PhUSE 2012

Paper HE07

Bootstrap Analysis Double-Independent Programming: Issues and Solutions.

Nils Pénard, UCB BIOSCIENCES GmbH, Monheim am Rhein, Germany

ABSTRACT

Bootstrap analysis is a valuable, iterative, statistical analysis tool useful when the distribution of an analysis variable is unknown, and likely non-normal and/or non-symmetrical. This family of techniques is garnering attention in the pharmaceutical industry with the development of faster and more powerful computers. The downside of such iterative analysis emerges when the final bootstrap results are to be included in a submission to regulatory authorities. In order to ensure that the bootstrap results are validated appropriately, double-independent programming is the preferred method. Unfortunately, bootstrap ADaM datasets used for these planned analyses are large, complex, and challenging to produce. They also cause extended processing times reading or writing to storage. Double-programming only magnifies these issues. This paper will present a case study of bootstrap analysis with a PRO efficacy endpoint to illustrate the issues and solutions that arise during double-independent programming validation.

INTRODUCTION

Bootstrapping is an umbrella term that covers a set of simulation-based methods using resampling as a way of approximating population distribution from a sample. This paper will focus on the double-independent programming of the bootstrap-t technique. This technique is useful when a researcher is interested in knowing whether the arithmetic mean difference between two groups is significant, but the usual statistical assumptions are likely not to be met. With this technique, it is possible to compare two means and decide whether they are significantly different. This method is particularly suited to the analysis of Health Economics or Patient Reported Outcomes parameters. This paper will illustrate the method using data coming from the WPS questionnaire.

The Work Productivity Survey (WPS) is a patient reported questionnaire measuring the impact of R.A. on the daily life of patients, both at work and at home (Osterhaus, Richard, & Purcaru, 2008). This questionnaire features eight questions and the patients' answers to each of these questions are considered outcomes. The answers for the WPS questions are usually far from being normally distributed. This precludes the use of the usual statistical tests based on distribution assumption such as a *t*-test.

Bootstrap analyses are often conducted using the boot macro provided on the SAS web site. This macro is very efficient and versatile but presents some inconveniences in the framework of double programming. A first issue is that this macro is not validated by the SAS institute, and is only provided by SAS support. A second issue with this macro is that each re-run of the program will use a different resampling (the SEED parameter is always different): this prevents any kind of validation by double-programming as the final results are likely to differ between two different runs of the program. It also means that the final results are likely to change between each re-run. A third issue is that the bootstrap macro by SAS does not integrate a variance stabilization module. This is an issue because previous analysis on WPS data has shown that bootstrap analysis without variance stabilization was sometimes heavily biased (Osterhaus, Richard, & Purcaru, 2009).

Double-independent programming is a quality control method widely used in the pharmaceutical industry. This method reduces the occurrences of programming errors as well as the uncertainty linked to the interpretation of specifications. Reconciling differences is an iterative process and may be time-consuming depending on the complexity of the output to control. This paper presents the unique challenges emerging from double programming a bootstrap analysis.

This paper presents an example for the bootstrap-t comparison between two treatment groups: a first group of patients taking an active drug (Active) and a second group of patients taking a placebo drug (Placebo). This example presents the case for one question, one time point, and one comparison. It uses an alpha level of 0.05 and B, the number of replication is 10000. In a more realistic setting, it is likely that this bootstrap analysis would entail many more questions, time points, and comparisons. This paper can also be used as a companion to the Barber et al. () paper.

The paper is organized along three chapters. A first chapter demonstrates how to implement the re-sampling step. Then, a second chapter describes how to compute the statistics for each re-sample. And finally, a third chapter explains how to derive the final output of the bootstrap-t: p-value and confidence interval. Each of these chapters present both the issues encountered as well as the solutions that were adopted.

1-RESAMPLING MODULE

The core of the bootstrap technique is to resample an observed sample a large number of times in order to estimate the shape of the distribution of a statistical estimator under the null hypothesis. This can be done by resampling randomly directly from this sample a large number of times. Although there are many different resampling techniques, this paper presents an example of a stratified and balanced resampling with replacement. In this type of resampling, the resampling is conducted independently in each sub groups (e.g., treatment 1, treatment 2 and placebo). This is because the distributions of each of these experimental groups are thought to differ. Because of the replacement aspect, one particular patient may appear several times in one sample and not at all in another one. In order to control for a bias in patient selection, each of the patients will appear an equal number of time. So for instance, if 10000 random samples are created, Patient 1 will appear 10000 times.

