Much Faster Bootstraps Using SAS®

John Douglas ("J.D.") Opdyke,* DataMineIt

Abstract

Seven bootstrap algorithms coded in SAS® are compared. The fastest ("OPDY"), which uses no modules beyond Base SAS®, achieves speed increases almost two orders of magnitude faster (over 80x faster) than the relevant "built-in" SAS® procedure (Proc SurveySelect). It is even much faster than hashing, but unlike hashing it requires virtually no storage space, and its memory usage efficiency allows it to execute bootstraps on input datasets larger (sometimes by orders of magnitude) than the largest a hash table can use before aborting. This makes OPDY arguably the only truly scalable bootstrap algorithm in SAS®.

Keywords: Bootstrap, SAS, Scalable, Hashing, With Replacement, Sampling
JEL Classifications: C13, C14, C63, C88 Mathematics Subject Classification: 62F40, 62G09

Introduction

Since its introduction three decades ago, the bootstrap (see Efron, 1979) has become a ubiquitous statistical procedure, and during that same time period SAS® has grown to become one of the most widely used statistical computing platforms globally.[1] The footprint of researchers and analysts using SAS® to perform bootstraps, therefore, is considerable. Combine this with the exponential growth in the size of individual datasets requiring data-intensive resampling procedures like the bootstrap, across most industries and scientific disciplines, and the large and critical need for increasingly fast SAS® bootstraps quickly becomes apparent.

This paper compares seven bootstrap algorithms coded in SAS®, only one of which requires SAS® modules beyond Base SAS® (the one that uses Proc SurveySelect requires SAS/STAT®). The core of the algorithms obviously is the with-replacement sampling required for a bootstrap, so they are relevant for most data-intensive statistical procedures requiring sampling with replacement. All implement the conventional monte carlo bootstrap, but all can be adapted to perform more involved bootstraps,[2] and all but one (HTIT) can be adapted to bootstrap any statistic besides the sample mean, which is used for convenience in this paper.

Background:
When is O(N) better than O(n) (even when $N>>n$)? Exploiting the Fast Sequential Access of SAS®

Key to this paper is the fact that SAS® is not a "matrix" language (like MATLAB or Gauss) or a "vector"

* J.D. Opdyke is Managing Director of Quantitative Strategies at DataMineIt, a consultancy specializing in applied statistical, econometric, and algorithmic solutions for the financial and consulting sectors. Clients include multiple Fortune 50 banks and credit card companies, big 4 and economic consulting firms, venture capital firms, and large marketing and advertising firms. J.D. has been a SAS® user for nearly 20 years and routinely writes SAS® code faster (sometimes orders of magnitude faster) than SAS® Procs (among them Proc Logistic, Proc MultTest, Proc Summary, Proc NPAR1WAY, and Proc SurveySelect as herein). He earned his undergraduate degree from Yale University, his graduate degree from Harvard University where he was a Kennedy Fellow, and has completed additional post-graduate work as an ASP Fellow in the graduate mathematics department at MIT. Additional of his peer reviewed publications spanning number theory/combinatorics, statistical finance, statistical computation, applied econometrics, and hypothesis testing for statistical quality control can be accessed at www.DataMineIt.com.

[1] See www.SAS.com. With over 45,000 registered user sites, SAS® is arguably the most widely used statistical platform globally, even without including users of the many SAS® "clones" that exist as well (see WPS (from World Programming – see http://teamwpc.co.uk/home), Dap (see http://www.gnu.org/software/dap/ and http://en.wikipedia.org/wiki/DAP_(software)), and arguably Carolina (see DullesOpen.com, a unit of Dulles Research LLC), and formerly, PRODAS (from Conceptual Software) and BASS (from Bass Institute)).

