*A markdown file (and whatever compiled variant) that explains what the two functions do and how to use them. This should also contain a short example analysis, including plots and interpretation. The data underpinning the example should also be available in the repository.*

Introduction to the problem

Bootstrapping allows us to calculate a 95% confidence bounds for our estimate without distributional assumptions, by resampling with replacement from the available data. By doing this in high repetition, we can get a good estimate with lower sampling uncertainty. However, to access this high repetition we must make the code fast to run and easy to use.

R

The optimized code is based on the provided sample, and was created by inspecting the code, and identifying the following parts that could be sped up:

* The *linear model* is a slow function to run, could rephrase as a matrix.
* The *rbind* function to be replaced.
* Avoid repetition by taking out the *for* loop and *if* statement
* There are no defined vector sizes, so calculations are repeated more than necessary.
* Similarly we wish to avoid using the function *nrow* multiple times.
* There is only space for a limited number of x covariates, and we wish to extend this.

What is the Maths behind it?

To optimize the linear model and consider multiple covariates, we rewrite the model into a matrix format, using the following theory. A linear model can be written as where is the error.

Writing this in a matrix form or simply .

**

*m* number of columns

Estimating two or more slope coefficients is straightforward using least squares. We find estimates that best fit the data by using .[[1]](#footnote-1)

How does it work?

The optimum function is given by *bestBadBootBrotherAnyCovars3 (nBoot, yDat, …).* This function takes at least three variables:

* nBoot: the number of iterations,
* yDat: the response vector,
* … : one or more vectors that will form the columns of covariate design matrix.

First we set up the parallelization of tasks by creating clusters based on the computer cores. Then we take the input data and form the matrices *X* and *Y* following the model above. We use a helper function *parBootAnyCovars* that bootstraps the data and calculates the estimates for the best fit, and we return as a matrix.

*Note*: we have implemented two optimized functions, which give similar results on reruns. We considered it best to include both for the sake of scalability, as they are built differently and whilst one is initially faster, the other seems to scale in better time.

SAS

The given code is a small and inefficient SAS program to perform nonparametric bootstraps for a regression. Significant efficiencies were made to this code by using the *surveyselect* and the *by replicate* statements and also by adding no print arguments to the proc commands and minimising internal computation. The timing is inserted to determine how much faster does the new code run.

How does it work?

The optimum function is given by the “sasfile” command. To make the given code efficient, the *proc surveyselect* command was implemented. It replaces the initial iteration-based method (the “%do for &NumberOfLoops”) by generating random samples into an enormous dataset all at once as opposed to the one-per-iteration approach.

This function takes several arguments:

* A “seed” is set for reproducibility reasons.
* The “method” is used to specify the type of random sampling. Here we use the Unrestricted Random Sampling (URS) technique.
* The “samprate” replaces the prior need to calculate the number of rows in the dataset by specifying SAS to sample 100% of the original dataset.
* The “rep” allow samplers to create what are known as replicate samples in sampling theory. Here we set it as equal to &NumberOfLoops and will thus make the new dataset (the output of “proc surveyselect”) to the length of :

number of rows \* number of loops

Now that the required dataset from *proc surveyselect* has been constructed, perform a regression on this randomised dataset and generate parameter estimates.

The regressions are performed using *proc reg*. It calculates the required number of regression parameter estimates (coefficients) from the dataset using the *by Replicate.* These coefficients are assigned to a new dataset called “Resultholder2” and renamed the “RandomIntercept” to “Intercept” and also from “RandomSlope” to “XVariable”.

Finally a *proc means* command was made to calculate the mean, the mean estimate of each parameter and their 95% confidence intervals. The output was made by an “ODS Listing” statement to control the saving of the data and when the output is displayed to the user.

Note: Other efficiencies to “*sasfile”,* are readily available to implement if the dataset or sought bootstrapping specifications are more numerous than the current data set and method.

*The sasfile’*s functionality is like an on/off switch for which turning it on entails loading the imported dataset fully into the RAM - as opposed to the less efficient memory usage of repeated memory allocation. However, in so doing this incur a minimal time required for said data transfer which makes the usage of “*sasfile”* only worthwhile if one seeks to perform above approximately 100’000 + iterations for the bootstrapping.  In light of this, and as bootstrapping for approximately 1000 iterations appears to be commonly sufficient to calculate confidence intervals, the “*sasfile”* statement has been disregarded in the current re-coding.

Analysis of Results

\*a short example analysis\*

\*need plot of times of how well each bootstrap function is doing\*

1. Taken from Chapter 3 of MT5753: Statistical Modelling notes from 2017. [↑](#footnote-ref-1)