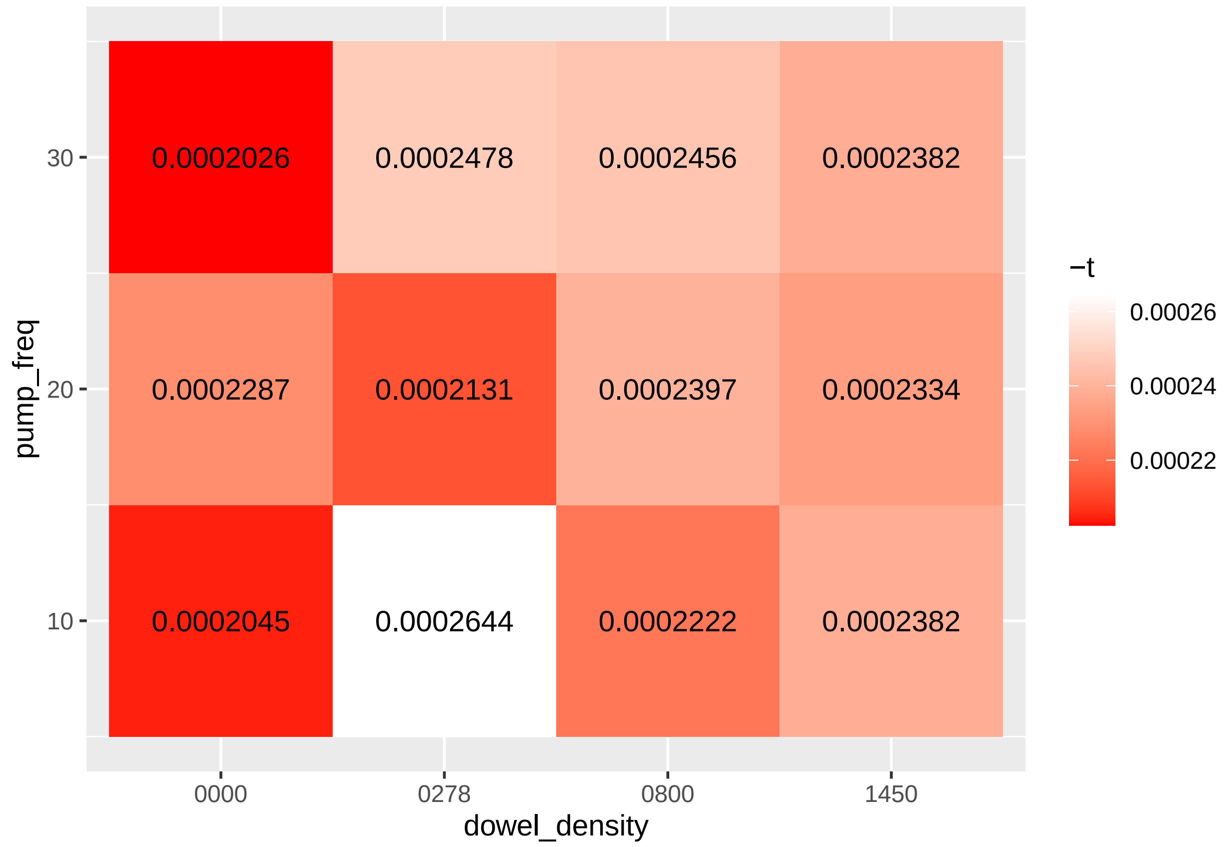
Whatever the explanation is, we (and other researchers using the exponential model) assume it is uniform across our independent variables. I don't know how safe that assumption is, but that'll have to wait for a future paper. It might be testable from our data but `n = 12` seems a bit of a small sample. 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. Though there are high **absolute** measurements at the beginning, on the **log-scale** they are far less influential than the lower values (`mvc < 10`) at the end. Removing the lower outliers towards the end (`t > 3000`) might be worth trying later if our results don't satisfy.

