response to andy 01

Justin Pomeranz

2024-05-10

Load libraries and their package versions

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(sizeSpectra)

print("tidyverse version")
```

```
## [1] "tidyverse version"
```

```
packageVersion("tidyverse")
```

```
## [1] '2.0.0'
```

```
print("sizeSpectra version")
```

```
## [1] "sizeSpectra version"
```

```
packageVersion("sizeSpectra")
```

```
## [1] '1.1.0'
```

```
# R version
R.version
```

```
##                _
## platform       x86_64-w64-mingw32
## arch           x86_64
## os             mingw32
## crt            ucrt
## system         x86_64, mingw32
## status
## major          4
## minor          4.0
## year           2024
## month          04
## day            24
## svn rev        86474
## language       R
## version.string R version 4.4.0 (2024-04-24 ucrt)
## nickname       Puppy Cup
```

Load example data for Andy

```
raw_orig <- read_csv("derived_data/example_for_andrew.csv")
```

```
## Rows: 4844 Columns: 22
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (10): site, site_date, sampling_method, organism_group, taxon, body_weig...
## dbl (12): year, month, sample, sampling_area, body_mass, body_length, count,...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Answers to Andy's Questions

1. Interpret `counts`. The way I interpret the count of 1.3 or 2.6 in your example would be that those numbers are the average abundance of that body size across 1 km^2 for fish, or the avergae number per m^2 for macroinvertebrates / + Fish.

As a larger point for the data resolution, this is going to be different for each data set. For this example (`df_Arranz`) I believe that fish were sampled with electrofishing and each individual was measured (either lengths or weights?). As far as I know, they were measured to the nearest x units do to rounding. So I guess technically they would be in "bins", but I think the original data was collected on an individual basis and the "discrete" results are due to rounding. Now that I say that out loud I guess that's kind of the point of the MEPS paper?

In contrast, the `df_Pomeranz` is work I did in New Zealand where every invertebrate was measured to the nearest 0.1 mm and these lengths were converted to dry mass using published length-weight regressions. For example:

```
raw_orig %>%
  filter(dat_id == "df_Pomeranz.xlsx") %>%
  select(site, body_mass, count, ind_n) %>%
  arrange(site, body_mass)
```

```
## # A tibble: 3,189 x 4
##    site    body_mass count ind_n
##    <chr>       <dbl> <dbl> <dbl>
##  1 Italian   0.0027      1  5.56
##  2 Italian   0.0027      1  5.56
##  3 Italian   0.0027      1  5.56
##  4 Italian   0.00281     1  5.56
##  5 Italian   0.00283     1  5.56
##  6 Italian   0.00288     1  5.56
##  7 Italian   0.00291     1  5.56
##  8 Italian   0.00291     1  5.56
##  9 Italian   0.00293     1  5.56
## 10 Italian   0.00296     1  5.56
## # i 3,179 more rows
```

So in this example, I *think* we can consider the body mass continuous, and the `ind_n` is just a way of getting the "counts" on the per m^2 scale. However, I'm keen to get your thoughts on this.

Action Item Do we need to get detailed information from all the original authors on data collection/resolution to infer "bin widths" due to rounding? Is there a way we can simplify this or make assumptions to make it easier to apply to all data sets?

2. *xmin*. Currently, we set *xmin* and *xmax* at the "local" scale. I.e., whatever the smallest and largest body size within a collection (i.e., the `group_id` column which is a unique combination of `dat_id` and `site_date`) is. Is your ongoing work with the histogram mode similar to the Clauset et al. 2009 method?

3. I don't think there was actually a question you were asking me?

4. My results for `group==3572` were exactly the same as what you got. This was using MLEcount without "summing" the body sizes. i.e., the body_mass value of 200 was repeated twice instead of doubling the ind_n value.

jpz's analyses for comparison

Starting out with one dataset to match Andy's initial analyses.

```
raw_simp_1 <- raw_orig %>%
  filter(group_id == 3572) %>%
  select(group_id, body_mass, ind_n)
raw_simp_1
```

```
## # A tibble: 109 x 3
##    group_id body_mass ind_n
##       <dbl>     <dbl> <dbl>
##  1     3572      5325  1.30
##  2     3572       525  1.30
##  3     3572       200  1.30
##  4     3572       325  1.30
##  5     3572      4175  1.30
##  6     3572     68975  1.30
##  7     3572      2500  1.30
##  8     3572      5625  1.30
```

```
##  9     3572     91250  1.30
## 10     3572      3750  1.30
## # i 99 more rows
```