Although this type of sampling appears complicated, it is relatively simple to implement. Unfortunately, although SAS (r and 9.1.3) provide PROC SURVEY SELECT, a ready-made procedure specialized in implementing very efficiently many resampling techniques, this procedure does not allow for balanced sampling. However, it can still be used in conjunction with other datasteps to fulfill this criterion. The following section describes how to implement such stratified balanced sampling.

1.1 DESCRIPTION OF THE SAMPLING ALGORITHM

The description of the algorithm of balanced stratified sampling used in this paper can be found in Gleason (1988). In a first step, duplicate and concatenate the observed data as many times as there are of bootstrap replications. Randomly shuffle this replication. Then slice this shuffled vector into subparts that have the same size as the observed group. Figure 1 illustrates how this algorithm works.

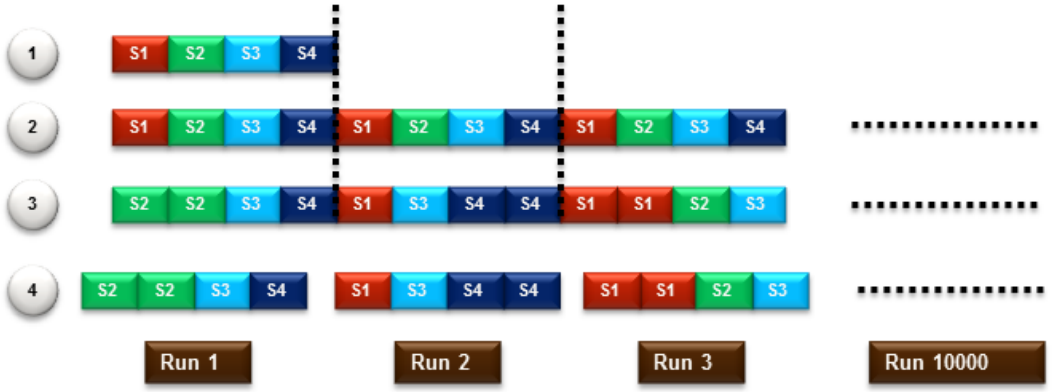


Figure 1: Illustration of the balanced sampling algorithm. Step 1 represents an observed sample for one group made up of four patients (S1, S2, S3 and S4). In Step 2, this observed sample is replicated as many times as there are planned bootstrap runs. In Step 3, the vector representing all of the permutations is randomly permuted. Each of the slices represented in Step 4 correspond to one re-sampling.

The following sections describe the SAS implementation of the Gleason (1988) algorithm as presented by Figure 1.

In order to replicate the original observed data as many times as there are bootstrap replications, it is possible to use PROC SURVEYSELECT. Note that in this paper; by contrast with the common use of this procedure, no random sampling occurs during this step.

```
PROC SURVEYSELECT NOPRINT DATA=indata OUT=sampling01
                  SEED=123 /*Seed of no importance here*/
                  METHOD=SRS
                  SAMPRATE=1
```

```
              REP=10000
;

RUN;
```

This bit of code replicates the dataset `indata` 10000 times. The dataset `indata` corresponds to Step 1 in Figure1 while the result of this replication corresponds to Step 2 in Figure 1. PROCSURVEY allows for an efficient and fast creation of this large vector. An additional benefit of PROCSURVEY is that it produces a very useful variable called "Replicate". This variable labels each of the replicates created by this procedure. This variable will be used to implement the "slicing" (See Step 3 in Figure 1).

Once the large vector has been created, it needs to be randomly permuted (as described at Step 3 in Figure 1). In order to do so, a simple datastep with a SEED statement will suffice. A possible way to implement this shuffling goes as follows:

```
%LET seed=42;

DATA sampling02;
SET sampling01;
rank=ranuni(&seed);
RUN;
```

The random variable `rank` is then used as a sort key to implement the shuffling:

```
PROC SORT DATA=sampling02;
BY rank;
RUN;
```

The following step implements the "slicing" using a MERGE statement. The idea is simply to merge together the original dataset containing the original replicated vector and the `replicate` variable (sampling01, with replicate going from 1 to 10000 and coding for the bootstrap resample), and the corresponding randomly-sorted dataset (sampling02). The implicit MERGE key in this case is the observation rank.