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

```r
fitdata <- coefficients(fits) %>%
cbind(date = as.numeric(row.names(coefficients(fits)))) %>%
  left_join(metadata, by = "date")

fitdata %>%
  ggplot(aes(x = dowel_density, y = pump_freq, fill = -t, label = round(-t,7))) +
    geom_tile() +
    geom_text() +
  scale_fill_gradient(low = "red", high = "white")
```



Mixed feelings. On the one hand it's reassuring that aside from the switcharoo at 20 Hz, all runs with dowels had greater k_t than runs without. However, it seems like there's some serious variance. **What confounders could we be missing???**

THE PROBLEM IN A NUTSHELL

Having a greater decay rate for the control run than the dowel run, by an amount greater than the difference in k_s between the two runs, implies a negative k_c. This seems unrealistic, or at least very out of step with our operating theory. For the rest of this analysis, I'll do what I slipped into my past code and then forgot about: leave out that control run and estimate its values as the mean of the other controls (it does lie at the mean of their independent vars. so weighting the average seems unnecessary). Not ideal! Maybe we should give this set of parameters another shot? But I don't feel great about cherrypicking experimental results that way either.

```r
# THIS CODE WAS USED IN THE FIRST VERSION BUT SHOULD BE LEFT OUT OF ERROR PROPOGATION VERSION

#
# fitdata[fitdata$date==190802,"t"] <- (fitdata[fitdata$date==190729,"t"] +
#                                        fitdata[fitdata$date==190321,"t"])/2
#
# fitdata
```

k__s (time-decay due to settling)

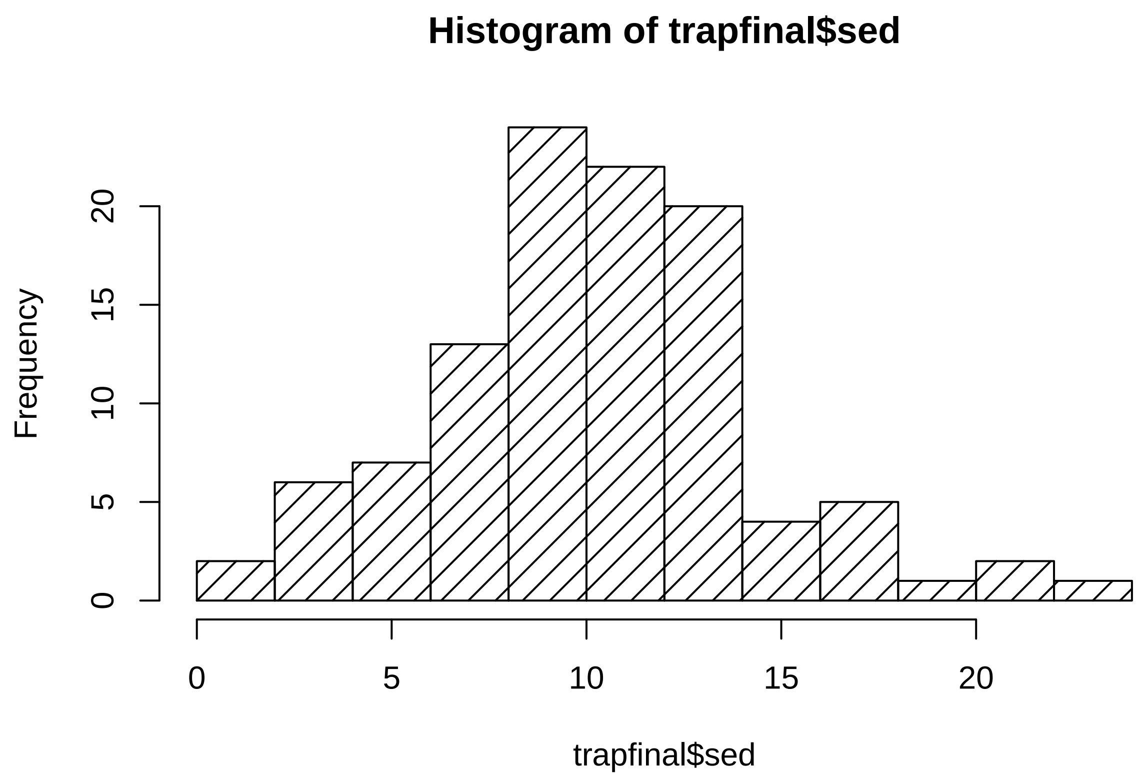
$$m_s = \frac{k_s}{k_{tot}}(1 - e^{-kT})m_0$$

