Bootstrapping

Martin Schobben

June 3, 2019

The theory behind bootstrapping

The "bootstrap" belongs to the non-parametric statistics family of resampling techniques. The method is coined by Brad Efron in 1979 and published in the Annals of Statistics (Efron 1979). The bootstrap involves n times sampling with replacement of a sample with size n, giving n^n possible bootstrap samples. The bootstrap is merely designed to check how a sample location or variance is affected by random sampling. This is essentially a nice exercise to show the inner workings of the central limit theorem in probability theory. However we do not look at the sampling distribution but the bootstrap distribution. Here the concept of *plug-in* is important. This concept is familiar as we do the same for populations, where the variance of the sample mean is $\sigma/\sqrt(n)$ but as we do not know σ we substitute this by the sample standard deviation s_x (Hesterberg 2015b). The bootstrap is then the next step and can tell us something about the sample distribution.

The special case m-out-of-n bootstrapping for timeseries

One drawback of bootstrap, like any other statistics, is that when n becomes smaller the bootstrap sample becomes a less good approximation of the real observation. In general with smaller bootstrap sample sizes m of a sample sized n would therefore give an overestimation of the variability in the estimator. Nevertheless in the case of time series the latter behavior might actually beneficial, as the time unit t_x of a time series might not be entirely independent of the consecutive element t_{x+1}. Hence bootstrap sampling with weaker dependence than the original sample contains information equivalent to this smaller set of independent observations. As a result, the variability of the bootstrap sample would better mimic the variability of the sample.

More importantly, as the variance and location in the estimate could change with different bootstrap sample sizes m for every time unit t_x, it is better to fix m over the time sequence to give a more internally consistent estimate of every time unit t_x throughout a time series. As a a result, the variance of the bootstrap samples can be more reliably compared, and would not be biased by differences in the sampling intensity at a specific time unit in the time series. We essentially try to answer the *what if* question; what happens to sample estimates if the sample size would have been smaller. Hence, this bootstrap procedure would exclude information that arises from better sampling in specific intervals of the time series, as these better sampled intervals can not be compared accurately with the poorly sampled intervals. This method allows us to determine which estimates of location (mean, median …) and variance can be reliably compared across the time series and exclude a strong control of the sampling density on those parameters.

Application of m-out-of-n bootstrapping with a rolling time window in R

In the following exercise, I will apply a rolling time window to an example time series. On each of the individual windows, I will then apply a m-out-of-n bootstrapping to obtain an estimate. This exercise is performed with R (R Core Team 2019), and the packages tidyverse (Wickham 2017), pangaear (Chamberlain et al. 2018), magrittr (Bache & Wickham 2014) and resample (Hesterberg 2015a)[1]. I will exemplify two

[1] R version 3.6.0 (2019-04-26), with : bibtex 0.4.2, magrittr 1.5, pangaear 0.6.0, resample 0.4, tidyverse 1.2.1.

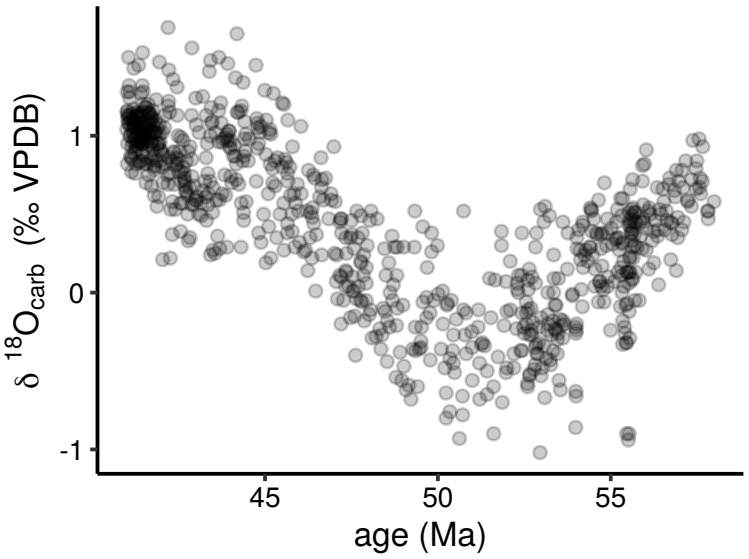

Figure 1: Benthic foraminiferal calcite $\delta^{18}O_{carb}$ for a timeperiod that spans the Eocene up to the Paleocene (Zachos et al. 2008).