The usual need for a merge key is not necessary in this case for two reasons. A first reason is that the datasets exhibit exactly the same number of observations. The second reason is that the datasets already feature the desired order. Note the use of OPTIONS MERGENOBY = NOWARN to prevent MERGE from displaying the warning that the (bold) user is attempting to merge without a key. Of course, It is a good idea to reinstate this warning once the merge is done (OPTIONS MERGENOBY = WARN). Obviously it is good to exercise caution with this type of merge.

A beneficial side-effect of this method is the avoidance for extraneous PROC SORT or DO loop. Keep in mind that because AdAM datasets store data in a vertical structure, the number of observations can be quite large. This has a significant impact in terms of sorting time performance and/or loops.

```
OPTIONS MERGENOBY=NOWARN; /*Danger Will Robinson*/
DATA sampling03;
MERGE  sampling01 (KEEP = replicate)
       sampling02 (KEEP = usubjid boottrt aval);

RUN;
OPTIONS MERGENOBY=WARN; /*Everything is safe again*/
```

The next PROC SORT is just here to shape the data so that SAS can directly use replicate as a by group factor in the following statistics computations.

```
PROC SORT DATA=sampling03;
BY replicate usubjid;
RUN;
```

1.3 ISSUES WITH DOUBLE PROGRAMMING THE RESAMPLING

The programmers are faced with three main problems in the context of double programming the necessary sampling algorithm. The first one is the random aspect of sampling. The second aspect is to perform a balanced sample. Finally, a third issue results from the large amount of data generated in this step. It is important that both the development programmer and the validation programmer agree on a detailed description of what the program structure should look like. Also, an agreement on the order in which the operations are to be performed will facilitate the validation process.

It is critical that both the development side and the validation side produce the exact same resampling of the data. If this condition is not respected, there will be differences in the final bootstrap results. The program described in the previous section provides a simple example of how to have a controlled randomness that is the same in both programs. This program also takes care of implementing the balancing aspect by following the algorithm proposed by Gleason.

The remaining issue is not so much about programming, but rather about process: validating this type of analysis using double-programming is hard because of the discrepancy between the huge amount of data created by the intermediary steps, and the small size of the final results: indeed, the re-sampling can create millions of observations. The mechanics of the bootstrap reduces this large number of data into a very nimble statistical output: a p-value and a confidence interval. The issue here is twofold: when a difference is found in the final results, it could originate from these millions of observations. This means that the resampling intermediary datasets need to be kept to have a trace of how the data was resampled. The second aspect of this issue originate from the first one: keeping a trace of all the intermediary datasets will quickly become difficult to manage in the framework of a clinical study folder structure.

For instance, in the presented example, the two compared groups are a group made of 80 patients taking an active drug (Active) and a group of 66 patients taking a placebo drug (Placebo). Once the sampling has been implemented, the total number of observations is (80 + 66) * 10000 = 1460000 observations. The corresponding dataset storing these observations has a weight of 70 Mbytes (with 5 variables). Although this may appear as a manageable dataset, keep in mind that the WPS features 8 questions, that there were 7 time points and 3 different comparisons (different active doses vs. placebo, although not presented in this paper). This amounts to a total of 70*8*7*3=11760 Mbytes, representing about 12 Gigabytes of information. Even though the COMPRESS option can be used to reduce the size of these datasets, the size remains significant, the I/O time spent to read and write these datasets will slightly increase, and finally the total number of datasets will remain the same, accumulating in the study folder structure.

A possible way to tract with this issue is to save only one selected intermediary sampling dataset in the temporary area. By default, the resampling dataset corresponding to an arbitrary question, time point, and comparison (e.g., first question, first time point, first comparison) is saved in the temporary area. These parameters can be coded as macro variables and changed later when needed. In addition, a logical switch is inserted in the program, allowing activating and deactivating the saving of this intermediary dataset. If a difference between development and validation is found, then the programmers can start by checking whether they run their bootstrap statistics using the same sampling dataset. Once it is established that the resampling dataset is identical, the saving of this intermediary dataset can be deactivated. If another difference occurs and it is suspected that the sampling is causing it, the saving of the intermediary dataset can easily be implemented again for a different question, time point, and comparison. When the quality control is final, the intermediary dataset can be deleted.