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).

```
hist(trapfinal$sed, breaks = 10, density = 10)
```

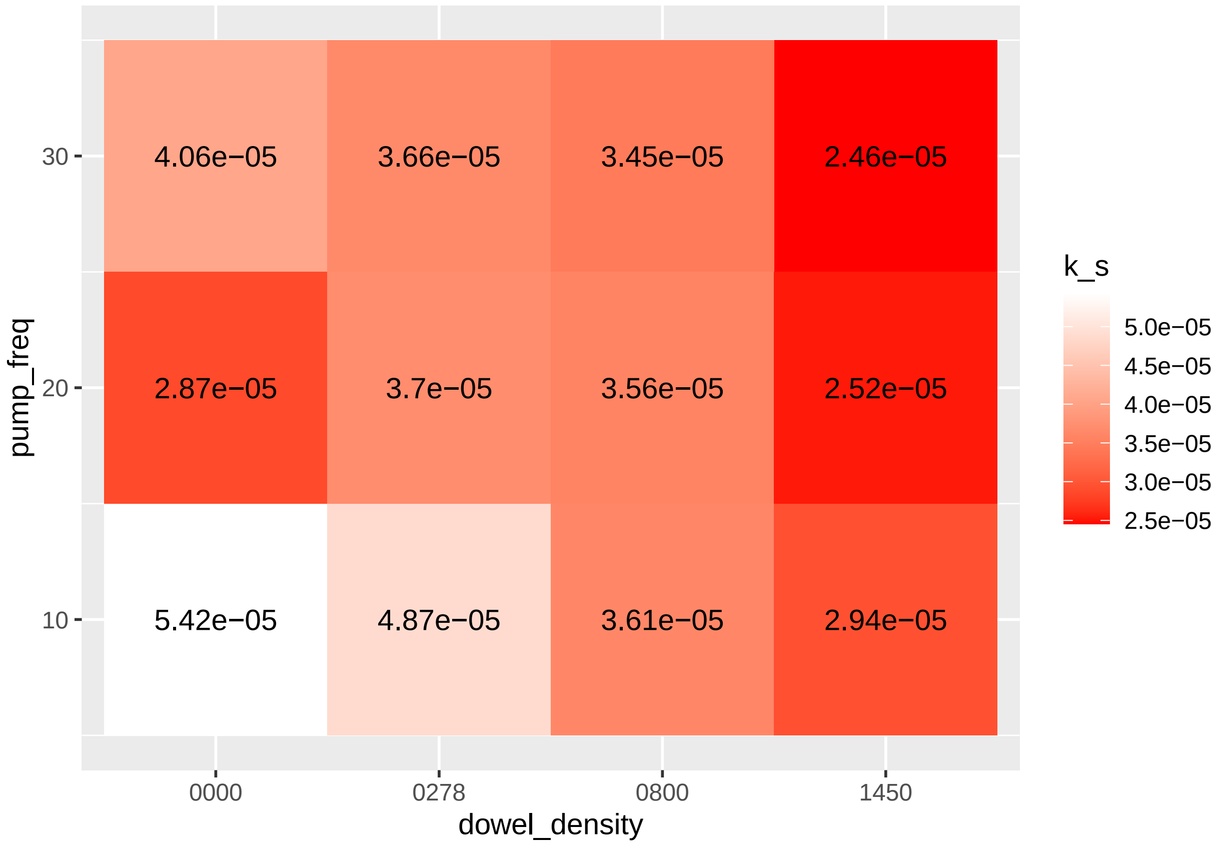


There aren't really any clear outliers, as we established in the boxplots above. While splines might be our final tool, for now an average will do.

```
final <- trapfinal %>%
  group_by(date) %>%
  summarise(m_trap = mean(sed)/1000) %>% #average sediment in trap converted from mg to g
  mutate(m_s = m_trap*1.95*.6/(pi*.0127^2)) %>% #times test section area divided by area of 1 trap
  left_join(fitdata, by = "date") %>%
  mutate(k_s = m_s/200/(1-exp(t*6000))*(-t))
```