[2] See CherNick (2007) and Davison and Hinkley (1997) for good surveys, and Efron and Tibshirani (1993) for the seminal introduction to the topic.

language (like S-Plus or R), but rather, with a few exceptions, it processes data sequentially, record-by-record. Since the late 1970's SAS® has become extremely fast and efficient at such sequential record processing,[3] and naturally, this strongly shapes the algorithms presented herein. It also leads to a bit of a paradox when comparing the time complexity of these algorithms: the real runtime ranking of these algorithms, when implemented in SAS®, can deviate notably from a ranking based on their theoretical time complexity. In other words, due to SAS®'s fast sequential access, O(*N*) algorithms are often "better" in SAS® than O(*n*) algorithms, even when $N>>n$. While this is important to note, the focus of this paper is real runtimes, and the speed with which SAS® users can obtain actual results. Resource constraints such as I/O speeds, storage space, and memory are discussed below, and CPU runtimes are presented alongside real runtimes in the Results section, but the goal here is not to develop or compare algorithms based on their theoretical time complexity.

The Seven Algorithms

The seven algorithms compared in this paper include: One-Pass, Duplicates-Yes (OPDY), Proc SurveySelect (PSS), HTPS (Hash Table, Proc Summary), HTHI (Hash Table, Hash Iterator), Direct Access (DA), Output-Sort-Merge (Out-SM), and Algorithm 4.8 (A4.8). SAS® v.9.2 code for all the algorithms except the two hash algorithms is presented in Appendix A. All except OPDY and A4.8 have been widely used in the SAS® literature, if not specifically for bootstraps, then for very closely related analyses. Each is completely modular and takes only five macro variables as inputs: the size of the bootstrap samples (*n* observations), the number of bootstrap samples (*m*), the "by variables" defining the strata, the input dataset, and the variable to be bootstrapped. All are described briefly below.

OPDY:
This is a completely new and original memory-intensive algorithm, which is why it is so fast. It uses no storage space, other than the original input dataset and a summary dataset of the record counts for each stratum. OPDY makes one pass through the dataset, record by record, stratum by stratum, and efficiently builds a large array of data values, essentially converting a column of data into a row of data. Then random draws are made from the array using a uniform random variate, *u*, drawn from 1 to *N* (where *N* is the size of the current stratum). Bootstrap results from each stratum are saved in cumulated macro variables to minimize the intermediate memory requirements of outputting a dataset in the data step. The only constraint on this algorithm is *not* the total number of records across all strata but rather, the number of records in the largest stratum, which determines the size of the array held in memory. Yet even on a system with very little memory (2 gigabytes of RAM), the algorithm is able to process input datasets where the largest stratum contains over 100 million records, so this should not be a constraint for most users and most practical uses (the hashing algorithms described below sometimes crash with a largest stratum size of only about three to four million records, in datasets with only two "by variables" defining twelve strata). With more reasonable memory resources (e.g. 16 gigabytes of RAM), OPDY can process datasets where the largest stratum is many billions of records.

PSS:
PSS uses the built-in SAS® procedure, Proc SurveySelect, to create a sampling variable in the original dataset indicating how many times record *i* is selected into the with-replacement random sample, and multiple bootstrap samples can be created automatically for multiple strata. These are then summarized by another procedure to obtain the desired bootstrap statistics. According to SAS® Documentation, Proc SurveySelect uses Floyd's Algorithm (see Bentley and Floyd, 1987, and SAS® OnlineDoc 9.2, 2007).[4]

[3] To quote the author of a previous SAS® bootstrap paper that extols the virtues of PROC SurveySelect, "Tools like bootstrapping and simulation are very useful, and will run very quickly in SAS .. if we just write them in an efficient manner." (Cassell, 2007).

[4] Note that PROC PLAN was not tested because it does not sample with replacement, and PROC MultTest was not tested because it conducts only *n*-out-of-*n* bootstraps and does not allow the user to define the size of the bootstrap sample (aside from changing the

DA:
DA is merely code for directly accessing specified records within a SAS® dataset. A uniform random variate, *u*, is drawn from 1 to *N* (the size of the stratum), and record # *u* is selected from the dataset using the point= option on the set statement. In a nested loop, this is done *n* times to obtain a single bootstrap sample, and then the entire process is repeated *m* times to obtain *m* bootstrap samples. The bootstrap samples do not need to be summarized after the data step because the bootstrap statistics are created sequentially, as each bootstrap sample is created, saved in an array, and then summarized by functions applied to the array when the end of the stratum is reached, all within the data step. DA requires virtually no storage space beyond the original input dataset.