different approaches; a vectorized function using tidyverse functions, and a version applying a `for` loop and other base functions.

Example dataset (Zachos et al. 2008)

In this example we look at the geochemical proxy $\delta^{18}O$ measured on foraminiferal shells. These geochemical values give an approximation of past seawater temperatures, which can be ordered according to time. This geochemical record can therefore give us an idea of climate change during past time intervals. Firstly, I will download the (Zachos et al. 2008) data set from the PANGAEA database, and plot the data for a first inspection with a `ggplot()`.

```
# raw data plot
p <-  ggplot(zachos.raw, aes(x = age, y = d18O))
p <- p + geom_point(alpha = .2)
p <- p  + ylab(expression(delta^{18}*O[carb]~"(\u2030 VPDB)"))
p <- p  + xlab("age (Ma)")
p
```

We can easily see that the $\delta^{18}O$ is not evenly distributed over the time sequence, where there seems to be a more dense clutter of data on the left hand side of the plot, and lower sampling densities at 50 Ma. The estimates of sample location and variance for specific time windows could therefore be more-or-less reliable approximations of the total population depending on the window-specific sample size n. Hence, a bootstrap with an adapted bootstrap sample size m could equalize this discrepancy between time windows.

Setting the parameters

For consistency between both the vectorized and for loop approach, we'll fix the most important parameters for the bootstrap. We choose a window size of 2 million year, 10^4 replicates (R) of the bootstrap, and the time grid along which the moving window slides. Here, the latter is based on the maximum range of the time series.

```r
# window size
b.win <- 2 # Myr

# number of replicates (bootstrap samples)
b.reps <- 10^4

# grid (100 kyr increments)
b.grid <- seq(min(zachos.raw$age), max(zachos.raw$age), 0.1)
```

The vectorized version powered by tidyverse and resample

The first version of the *m-out-of-n bootstrap* is vectorized, and is essentially a modified version of `bootstrap()` function of the resample package. This wrapper function requires the following arguments: a data frame with the proxy of interest sorted in the age/depth domain (`df`), an object representing the proxy data (`proxy`) and age (`age`) of the data frame, a double value for the bootstrap sample size (`m`), a double value for the number of replications of the bootstrap (`reps`), and a summary statistic for the replicates of the sample as a character string (`fun.method`). The function also accepts additional arguments to be passed to the statistics function.

```r
# adapted resample function for m-out-of-n boostrapping
mod_boot <- function(df, proxy, age, m, reps, fun.method, ...){

  # enquo() captures a raw expression and its environment
  # (object called: quosure)
  proxy <- enquo(proxy)
  m  <- enquo(m)
  age <- enquo(age)

  # selecting the variables of interest
  m <- df %>% select(!!m) %>%  pull(!!m) %>% unique()
  b.proxy <- df %>% select(!!proxy) %>%  pull(!!proxy)
  b.age <- df %>% select(!!age) %>%  pull(!!age) %>% unique()

  # wrapper for sampler function for the bootstrap funciton
  sampler.m <- function(m, reps){

      samp.bootstrap(n = m , R = reps)
  }

  # function for statistics selection
  method.switch <- function(type){
    switch (type,
      median = expr(median),
      mean = expr(mean),
      IQR = expr(IQR),
      quantile = expr(quantile),
      stop("Unknown type", call. = FALSE)
    )
  }

  # arguments for bootstrap function
  args <- list(data = b.proxy,
               statistic = method.switch(type = fun.method),
```

```
                statisticNames = as.character(method.switch(type = fun.method)),
                R = reps,
                sampler = sampler.m,
                args.stat = list(...))

  # calling the bootstrap() function
  df.raw <- do.call(resample::bootstrap, args)

  # changing the variable name for stat function that require additional
  # arguments
  if(length(args[["args.stat"]]) >= 1) {
    fun.method <- paste0(fun.method, " (", names(args[["args.stat"]]), " = ",
                         args[["args.stat"]], ")")}

  # clean tibble output of the bootstrap, using pluck() to grap relevant
  # information from the list created by the boostrap() function
  tibble(age = b.age,
         stat.value = pluck(df.raw,  "observed"),
         run = paste0("run_", 1:reps),
         d18O = as.vector(pluck(df.raw, "replicates")),
         stat.name = fun.method,
         method = "vectorized")

                        }
```