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

```
final %>%
  ggplot(aes(x = dowel_density, y = pump_freq, fill = k_s, label = round(k_s,7))) +
    geom_tile() +
    geom_text() +
  scale_fill_gradient(low = "red", high = "white")
```



If we're not using the k_{tot} from the 20 Hz control, I think we should probably do the same with k_s, not least because it gives us linear k_{bg} values, though this all still feels a little iffy.

```
# THIS CODE WAS USED IN THE FIRST VERSION BUT SHOULD BE LEFT OUT OF ERROR PROPOGATION VERSION

# final[final$date==190802,"k_s"] <- (final[final$date==190729,"k_s"] +
#                                      final[final$date==190321,"k_s"])/2
```

Gladly, settling across the dowel treatments does seem to decrease, with a less clear pattern across 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") %>%
  transmute(pump_freq, k_bg = -t - k_s) %>%
  arrange(pump_freq)

bgvals
```

```
## # A tibble: 3 x 2
##   pump_freq     k_bg
##        <dbl>    <dbl>
## 1         10 0.000150
## 2         20 0.000200
## 3         30 0.000162
```

To add it to the table:

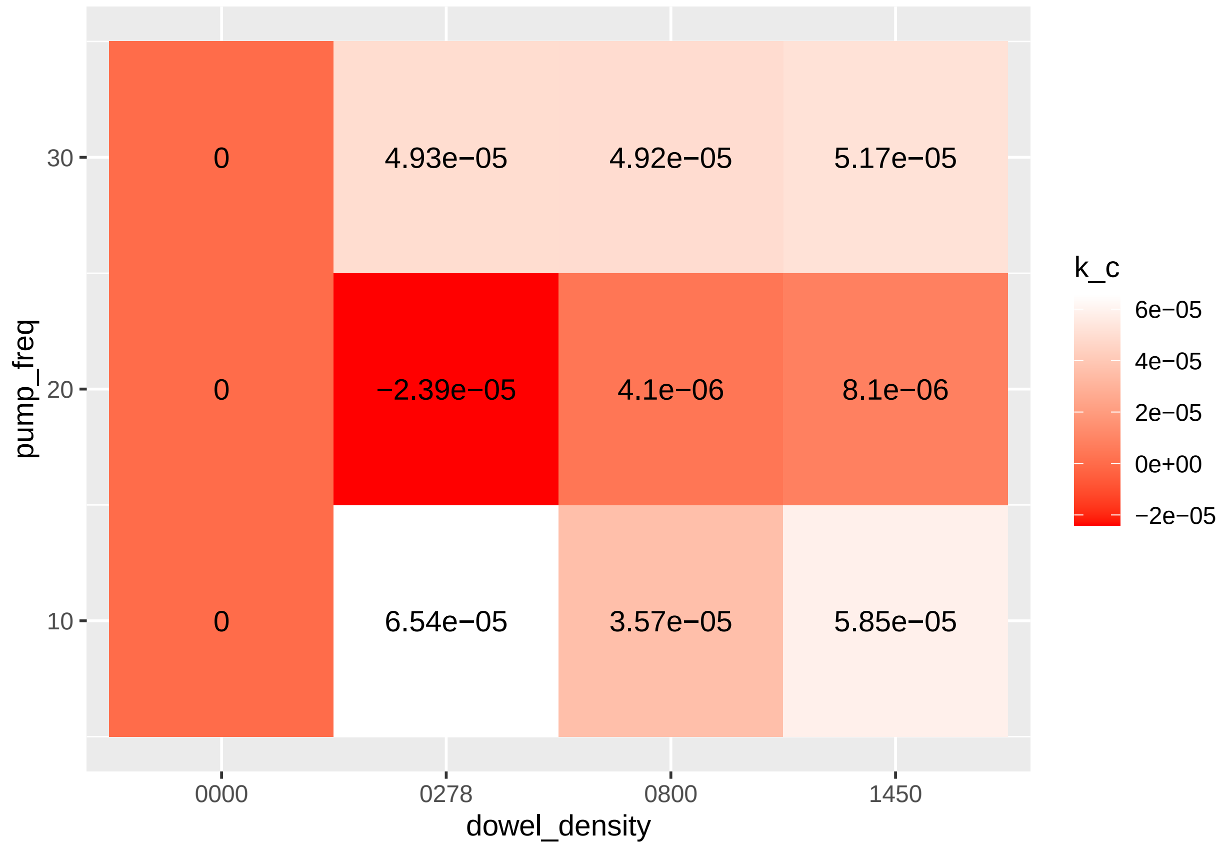
```
final <- left_join(final, bgvals, by = "pump_freq")
```

k_c (time-decay due to collection)

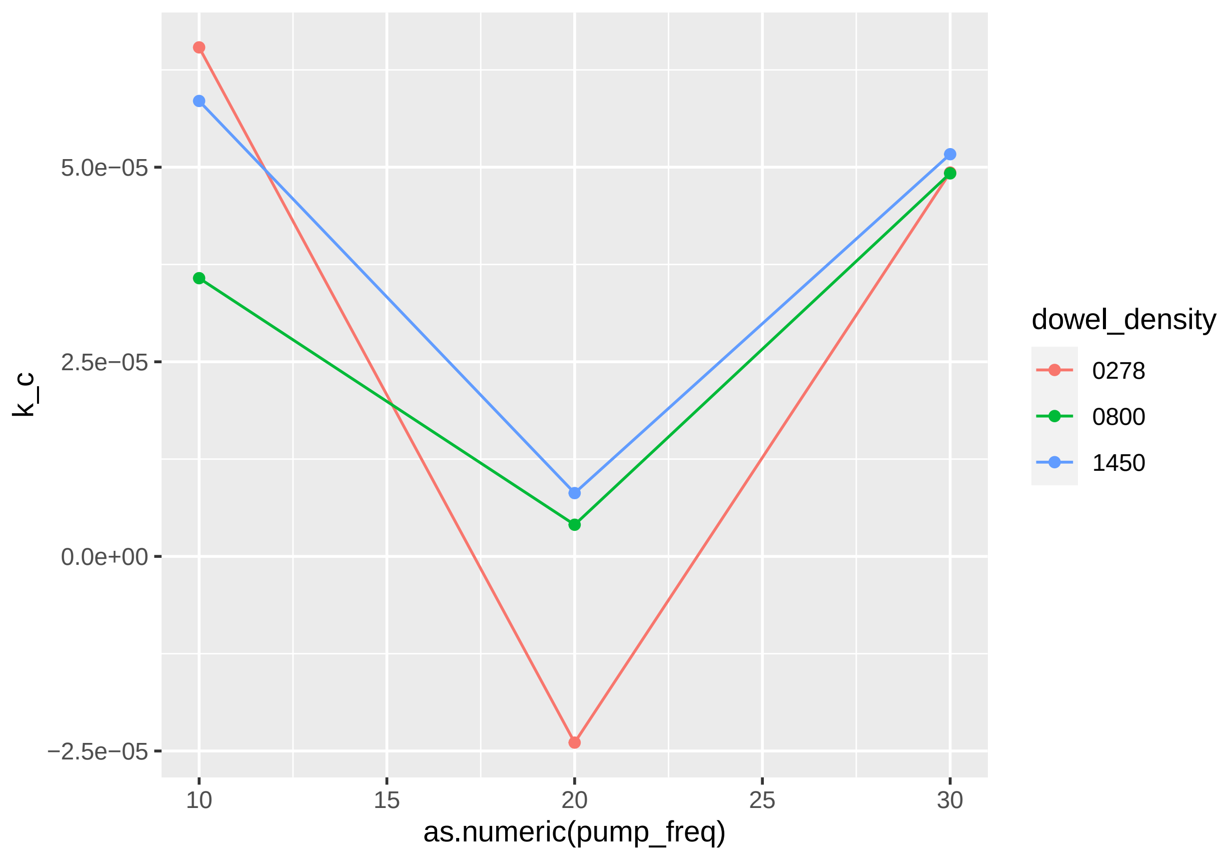
So, now that we have our values in the table, k_c is just a subtraction away:

```
final <- final %>%
  mutate(k_c = -t - k_s - k_bg)

final %>%
  ggplot(aes(x = dowel_density, y = pump_freq, fill = k_c, label = round(k_c,7))) +
    geom_tile() +
    geom_text() +
  scale_fill_gradient(low = "red", high = "white")
```