Out-SM:
Out-SM also uses a uniform random variate, *u*, drawn from 1 to *N* to identify the records to include in the samples, but instead of using direct access it creates a new dataset by outputting, in a nested loop, one record for each record # identified by *u*, looping *n* times to create one bootstrap sample. This is done *m* times in a nested loop to create *m* bootstrap samples, while keeping track of both the record numbers (*u*) and the number of the bootstrap sample (*m*). This new dataset is then sorted and merged with the original dataset (in which a record counter was created) to obtain the actual data records for each bootstrap sample. These are then summarized at the level of the bootstrap sample.

HTHI:
The Hash code used in HTHI follows two published SAS® papers. The structure of the code is essentially the same as that of Out-SM, except that a hash table, instead of a merge statement, is used to merge the original dataset and the bootstrap sample dataset (see Secosky and Bloom, 2007), and a hash iterator, instead of Proc Summary, is used to summarize the bootstrap samples (see Secosky and Bloom, 2008). This algorithm is memory-intensive, as well as being storage-intensive, but it is fast. However, it has two major drawbacks: first, the hash iterator can only sum the values of the variable being bootstrapped, so if the statistic of interest requires more complex calculations, additional or separate data steps and/or procedures would be required (this is handled by HTPS, which uses a Proc Summary instead, but which is slightly slower as a result). Secondly, the memory constraints of hashing are far greater than those of OPDY, causing it to crash under dataset sizes much smaller than those OPDY can handle (this constraint also applies to HTPS below). For example, using input datasets with two "by variables" defining twelve strata, OPDY can handle strata with over 100 million records, while both hashing algorithms often crash when only three to four million records per strata are used. And of course, since the size constraint on OPDY is the number of records in the largest stratum and *not* the number of records overall in the dataset, it actually can handle datasets orders of magnitude larger than those the hashing algorithms can handle, as long every stratum is below a certain size.

HTPS:
This algorithm is identical to HTHI except that Proc Summary is used instead of the hash iterator. While slower than the hash iterator, Proc Summary allows the user to bootstrap any statistic, not just those based on the sum of the variable of interest.

A4.8:
I believed I had discovered this with-replacement sampling algorithm, but it is, in fact, listed as Algorithm 4.8 in Tillé (2006), who calls it "neglected." The original source of the algorithm is unknown to Tillé (see email correspondence, 09/30/10), and no proof of its validity is provided in Tillé (2006), so a proof is provided herein in Appendix B. The main advantage of this algorithm is that it requires virtually no storage space beyond the original input dataset (no extra bootstrap "sample" dataset is created, as in Out-SM and the two hashing

---

size of the input dataset, which would require an extra randomization step which would defeat the purpose of conducting efficient sampling with replacement). And the SASFILE statement used with PROC SurveySelect (see Cassell, 2007) is useless when the datasets to be bootstrapped are too large for the extant memory – and that is the only time that fast bootstraps really are needed.

algorithms), and it has relatively low memory requirements. In addition, no nested loops are required, like the n x m nested loops required for DA, Out-SM, and the hashing algorithms: A4.8 requires only a single pass through the dataset, with one loop performed m times on each record, to generate the desired statistic for all m bootstrap samples.

Although not noted elsewhere, the algorithm closely follows the structure of Bebbington (1975), one of the first and most straightforward sequential-sampling-without-replacement algorithms. Like Bebbington (1975), the algorithm is sequential, it requires that N is known ahead of time, and it makes exactly n selections from N items, each with equal probability. It makes one important modification to Bebbington (1975) that transforms it from a without-replacement algorithm to a with-replacement algorithm: it uses binomial random variates instead of uniform random variates. Below, for clarity, both Bebbington and A4.8 are presented.

1. Initialize: Let $i \leftarrow 0$, $N' \leftarrow N + 1$, $n' \leftarrow n$
2. $i \leftarrow i + 1$
3. If $n' = 0$, STOP Algorithm
4. Visit Data Record i
5. $N' \leftarrow N' - 1$
6. Generate Uniform Random Variate $u \sim$ Uniform[0, 1]
7. If $u > n' / N'$, Go To 2.
   Otherwise, Output Record i into Sample
   $n' \leftarrow n' - 1$
   Go To 2.