Prepare the dataframe by introduction of a rolling window

In order to introduce a rolling time window (and thus overlapping windows) on which to apply the *m-out-of-n bootstrap*, we have to convert the initial Zachos et al. (2008) data set. For the vectorized variant, we therefore mainly rely on the base function (`expand.grid()`) and a predicate (`mutate()` in combination with `if_else()`, and `between()`), where the latter function (`between()`) checks whether the values fall in the designed time window.

```
# preparation for dataframe for bootstrapping

# adding unique observation identifier, this as the age variable can contain
# duplicates (replicates from the same sample)
zachos <- zachos.raw %>% mutate(nrow = row_number())

# expand grid and link to the dataset
zachos <- expand.grid(nrow = unique(zachos$nrow), stat.grid = b.grid) %>%
  left_join(. , zachos,  by= "nrow") %>%
  left_join(. , tibble(stat.grid = b.grid, nsample = 1:length(b.grid)),
            by = "stat.grid") %>%
# difference between time axis and defined grid
  mutate(grid.diff = age - stat.grid) %>%
# select observations where difference between time axis and defined grid falls
# in the windowsize
  mutate(selecter = if_else(between(grid.diff, -b.win / 2, b.win / 2),
                            "select", "unselect")) %>%
  filter(selecter == "select") %>%
  select(-c(nrow, selecter, grid.diff)) %>%
# calculation of minimum (m) sample size for the timeseries
```

```
  group_by(stat.grid) %>%
  mutate(n = n()) %>%
  ungroup() %>%
  mutate(min_n = min(n))
```

Applying the function by nesting the windows

The previously designed `mod_boot()` is applied to the modified data frame of Zachos et al. (2008). Here, I use the package purrr and the map function family to apply the function to each time window, which is again wrapped by the `reduce()` function. For a detailed explanation of this working routine, check the purr cheat-sheet: https://evoldyn.gitlab.io/evomics-2018/ref-sheets/R_purrr.pdf.

```
# nesting
boot.raw <- zachos %>%
            nest(-nsample)


# mapping of modified bootstrap function
boot.tidy <- reduce(list(map_dfr(boot.raw$data, ~ mod_boot(df =.x,
                                    proxy = d18O,
                                    age = stat.grid,
                                    m = min_n,
                                    reps = b.reps,
                                    fun.method = "median")),
                         map_dfr(boot.raw$data, ~ mod_boot(df =.x,
                                    proxy = d18O,
                                    age = stat.grid,
                                    m = min_n,
                                    reps = b.reps,
                                    fun.method = "quantile",
                                    probs = 0.25)),
                         map_dfr(boot.raw$data, ~ mod_boot(df =.x,
                                    proxy = d18O,
                                    age = stat.grid,
                                    m = min_n,
                                    reps = b.reps,
                                    fun.method = "quantile",
                                    probs = 0.75))),
                         bind_rows)
```

The base version of the function (for loop)