```
final %>%
  filter(dowel_density != "0000") %>%
  ggplot(aes(x = as.numeric(pump_freq), y = k_c, color = dowel_density)) +
  geom_line() +
  geom_point()
```



Unlike the calculation with the measured values at the 20 Hz control, all k_c are in the green, but the pattern is still quite odd.

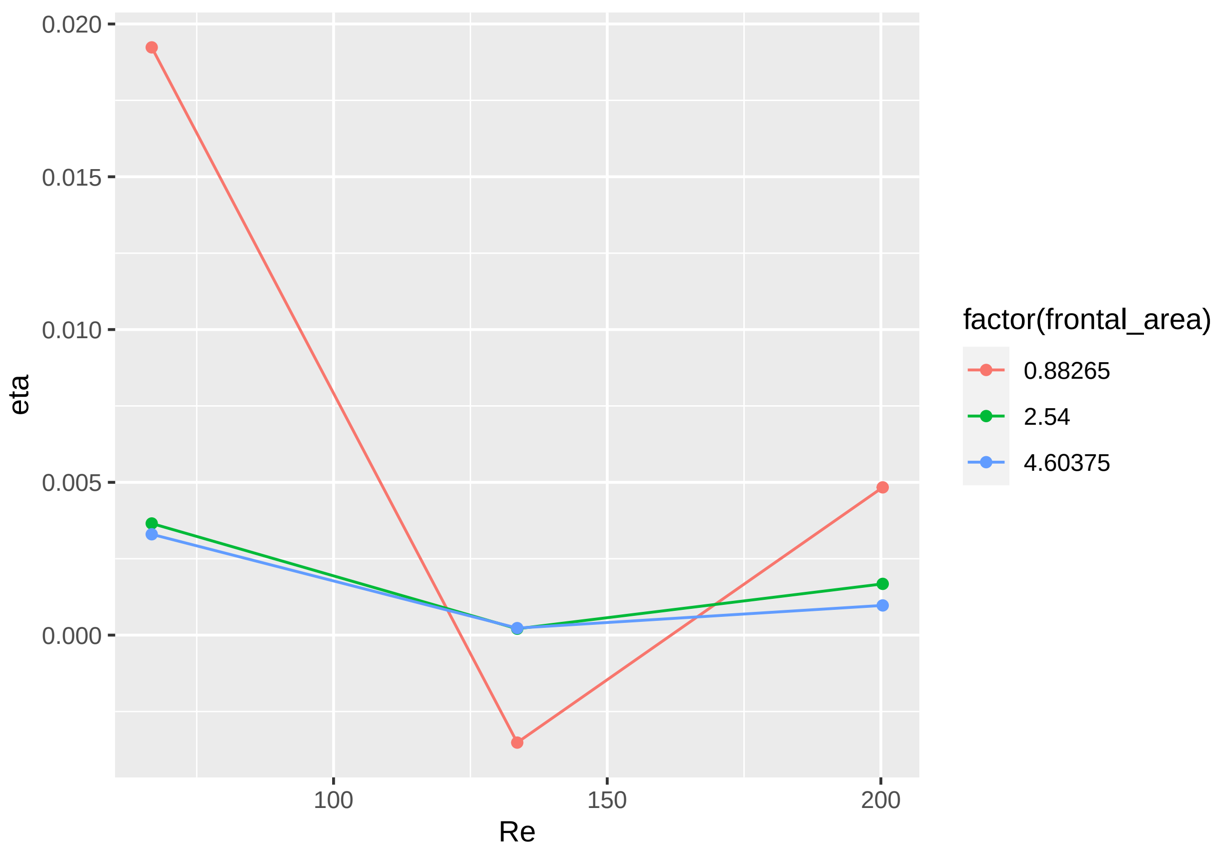
eta

```
#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 %>%
  transmute(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) %>%
  mutate(eta = eta * 2.43/(1.95*.4*.6)) # don't forget to correct for time out of test section!

final %>%
  filter(frontal_area != 0) %>%
  ggplot(aes(x = Re, y = eta, color = factor(frontal_area))) +
  geom_line() +
  geom_point()
```



Possible additions

- Turbulence data
  - The calculations are done
  - But it might be good to include here for the sake of comprehensiveness