MLEcount

Here is my version of the mlecounts. I used slightly different syntax but we get exactly the same result.

```
mle_count_res <- calcLike(
      negLL.fn = negLL.PLB.counts,
      x = raw_simp_1$body_mass,
      c = raw_simp_1$ind_n,
      p = -1.5,
      vecDiff = 0.5)
```

```
## Warning in nlm(f = negLL.fn, p = p, ...): NA/Inf replaced by maximum positive
## value
```

```
mle_count_res
```

```
## $MLE
## [1] -1.115794
##
## $conf
## [1] -1.193794 -1.038794
```

Comparing with "normal" mle

Here, we can use the `negLL.PLB` method since each individual is represented once (i.e., the original "count" = 1). This won't work for all the datasets because of the difference in sampling effort and scales but we can do it here for comparison.

```
range(raw_simp_1$ind_n)
```

```
## [1] 1.30039 1.30039
```

Since the "counts" (or in this case, `ind_n`) are all the same, we can run a regular mle to compare the results.

```
mle_res <- calcLike(
      negLL.fn = negLL.PLB,
      x = raw_simp_1$body_mass,
      xmin = min(raw_simp_1$body_mass),
      xmax = max(raw_simp_1$body_mass),
      n = length(raw_simp_1$body_mass),
      sumlogx = sum(log(raw_simp_1$body_mass)),
      p = -1.5,
      vecDiff = 0.5)
```

```
## Warning in nlm(f = negLL.fn, p = p, ...): NA/Inf replaced by maximum positive
## value
```

```
mle_res
```

```
## $MLE
## [1] -1.115794
##
## $conf
## [1] -1.204794 -1.027794
```

The estimate is the same, but the CI is a decent bit narrower.

Creating "long" data

Another option is to multiply the number of rows based on the number in `ind_n`. The `uncount()` function cannot take non-integers, but we can multiply the ind_n column by 10 to make 1.3004 = 13.004 and then round. This might not be the best option overall, but just for comparison's sake.

Also, for what it's worth, we did a similar thing in Pomeranz et al. 2022 using the NEON invertebrate data. This was before I knew about MLEcounts and so we duplicated the number of rows based on the estimated abundance and used the "standard" MLE method. For the NEON invertebrate data, all the counts were integers.

```
# expand the data into a long format
simp_long <- raw_simp_1 %>%
  mutate(count_integer = round(ind_n*10)) %>%
  uncount(count_integer)
# head of new data
head(simp_long)
```

```
## # A tibble: 6 x 3
##   group_id body_mass ind_n
##      <dbl>     <dbl> <dbl>
## 1     3572      5325  1.30
## 2     3572      5325  1.30
## 3     3572      5325  1.30
## 4     3572      5325  1.30
## 5     3572      5325  1.30
## 6     3572      5325  1.30
```

```
# dim of original and long data
dim(raw_simp_1)
```

```
## [1] 109   3
```

```
dim(simp_long)
```

```
## [1] 1417    3
```

```
# do the number of rows make sense?
# original had 109 rows, each has a new count of 13 so:
109 * 13
```

```
## [1] 1417
```

```
# same as long data
```

Now, instead of each observation appearing in one row, they are duplicated ~13 times.

mle estimate of long data

```
mle_long_res <- calcLike(
      negLL.fn = negLL.PLB,
      x = raw_simp_1$body_mass,
      xmin = min(simp_long$body_mass),
      xmax = max(simp_long$body_mass),
      n = length(simp_long$body_mass),
      sumlogx = sum(log(simp_long$body_mass)),
      p = -1.5,
      vecDiff = 0.5)
```

```
## Warning in nlm(f = negLL.fn, p = p, ...): NA/Inf replaced by maximum positive
## value
## Warning in nlm(f = negLL.fn, p = p, ...): NA/Inf replaced by maximum positive
## value
```

```
mle_long_res
```

```
## $MLE
## [1] -1.115794
##
## $conf
## [1] -1.139794 -1.091794
```

compare results

```
res_df <- data.frame(method = c("count = 1.3",
                        "mle_long",
                        "mle"),
           estimate = c(mle_count_res$MLE,
                        mle_long_res$MLE,
                        mle_res$MLE),
           conf_lo = c(mle_count_res$conf[1],
                        mle_long_res$conf[1],
                        mle_res$conf[1]),
           conf_hi = c(mle_count_res$conf[2],
                        mle_long_res$conf[2],
                        mle_res$conf[2])) %>%
  mutate(ci_width = conf_hi - conf_lo)
res_df
```

```
##        method  estimate   conf_lo   conf_hi ci_width
## 1 count = 1.3 -1.115794 -1.193794 -1.038794    0.155
## 2    mle_long -1.115794 -1.139794 -1.091794    0.048
## 3         mle -1.115794 -1.204794 -1.027794    0.177
```

All three methods give the same estimate but there is variation in the CI's. the "long" data has the narrowest CIs almost certainly because of the increased sample size (i.e., nrows = 1417 vs. 109).