Now I get to the base variant of the function which employs the for loop (`for()`). This function takes the following arguments: a double value for the number of replications of the bootstrap (`reps`), a double vector of proxy data (`proxy`) and age/depth axis (`time`), a double vector for `grid` which is essentially the grid along which the window crawls, as defined at the very start, a double value for the bootstrap sample size (`m`), where we use the previous defined minimum sample size, a `windowsize` as defined at the very start, and a function for the summary statistics (`method`). The function also accepts additional arguments to be passed to statistics function.

```
# the for loop alternative
```

```
BootSlide <- function(reps, b.proxy, time, grid, m, windowsize, method, ...){

  # create empty matrix to store results
  l0.boot <- matrix(NA, nrow = length(grid), ncol = reps)

  for(i in seq_along(grid)){

    # extracting window specific values of the proxy through indexing
    f <- b.proxy[time >= (grid[i] - (windowsize / 2)) &
                 time <= (grid[i] + (windowsize / 2))]

      for(t in 1:reps){

        set.seed(t * i)

        # the safer wrapper function for vetor length 1, see ?sample()
        resample <- function(x, ...) x[sample.int(length(x), ...)]
        boot <- resample(f, m, replace = TRUE)
        # applying stats
        m1   <- method(boot, ...)

        # storing the stat results
        l0.boot[i, t] <- m1
    }
   }
  return(l0.boot)
}
```

Applying the base function and tidying the output

I use this function with the data of the raw data frame, and compile the out-compete to one long format tibble with `purrr::reduce()`. I calculate the average value for each bootstrap sample with `mutate()` and `rowMeans`. However this can also be done with the base function `apply()`.

```
boot.for <- reduce(
              list(

                # median
                BootSlide(reps = b.reps, b.proxy =  zachos.raw$d18O,
                          time = zachos.raw$age, grid = b.grid,
                          m = unique(zachos$min_n),
                          windowsize = b.win, method = median) %>%
                as_tibble()  %>%
                mutate(stat.value = rowMeans(.)) %>%
                bind_cols(age = b.grid, .) %>%
                set_colnames(c("age", paste0("run_", 1:b.reps), "stat.value")) %>%
                mutate(stat.name = "median", method = "for_loop") %>%
                gather(., key = "run", value = "d18O", 1:b.reps + 1),

                # quantile, probs = 0.25
                BootSlide(reps = b.reps, b.proxy =  zachos.raw$d18O,
                         time = zachos.raw$age, grid = b.grid,
                         m = unique(zachos$min_n),
```

```
                    windowsize = b.win, method = quantile, probs = 0.25) %>%
          as_tibble() %>%
          mutate(stat.value = rowMeans(.)) %>%
          bind_cols(age = b.grid, .) %>%
          set_colnames(c("age", paste0("run_", 1:b.reps), "stat.value")) %>%
          mutate(stat.name = "quantile (probs = 0.25)", method = "for_loop") %>%
          gather(., key = "run", value = "d18O", 1:b.reps + 1),

          # quantile, probs = 0.75
          BootSlide(reps = b.reps, b.proxy =  zachos.raw$d18O,
                  time = zachos.raw$age, grid = b.grid,
                  m = unique(zachos$min_n),
                  windowsize = b.win, method = quantile, probs = 0.75) %>%
          as_tibble() %>%
          mutate(stat.value = rowMeans(.)) %>%
          bind_cols(age = b.grid, .) %>%
          set_colnames(c("age", paste0("run_", 1:b.reps), "stat.value")) %>%
          mutate(stat.name = "quantile (probs = 0.75)", method = "for_loop") %>%
          gather(., key = "run", value = "d18O", 1:b.reps + 1)

          ),

        bind_rows)
```