2-STATISTICS MODULE

After the creation of the resampling, the next step consists into computing statistics for each of the B resamples. This step is the most straightforward in the bootstrap process. In this example, the *t* for independent measurements is computed for each of the bootstrap replications (10000 times), as well as for the observed sample (1 time). The set of t coming from the replications offer an estimate of the distribution of the statistical estimator t. Formula 1 describes how the bootstrap t (called t*) is computed where $\hat{\theta}$ is the observed mean difference while $\hat{\theta}_b^*$ is the bootstrap mean difference for replication b. Formula 2 specifies the bootstrap standard error. Note that the computations for $\hat{\theta}$ and $\hat{\theta}_b^*$ can be achieved using PROC TTEST and selecting the output corresponding to the Satterthwaite method. Also note that unfortunately, this procedure does not output the corresponding bootstrap standard error $\widehat{\mathrm{SE}}_b^*$, and that this quantity will be needed later in the process to compute the confidence intervals in the variance stabilized version of the bootstrap (Section 3.2).

$$(1)\ t_b^* = \frac{\widehat{\theta}_b^* - \widehat{\theta}}{\widehat{\mathrm{SE}}_b^*} \qquad (2)\ \widehat{\mathrm{SE}}_b^* = \sqrt{(\mathrm{SD}_{yb}^{*2}/m + \mathrm{SD}_{zb}^{*2}/n)}$$

From the double programming perspective, the programmers on both sides have to insure that they minimize the possibilities of rounding errors. Although the floating point representation of number used by most computer systems today is very powerful, it cannot represent exactly certain quantities. As a consequence of this feature, rounding errors may occur at all level of computations. These errors are often very small but can be amplified in a succession of arithmetical operations.

Although floating point representation errors are certainly not specific to the bootstrap technique, the sheer number of computations required by such technique clearly exacerbates the possibilities of observing such errors.

Rounding error issues may emerge if the programmers do not use the exact same SAS procedure. In such case, it is possible to obtain slightly different results for the same statistics. Another situation where rounding errors are even more likely to occur is when the statistics are not computed using a SAS procedure but rather with a formula. In this case, the programmers will need to agree about the order in which the computations described by the formula are performed in order to avoid different floating point errors on both side. Such errors, although small, have the potential for rendering the final conciliation of the bootstrap results challenging. For instance the author reports a difference occurring at the third decimal point of a p-value. This level of precision is the one reported by most tables for such a variable.

To illustrate the importance of the order in which computations are performed, here are two examples in which a same percentage can be computed. The only difference is the order of execution of the operations.

$$(3)\ \frac{a}{b} \times 100 \quad \text{vs.} \quad (4)\ \frac{a \times 100}{b}$$

In Formula 3, the rounding error produced by the computation a/b is multiplied by a factor of 100. Formula 4 computes the same quantity but does not amplify the rounding error. Multiplication and division are obvious culprits, but it is sometimes difficult to insure that rounding error is kept as small as possible. It is good to keep in mind that combination of other SAS functions (for instance ABS, ROUND, LOG) may also cause rounding errors. The reader can find a good introduction on floating point representation errors and some examples in Montgomery (2008).

3-BOOSTRAP STATISTICS MODULE

The set of t_b^*, represents an approximation of the estimated distribution of the t-statistics for the mean difference of interest under the null hypothesis. Figure 2 presents an example of such distribution. Using this distribution, it is possible to approximate the confidence interval around the observed mean difference $\widehat{\theta}$. This paper will first present how to compute the confidence interval when the bootstrap is known to not be biased (no variance stabilization) and what can affect double programming. Then, a second section will describes additional computational steps when the bootstrap is biased (variance stabilization) and its consequence on double programming