MLEbin

Andy, please see below and let me know if I'm doing something wrong or misunderstanding

This is based off of the MLEbins recommend file in the SizeSpectra github page.

binData()

First modify the data to put into `binData()` function, then try with non-integer `counts` column

```
raw_for_bin <- raw_simp_1 %>%
  rename(counts = ind_n) %>%
  select(body_mass, counts)
```

The following is not run because it throws an error and prevents knitting the document. I incude it here and the error output below for reference.

```
binData(counts = raw_for_bin,
        binWidth = "2K")
```

Error out put from `binData()`

```
## Error in binData(counts = raw_for_bin, binWidth = "2K") :
## numbers in counts need to be integers in binData;
## for non-integer count see a new function. Currently,
## such a new function has no name [so says Jaqen H'ghar]. Or it may be easier
## to adapt binData.
```

Jagen H'ghar is right, this function does not work for non-integer counts.

Using a weighted histogram function

I found a function which will calculate histograms using non-integer counts/weights. I'm not sure if this is doing exactly the same thing as `binData()` but I did my best to match outputs based on `binData()` function.

```
library(weights)
```

```
## Loading required package: Hmisc

##
## Attaching package: 'Hmisc'

## The following objects are masked from 'package:dplyr':
##
##     src, summarize

## The following objects are masked from 'package:base':
##
##     format.pval, units
```

```
packageVersion("weights")
```

```
## [1] '1.0.4'
```

```
# watchout for dplyr::summarize() being masked by `weights` package.
```

Use weights to bin the data in log2 bins

```
x = raw_simp_1$body_mass
ind_n =  raw_simp_1$ind_n
minx <- min(x)
maxx <- max(x)

  # weighted histogram
wtd_result <- wtd.hist(
    x = x,
    breaks = 2^(floor(log2(minx)):ceiling(log2(maxx))),
    weight = ind_n,
    plot = FALSE)

  wtd_out <- bind_cols(wtd_result[c(2,5)])
  wtd_out <- wtd_out %>%
    mutate(log_counts = log(counts),
           log_mids = log(mids))

  # estimate lambda

mle_binned_res <- calcLike(
    negLL.fn = negLL.PLB.binned,
    p = -1.5,
    w = wtd_result$breaks,
    d = wtd_result$counts,
    J = length(wtd_result$counts),   # = num.bins
    vecDiff = 0.5)
```

```
## Warning in nlm(f = negLL.fn, p = p, ...): NA/Inf replaced by maximum positive
## value
```

```
mle_binned_res
```

```
## $MLE
## [1] -1.084552
##
## $conf
## [1] -1.154552 -1.015552
```

estimate is quite a bit different from the other methods.

```
data.frame(method = c("count = 1.3",
                      "mle_long",
                      "mle",
```

```
                      "weighted_bin"),
           estimate = c(mle_count_res$MLE,
                        mle_long_res$MLE,
                        mle_res$MLE,
                        mle_binned_res$MLE),
           conf_lo = c(mle_count_res$conf[1],
                       mle_long_res$conf[1],
                       mle_res$conf[1],
                       mle_binned_res$conf[1]),
           conf_hi = c(mle_count_res$conf[2],
                       mle_long_res$conf[2],
                       mle_res$conf[2],
                       mle_binned_res$conf[2])) %>%
  mutate(ci_width = conf_hi - conf_lo)
```

```
##         method  estimate   conf_lo   conf_hi ci_width
## 1  count = 1.3 -1.115794 -1.193794 -1.038794    0.155
## 2     mle_long -1.115794 -1.139794 -1.091794    0.048
## 3          mle -1.115794 -1.204794 -1.027794    0.177
## 4 weighted_bin -1.084552 -1.154552 -1.015552    0.139
```

binned estimate is higher (shallower) than others but CI width is slightly narrower except for the "long" data which is likely narrow due to increased sample size.