Combining results and comparing output

Now that I have applied both procedures (vectorized and the for loop), I will plot and compare the results (Fig. 2).

```
# a tibble combing both procedures as long format
boot.comp <- bind_rows(boot.for  %>% select(-d18O) %>%  spread(stat.name, stat.value), boot.tidy %>% sel

# a tibble combing both procedures as wide format for checking co-variance
boot.cross <- bind_rows(boot.for  %>% select(-d18O),  boot.tidy %>% select(-d18O)) %>%  select(-run) %>%

# plots of the bootstrap sample median for both procedures
p <- ggplot(bind_rows(boot.for  %>% filter(stat.name == "median"), boot.tidy %>% filter(stat.name == "me
p <- p + geom_line(alpha = 0.01)
p <- p + ylab(expression(delta^18*O~"(\u2030 VPDB)"))
p <- p + facet_grid(cols = vars(method))
p <- p + geom_line(aes(x = age, y = stat.value, group = run), color = "red")
p
```

Note, that it seems that the confidence intervals for the bootstrap sample location (here median) are different for both approaches. This needs further investigation. However, I will show in the following that this does not affect the general trends of the bootstrap sample location (median) and variance (quantile range).

Linear model fit to compare results

To test how the bootstrap sample location (median) and variance (quantiles) for both methods compare with each other, I perform an analysis of co-variance with a linear model fit. For this, I designed a function that

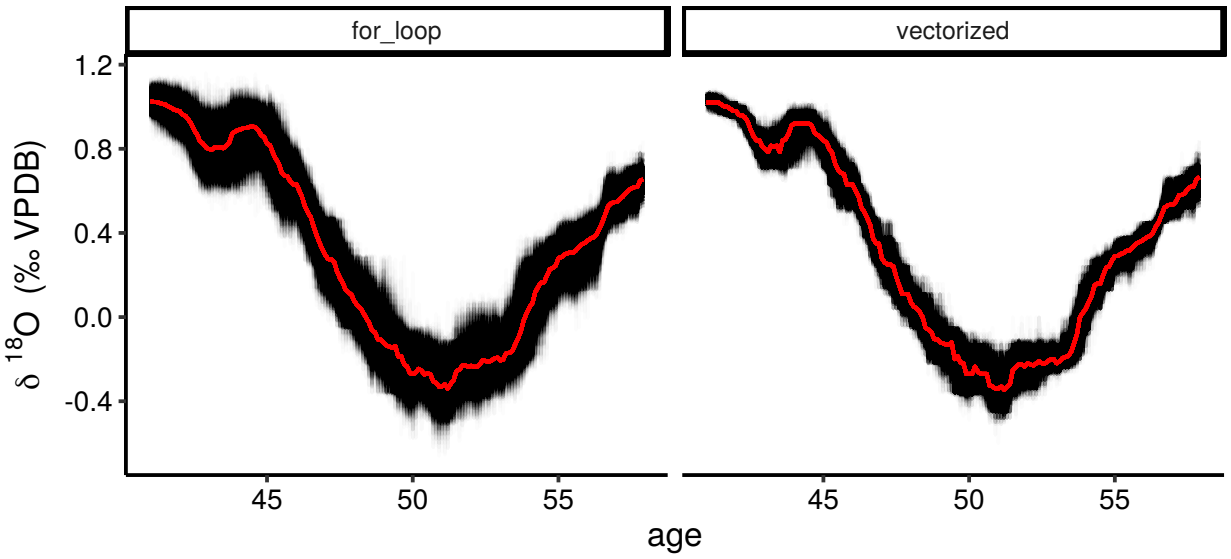

Figure 2: The rolling window m-out-of-n bootstrap sample median for benthic foraminiferal calcite $\delta^{18}O_{carb}$ compared for both procedures, vectorized and for loop.