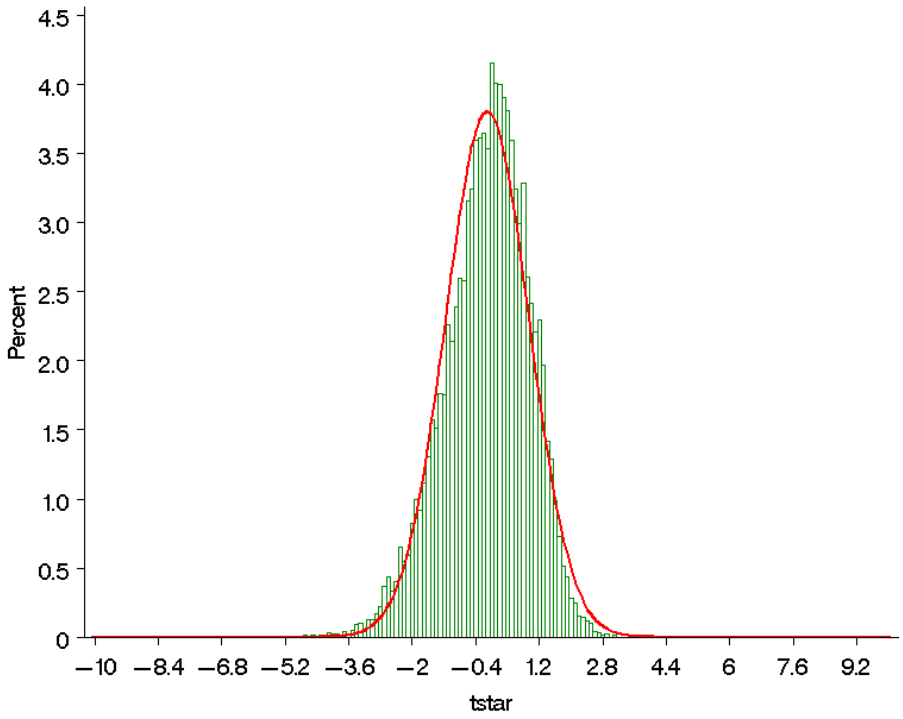


Figure 2: Example of a distribution of bootstrap t^*. In this example, 10000 replications were conducted using WPS data on one question. The Active group had 80 patients and the Placebo group had 66 patients. Note that the distribution looks reasonably symmetrical which indicates that the bootstrap should not be biased.

3.1 CONFIDENCE INTERVALS WITHOUT VARIANCE STABILIZATION

The confidence interval is computed as follows: (5) $(\hat{\theta} - t^*_{\left(1-\frac{\alpha}{2}\right)} S\hat{E}(\theta),\ \hat{\theta} - t^*_{(\alpha/2)} S\hat{E}(\theta))$

$t^*_{\left(1-\frac{\alpha}{2}\right)}$ and $t^*_{(\alpha/2)}$ are the high and the low percentiles respectively of the t^* distribution. These percentiles are found by sorting the t^* values in an ascending order. In this example, the lowest t^* would be called t1 and the hightest t^* would be called t10000. The percentiles are obtained by selecting the $\left(1-\frac{\alpha}{2}\right) \times (B+1)$ th value for the high and the $(\alpha/2) \times (B+1)$ th value for the low percentile. In this example the alpha level is α=0.05 and B is 10000, so $t^*_{\left(1-\frac{\alpha}{2}\right)}$ is approximated by the 9750.975th value, and $t^*_{(\alpha/2)}$ is approximated by the 250.025th value. An obvious issue in this case is that the percentiles are not integer numbers. So for instance, that would mean that the low t percentile would be found between t250 and t251. This implies that the programmers will need to implement some kind of linear interpolation to estimate the t^* percentiles. This added step increases the program complexity and renders the double programming more difficult. An easy way to simplify this aspect is to choose a number of bootstrap replications that will generate integer numbers. For instance, choosing B=9999 instead of 10000, will change the low and the high percentiles to the 250th and 975th values of the distribution. This approach is to be preferred when the bootstrap program does not need to be flexible in terms of replications. If however, the number of replications needed varies, then the added complexity of linear interpolation cannot be avoided.

In the case the use of linear interpolation is required; developing a macro for such a function will greatly facilitate the double programming. It will allow the programmers to focus on the bootstrap itself rather than on accessory aspects of the technique. The development and validation work pertaining to the macro can also be conducted in parallel by other programmers, improving the general efficiency.

3.2 CONFIDENCE INTERVALS WITH VARIANCE STABILIZATION

Previous bootstrap analyses on the WPS data (reference) have shown that it sometimes produced biased results. This bias occurs when the bootstrap mean difference and the bootstrap standard error are not independent. See Figure 3 for an example. Also compute an example and show a difference. As a consequence of this, the confidence interval as it is computed in the above section will be biased as well. A way to eliminate the bias is to transform the data in a way that de-correlates mean and standard error. Barber et al. (2000) describe an approximate variance stabilizing transformation method.