compare weighted and binned approach

Trying out the `binData()` function on the "long" data format. Here, I multiplied the `ind_n` by 10 and then rounded to make it an integer (i.e., 13) and then used the `uncount()` function to repeat each row 13 times.

```
x.binned <- binData(x = simp_long$body_mass,
                    binWidth = "2k")

num.bins <- nrow(x.binned$binVals)

# bin breaks are the minima plus the max of the final bin:
binBreaks <- c(dplyr::pull(x.binned$binVals, binMin),
               dplyr::pull(x.binned$binVals, binMax)[num.bins])

binCounts <- dplyr::pull(x.binned$binVals, binCount)

MLEbin.res <-  calcLike(negLL.fn = negLL.PLB.binned,
                        p = -1.5,
                        w = binBreaks,
                        d = binCounts,
                        J = length(binCounts),   # = num.bins
                        vecDiff = 1)
```

```
## Warning in nlm(f = negLL.fn, p = p, ...): NA/Inf replaced by maximum positive
## value
## Warning in nlm(f = negLL.fn, p = p, ...): NA/Inf replaced by maximum positive
## value
```

```
MLEbin.res
```

```
## $MLE
## [1] -1.084552
##
## $conf
## [1] -1.106552 -1.063552
```

Comparing all estimates and CIs

```
data.frame(method = c("count = 1.3",
                      "mle_long",
                      "mle",
                      "weighted_bin",
                      "MLEbin"),
           estimate = c(mle_count_res$MLE,
                        mle_long_res$MLE,
                        mle_res$MLE,
                        mle_binned_res$MLE,
                        MLEbin.res$MLE),
           conf_lo = c(mle_count_res$conf[1],
                       mle_long_res$conf[1],
                       mle_res$conf[1],
                       mle_binned_res$conf[1],
                       MLEbin.res$conf[1]),
           conf_hi = c(mle_count_res$conf[2],
                       mle_long_res$conf[2],
                       mle_res$conf[2],
                       mle_binned_res$conf[2],
                       MLEbin.res$conf[2])) %>%
  mutate(ci_width = conf_hi - conf_lo)
```

```
##         method  estimate   conf_lo   conf_hi ci_width
## 1  count = 1.3 -1.115794 -1.193794 -1.038794    0.155
## 2     mle_long -1.115794 -1.139794 -1.091794    0.048
## 3          mle -1.115794 -1.204794 -1.027794    0.177
## 4 weighted_bin -1.084552 -1.154552 -1.015552    0.139
## 5       MLEbin -1.084552 -1.106552 -1.063552    0.043
```

It does appear that something different is ocurring when using the `wtd.hist()` function compared to the MLEbins recommended approach (i.e., `binData()`, then `negLL.PLB.binned()`) Maybe this is a difference in bin edges, or maybe the sample size isn't being correctly imported from the `wtd.host()` function? That might explain why there are wider CI's for that method.

Plotting option

Here, I will use the `mle_count_res` as an example.

In the past, we have played around with this option. BAsically, the steps are as follows:

1. Resample the raw data using the ind_n column as a "weight".

- We set the sample size to be consistent across samples. This makes it easier to compare the results from one sample to another.

2. Create a new "y" variable called `order`. This is done by sorting the body sizes and then dividing by the sample size to make it a probability that x >= x (again for comparison sake).
3. Create a new data frame with the "fitted" values. Use the MLE estimate and confidence intervals to estimate the fitted values across the range of body sizes.