extracts the coefficients of a linear (`lm()`) fit (i.e., intercept and slope), and the coefficient of determination (R^2).

```
# statistics of linear model fit
boot.lm <- boot.cross %>% group_by(stat.name) %>%
                          nest()


b_fun <- function(mod) {
    broom::tidy(lm(for_loop~vectorized, data = mod )) %>%
    select(term, estimate) %>%
    spread(term, estimate) %>%
    rename(intercept = `(Intercept)`, slope = vectorized) %>%
    bind_cols(., broom::glance(lm(for_loop~vectorized, data = mod))) %>%
    mutate_all(~round(., 2))}

boot.lm <- boot.lm %>% mutate(data = map(data , b_fun)) %>% unnest()




p.c <- ggplot(boot.cross, aes(x = vectorized, y = for_loop))
p.c <- p.c + geom_point(alpha = 0.8)
p.c <- p.c + facet_grid(cols = vars(stat.name), scales = "free")
p.c <- p.c + geom_label(data = boot.lm, inherit.aes=FALSE, aes(x = 0.2, y = 0.9,
                        label = paste("slope =", slope,"\n",
                                      "intercept =",intercept,"\n",
                                       "R2 =",r.squared)),
                        size = 2)
p.c <- p.c + ylab(expression(delta^18*O~"(\u2030 VPDB) (for_loop)"))
p.c <- p.c + xlab(expression(delta^18*O~"(\u2030 VPDB) (vectorized)"))
p.c
```

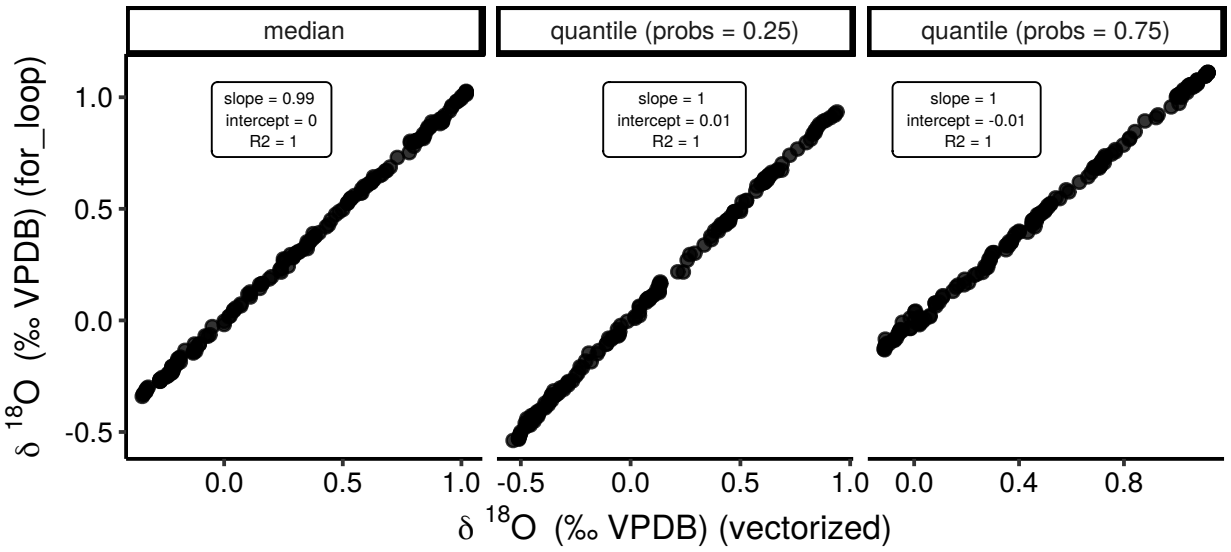

Figure 3: Crossplot of vectorized and for loop variant of m-out-of-n bootstrap for different statistics, along with coefficients of a linear model fit.

Note, that the fit is nearly perfect with only very small biases observed in the intercept and slope, the latter suggest a minimal systematic offset.