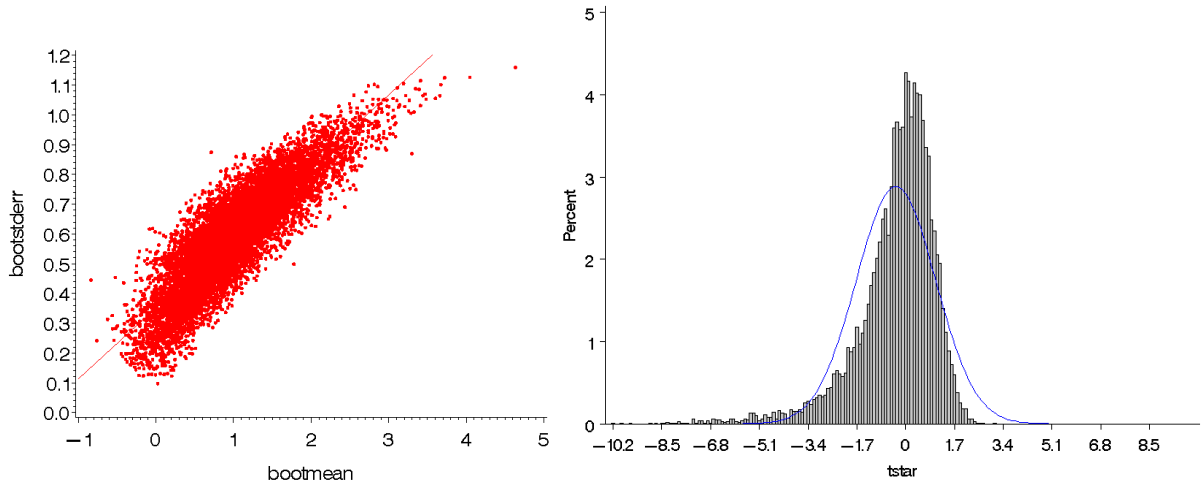


Figure 3: This figure illustrates the case where the bootstrap is biased. The left part of the figure shows the bootstrap mean difference as a function of the bootstrap SE. It shows that these two parameters are strongly correlated. The bootstrap bias caused by this bias is clearly visible on the right part of the figure: the tstar distribution is clearly skewed to the left. Without correction (e.g., variance stabilization), the corresponding bootstrap confidence intervals would be biased.

Step 1: In a first step, estimate the relationship between the bootstrap mean difference and its standard error. Because the relationship is unlikely to be linear, it is advised to use a non-linear technique. Perform a non-linear regression (in our example, lowess) with the bootstrap mean difference $\hat{\theta}_b^*$ as the independent variable and the standard error of the bootstrap mean difference $S\hat{E}(\theta)$ as the dependent variable. The regression function provides a smoothed approximation of the relationship between $\hat{\theta}_b^*$ and $S\hat{E}(\theta)$.
For the process of double programming, it is preferable that the programmers agree upon using the same SAS procedure so as to avoid the complexity of programming non-linear regression. PROC LOESS was used in this paper.

Step 2: Then, the following variance stabilization function is applied (Tibshirani, 1988):

$$g(x) = \int^{x} \frac{1}{s(u)} \mathrm{d}u \qquad (6)$$

In this function, *s*(*u*) corresponds to the non-linear function derived in the step above and *x* corresponds to the bootstrap mean difference $\hat{\theta}_b^*$. Basically, the equation amounts to compute the area under the curve for the inverse of the lowess function. This can be done by using one of many available numeric integration techniques. In this paper, the Simpson method was used, but a much simpler trapezoid method could also have been used. Although SAS provides many integration functions, they are analytical and assume that one knows the nature of the function to integrate. In this case, the integration needs to be performed numerically. As for the linear interpolation step mentioned in Section 3.1, the development and validation of an independent macro performing the integration will greatly facilitate the double programming. In this paper, the integration was implemented following the Simpson method.