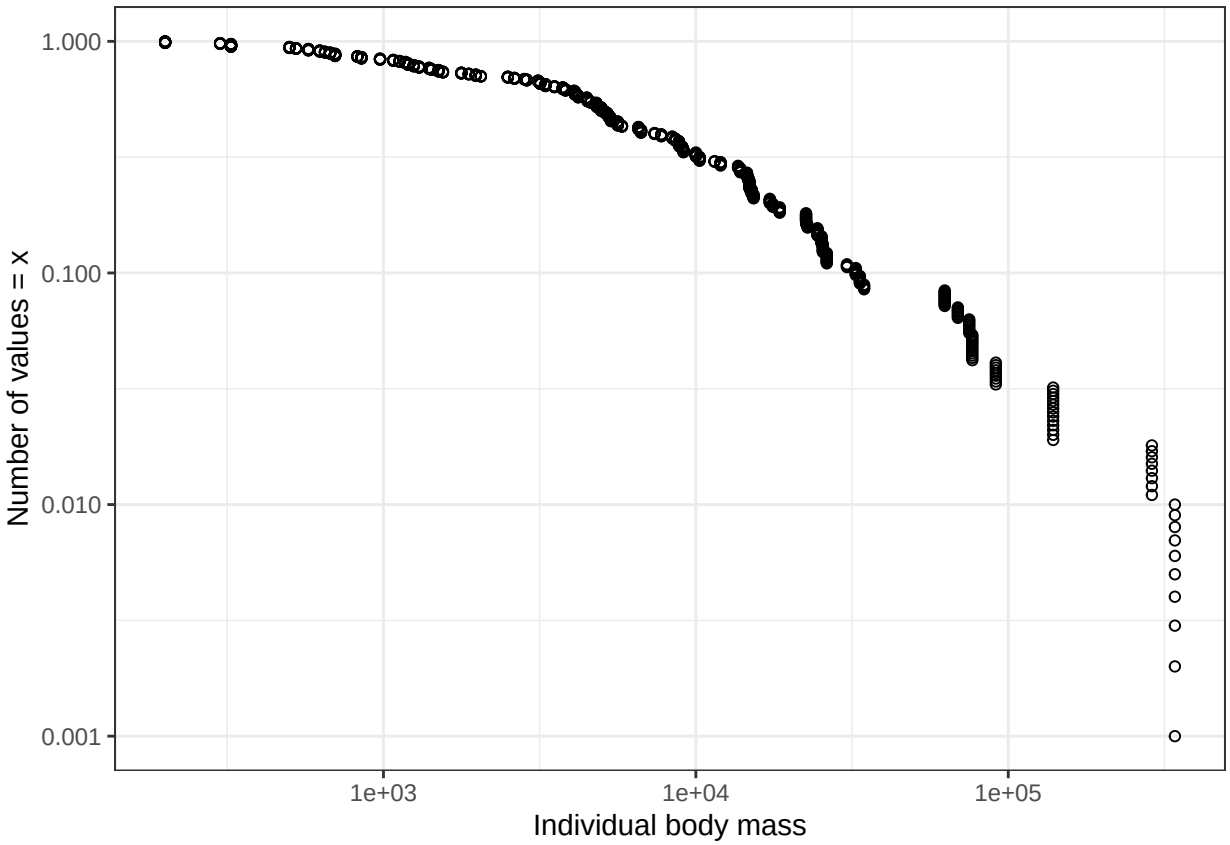
- This is data-heavy. Sequencing all the numbers across the body size range takes a very large vector. There may be a better / more efficient way to do this? For this toy example it's not an issue, but if we applied it to all >3k sites I think my computer would explode.

4. Combine the data frames and plot the raw data points with the "fitted" values.

1 and 2 resampling

```
resampled_df <- raw_simp_1 %>%
  sample_n(size = 1000,
           weight = ind_n,
           replace = TRUE) %>%
  arrange(-body_mass) %>%
  mutate(order = row_number() / 1000)

resampled_df %>%
  ggplot(aes(x = body_mass,
             y = order)) +
  geom_point(shape = 1) +
  theme_bw() +
  scale_x_log10() +
  scale_y_log10() +
  labs(y = "Number of values \u2265 x",
       x = "Individual body mass") +
  guides(size = "none") +
  NULL
```