Final verdict

Both variants of the *m-out-of-n bootstrap* yield consistent bootstrap sample location (median) and variance (quantiles) for both the vectorized and for loop variant. Thus the results (Fig. 4) give an accurate depiction of average trends and variance throughout the time series unbiased by difference in the sample density. This does, however, not mean that values falling outside of the here depicted interquartile range (Fig. 4) are more prone to be biased.

```
p <- ggplot(boot.comp, aes(x = age, y = median))
p <- p + ylab(expression(delta^18*O~"(\u2030 VPDB)"))
p <- p + facet_grid(cols = vars(method))
p <- p + geom_ribbon(aes(ymin =`quantile (probs = 0.25)`, ymax = `quantile (probs = 0.75)`), fill ="#ec
p <- p + geom_line(color = "#2b8cbe")
p
```

References

Bache SM, Wickham H (2014) Magrittr: A forward-pipe operator for r.

Chamberlain S, Woo K, MacDonald A, Zimmerman N, Simpson G (2018) Pangaear: Client for the 'pangaea' database.

Efron B (1979) Bootstrap Methods: Another Look At The Jackknife. The Annals of Statistics 7:1–26.

Hesterberg T (2015a) Resample: Resampling functions.

Hesterberg TC (2015b) What Teachers Should Know About The Bootstrap: Resampling In The Undergrad-

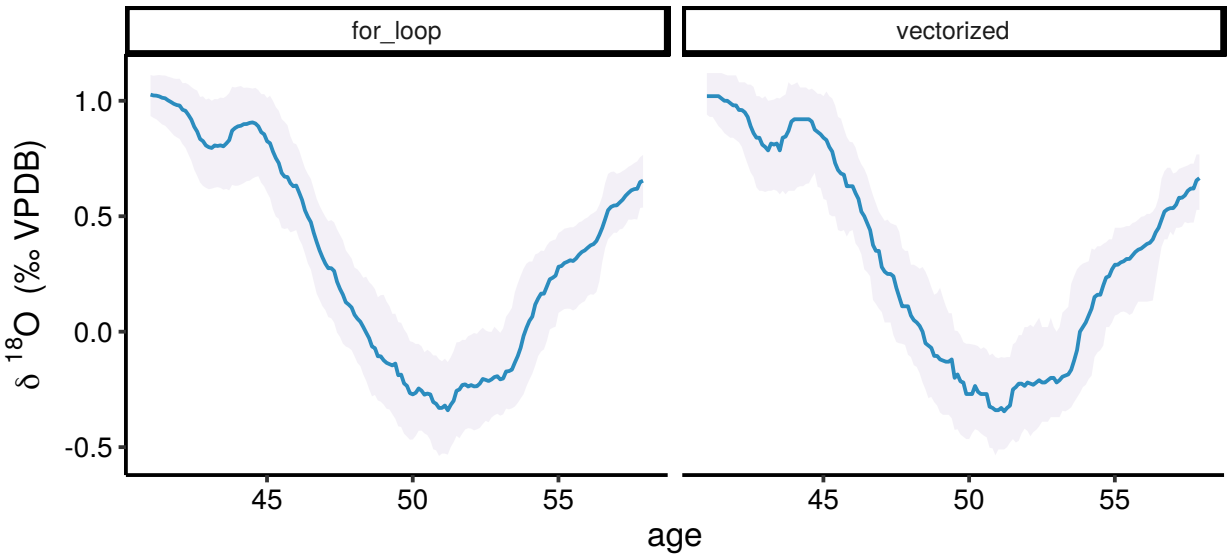

Figure 4: The rolling window m-out-of-n bootstrap sample median and interquartile range for benthic foraminiferal calcite $\delta^{18}O_{carb}$ compared for both procedures, vectorized and for loop.

uate Statistics Curriculum. American Statistician 69:371–386.

R Core Team (2019) R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria.

Wickham H (2017) Tidyverse: Easily install and load the 'tidyverse'.

Zachos JC, Dickens GR, Zeebe RE (2008) An Early Cenozoic Perspective On Greenhouse Warming And Carbon-cycle Dynamics. 451:279–283.