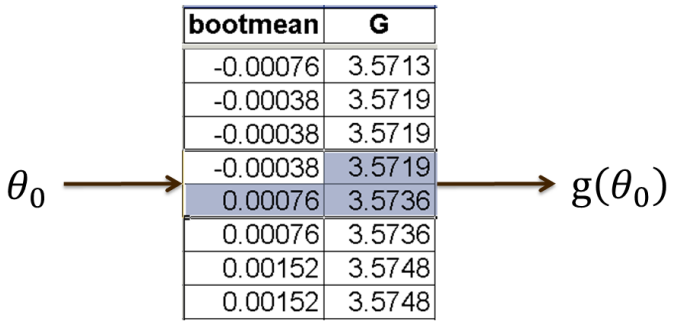
Step 3: Once Step 1 and Step 2 are completed, it is possible to build a look-up table matching THETAS and G(BOOTMEAN). This look-up table is used to implement the g-transformation. A consequence of the transformation is that the transformed bootstrap standard error $g(\widehat{\mathrm{SE}}_b^*)$ can be considered constant and approximately equal to 1. This allows for a simplification of the formulas used previously to compute the t^* and the confidence intervals. Formulas 1 and 5 can be replaced by Formula 7 and 8 respectively:

(7) $t_b^* = \mathrm{g}(\hat{\theta}_b^*) - \mathrm{g}(\hat{\theta})$ (8) $(\hat{\theta} - t^*_{\left(1-\frac{\alpha}{2}\right)}, \hat{\theta} - t^*_{(\alpha/2)})$

The observed t-test (i.e., the t value obtained using the observed sample) will also need to be g-transformed. Although the role of the observed t-test was not mentioned until now, it is used to determine the p-value associated with the confidence interval. How to derive the p-value is described in more details in the next section (Section 3.3). Formula 9 describes the g-transformation of the observed t-test:

(9) $t_{obs} = \mathrm{g}(\hat{\theta}) - \mathrm{g}(\theta_0)$ with $\hat{\theta}$ being the observed mean difference and θ_0 corresponding to the value for the mean difference under the null (in this example $\mathrm{g}(\theta_0) = g(0)$).

The look-up table is used to perform the g-transformations. To operate a g transformation, one select the value x to be transformed, and select the corresponding value g(x). Remember that this look-up table only contains the existing bootstrap mean difference (x). As a consequence it is not likely that the table will contain the exact values for $\hat{\theta}$ and θ_0. As a consequence, a linear interpolation will be needed again to approximate the g-transformation. As highlited in Section 3.1, a validated macro performing linear interpolation will be greatly beneficial to the double programming. Figure 4 offers an example of a g-trasnformation using the lookup table.



bootmean	G
-0.00076	3.5713
-0.00038	3.5719
-0.00038	3.5719
-0.00038	3.5719
0.00076	3.5736
0.00076	3.5736
0.00152	3.5748
0.00152	3.5748

Figure 4: Description of the g-transform function using a lookup table. Note that because $\mathrm{g}(\theta_0)$ is obtained via a linear interpolation because θ_0 does not exist among the 10000 computed mean differences. The same lookup table is used for implementing the g-1 transformation: in this transformation, a value is read in the G column, and its corresponding value in the bootstrap mean difference column is selected.

Step 4: The t^* percentiles are found in the same way as described in Section 3.1.Once the t^* percentiles are found, the confidence intervals in the g-space can be computed. Then, the confidence intervals need to be back transformed to be able to be used with the real observed mean. The back transformation g-1 is implemented with the same lookup table as before, only in the opposite direction. As for the g-transformation, linear interpolation will also be needed for most of the cases.

In summary, when the bootstrap with stabilized variance is the preferred solution, then a sizable amount of additional processing, such as linear interpolation, integration, transformation and back-transformation using a lookup table, is required. Double programming becomes much easier if those processes are implemented as separate macros, and not hardcoded within the bootstrap program itself.

3.3 ESTIMATING THE *P*-VALUE

Estimating the *p*-value follows the same process for both the bootstrap without variance stabilization and for the bootstrap with variance stabilization. The idea is to use the estimated distribution of the t-statistic under the null and

compare it to the observed t-statistic to estimate the p-value associated with the observed mean difference. Formula x is as follows: (10) $\hat{P}_{boot} = \#\{|t_b^*| \geq |t_{obs}|\}/B$. The # symbol means "number of time such event occurs" and B is the number of bootstrap replications. Note that the *p*-value corresponds to a two-sided test: the absolute values of the t-statistics are compared.