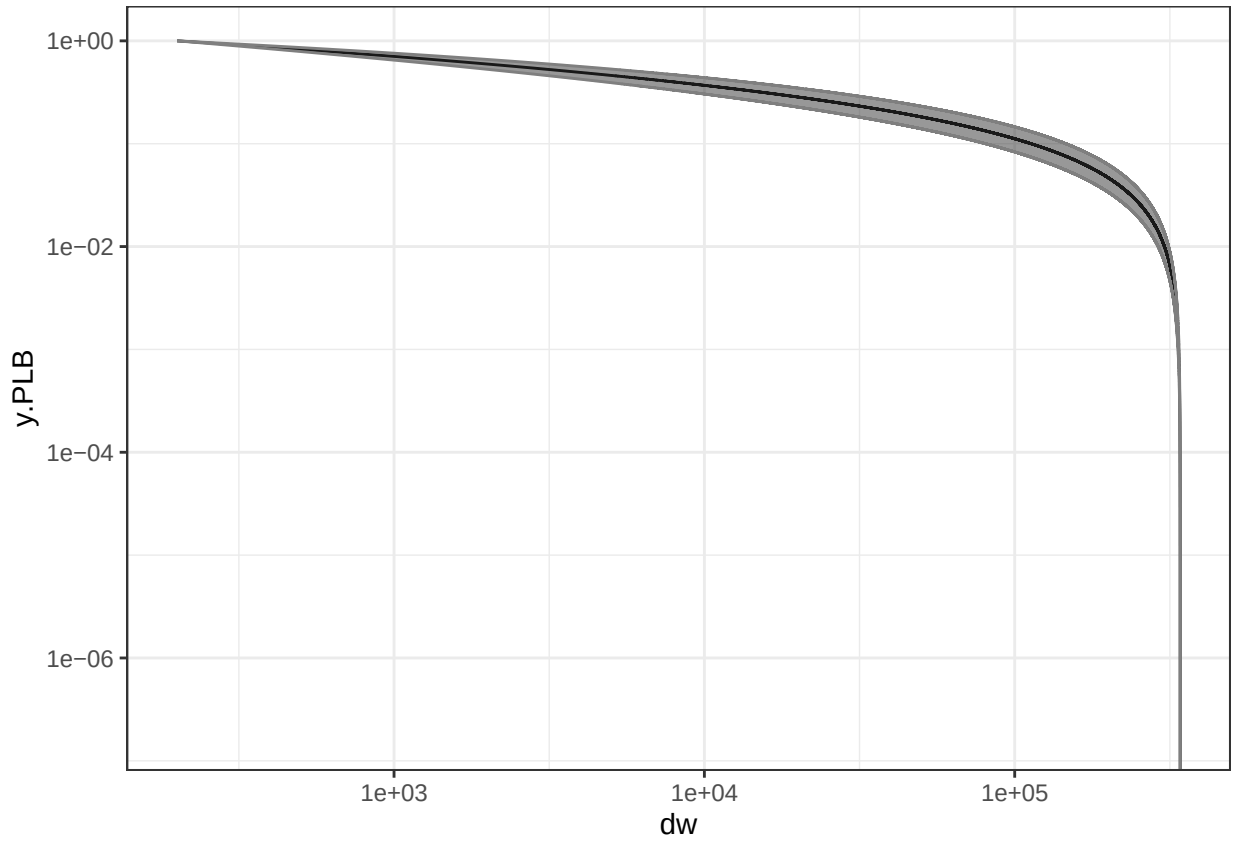


3 Calculate "fitted" data frame

```
xmin = min(raw_simp_1$body_mass)
xmax = max(raw_simp_1$body_mass)
fit_df <- data.frame(
  dw = seq(xmin,
           xmax),
  lambda = mle_count_res$MLE,
  .lower = mle_count_res$conf[1],
  .upper = mle_count_res$conf[2]) %>%
  mutate(
  y.PLB = (1 - (dw^(lambda + 1) - (xmin^(lambda+1)))/(xmax^(lambda + 1) -
                             (xmin^(lambda+1)))),
  y.PLBlower = (1 - (dw^(.lower + 1) -(xmin^(.lower+1)))/(xmax^(.lower + 1) -(xmin^(.lower+1)))),
         y.PLBupper = (1 - (dw^(.upper + 1) - (xmin^(.upper+1)))/(xmax^(.upper + 1) - (xmin^(.upper+1))))

ggplot(fit_df,
       aes(x = dw,
           y = y.PLB,
           ymin = y.PLBlower,
           ymax = y.PLBupper)) +
  geom_line() +
  geom_ribbon(color = "grey50", alpha = 0.5) +
  scale_x_log10() +
```

```
  theme_bw() +
  scale_y_log10()
```



4 Combine and plot

```
resampled_df_dw <- resampled_df %>%
  rename(dw = body_mass)

full_join(fit_df, resampled_df_dw) %>%
  ggplot(aes(x = dw,
             y = order)) +
  geom_point(shape = 1, color = "red") +
  #theme_dark() +
  theme_bw() +
  scale_x_log10() +
  scale_y_log10() +
  labs(y = "Number of values \u2265 x",
       x = "Individual body mass") +
  guides(size = "none") +
  geom_line(aes(x = dw,
           y = y.PLB)) +
  geom_ribbon(aes(
           ymin = y.PLBlower,
```

```
          ymax = y.PLBupper),
   color = "grey50", alpha = 0.5) +
theme_bw() +
NULL
```