In this last step of the bootstrap, the only factor that will affect the double programming process is rounding error. Among the many (10000 in this example) bootstrap replications, it is likely that a certain number of random samples will produce t^* very close (or even exactly equal) to the observed t_{obs}. Sometimes the computed t stars will be only slightly different on each side, but the result of the comparison will be completely different: $t^* > t_{obs}$ and $t^* < t_{obs}$. The formula described above is a discrete transformation: although the difference between t^* and t_{obs} is very small, the difference is amplified by this transformation function. There are several ways to deal with these discrepancies: agreeing on the order of execution of the formulas, agreeing on rounding, and finally using a criterion in the final comparison. As long as the difference between development and validation remain so small that the tfls displaying these results are not affected, all is good. However, despite making these efforts, there is always a risk that a difference will emerge for one specific set of data. In the conclusion, a way of avoiding the effect of these rounding errors is presented.

CONCLUSION

This paper described the issues and solutions encountered while validating a bootstrap-t analysis using double-independent programming. This paper showed the benefits of saving an intermediary dataset containing the information about the sampling, but that it is not necessary to keep this information for all the bootstrap comparisons. The paper also showed the benefit of keeping all aspects of the analysis as modular as possible, be it for the bootstrap method itself (resampling, bootstrap statistic distributions, p-value and confidence interval), or for other generic data-processing functions (linear interpolation, numeric integration, determining the percentiles, variable transformation by lookup tables). This modular approach facilitates the implementation of changes in specification: for instance a new type of sampling algorithm, or a bootstrap on a different statistics (median difference instead of mean difference). The same approach can be applied advantageously to any simulation-based methods calling for very large amounts of computations.

Double-independent programming was a great tool to develop this analysis method, and allowed the programmers to become aware of errors and difficulties by implementing the same algorithm in a sometimes very different fashion. Once quality control is reached using double-independent programming, the next logical step is to transform the programs into a unique validated macro (or a set of validated macros) that can then be used by both the developer and the validator programmers. Having a validated bootstrap macro will not preclude the use of double-independent programming. Even if both sides use the same macro, there are still many opportunities to make mistakes at the level of the data input, or at the level of the macro usage.

Finally, the reader should keep in mind that the most persistent issue during the quality control was caused by floating point representation rounding errors. This paper offered some solutions for dealing with these rounding errors. The ultimate way of avoiding rounding errors is the use of a unique validated macro: rounding errors are eliminated because the computations are performed in the exact same fashion on both sides.

PRO and Health Outcomes are a growing part of the work involved in clinical trials, and as the need for differentiating compounds from competitors increase, the workload increases as well. As mentioned before, these parameters often exhibit distribution answers for which standard statistical tests are not the best-suited analysis methods. As a consequence, it is likely that the use of simulation-based methods will become more frequent. It is well worth investing time now and to think about the structure of such programs, so as to minimize programming time, runtime and facilitate submission to regulation authorities.

REFERENCES

- Barber J.A., & Thompson S.G. (2000). Analysis of cost data in randomised controlled trials: an application of the non-parametric bootstrap. Statistics in Medicine, 19, 3219-3236.

- Gleason J. R. (1988). Algorithms for Balanced Bootstrap Simulations. American Statistician, 42, 263-266.

- Montgomery N. (2008). Floating Point error – what, why and how to!! PHUSE 2008, Paper CS08, http://www.phuse.eu/download.aspx?type=cms&docID=539

- Osterhaus J. T., Richard L., & Purcaru O. (2008). Ann Rheum Dis. 2008; 67 (Suppl II), p. 573-573. Eular Conference (Paris).

- Osterhaus J. T., Richard L., & Purcaru O. (2009). Discriminant validity, responsiveness and reliability of the rheumatoid arthritis specific Work Productivity Survey (WPS-RA). Arthritis Res Ther,11:R73.

- Tibshirani R. J. (1988). Variance stabilization and the bootstrap. Biometrika, 75, 433–444.

ACKNOWLEDGMENTS

I would like to thank Yves Brabant for his help with implementing and validating the bootstrap. I would also like to thank David Friesen, Knut Mueller, Raimund Storb and Wu Yang for their help and insightful inputs.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Nils Pénard
UCB BIOSCIENCES GmbH
Alfred-Nobel-Str. 10
40789 Monheim am Rhein
Germany

Tel: +49.2173.48.1123
Nils.Penard@ucb.com
Web-Site: www.ucb.com