Bebbington's (1975) algorithm above guarantees that 1) all possible samples of size n drawn from N are equally likely; 2) exactly n items will be selected; 3) no items in the sample will be repeated; and 4) items in the sample will appear in the same order that they appear in the population dataset. The A4.8 algorithm presented below guarantees 1), 2), and 4), and instead of 3), it allows duplicates into the sample.

1. Initialize: Let $i \leftarrow 0$, $N' \leftarrow N + 1$, $n' \leftarrow n$
2. $i \leftarrow i + 1$
3. If $n' = 0$, STOP Algorithm
4. Visit Data Record i
5. $N' \leftarrow N' - 1$
6. Generate Binomial Random Variate $b \sim$ Binomial(n', $p \leftarrow 1/N'$) [5]
7. If $b = 0$, Go To 2.
   Otherwise, Output Record i into Sample b times
   $n' \leftarrow n' - b$
   Go To 2.

The only difference between the code implementation of A4.8 in Appendix A and its algorithm definition above is that the check of $n' = 0$ on line 3. is excluded from the code: execution simply stops when all records in the dataset (stratum) are read once through, sequentially. This is more efficient for these purposes, since all bootstrap samples are being created simultaneously, and the likelihood that all of them will be complete before that last record of the dataset (stratum) is read is usually very small – not nearly large enough to justify explicitly checking for it on line 3.

[5] Using A4.8, p will never equal zero. If $p = 1$ (meaning the end of the stratum (dataset) is reached and $i = N$, $N' = 1$, and $n' = 0$) before all n items are sampled, the rand function b=rand('binomial',p,n') in SAS® assigns a value of n' to b, which is correct for A4.8.

Results

The real and CPU runtimes of each of the algorithms, relative to those of OPDY, are shown in Table 1 below for different #strata, *N*, *n*, and *m* (for the absolute runtimes, see Table B1 in Appendix B). The code was run on a PC with 2 GB of RAM and a 2 GHz Pentium chip. OPDY dominates in all cases that matter, that is, in all

Table 1: Real and CPU Runtimes of the Algorithms Relative to OPDY for Various *N*, #strata, *n*, and *m*
(ex = excessive, cr = crashed)

				REAL						CPU					
N	**#**								**Out-**						**Out-**
(per stratum)	***strata***	***n***	***m***	**PSS**	**A4.8**	**HTPS**	**HTIT**	**DA**	**SM**	**PSS**	**A4.8**	**HTPS**	**HTIT**	**DA**	**SM**
10,000	2	500	500	5.3	10.2	9.0	4.5	8.8	7.5	2.6	10.1	3.3	2.1	3.8	6.0
100,000	2	500	500	23.3	83.3	6.2	4.6	7.3	9.7	18.0	81.4	7.3	2.4	3.1	4.4
1,000,000	2	500	500	31.1	121.2	2.5	1.9	1.4	4.2	25.0	119.3	1.2	1.2	0.7	1.8
10,000,000	2	500	500	42.2	164.9	2.5	2.7	0.7	7.6	33.4	162.0	1.4	1.4	0.4	1.8
10,000	6	500	500	8.4	16.8	10.0	6.8	13.8	14.3	5.8	16.6	4.5	3.6	6.0	7.6
100,000	6	500	500	23.7	84.0	4.7	4.3	7.4	8.7	18.9	82.7	2.8	2.5	3.3	4.4
1,000,000	6	500	500	37.3	145.0	2.3	2.1	4.4	4.5	30.4	143.2	1.4	1.4	0.8	2.2
10,000,000	6	500	500	39.8	370.8	cr	cr	7.8	8.8	32.5	349.1	cr	cr	0.4	1.8
10,000	12	500	500	11.4	23.7	17.9	28.0	18.1	16.8	8.5	23.1	5.8	4.6	8.0	9.6
100,000	12	500	500	31.4	96.3	9.4	9.0	8.7	8.6	24.7	94.2	3.2	2.7	3.4	5.0
1,000,000	12	500	500	47.0	160.7	3.8	3.3	10.1	4.2	38.1	155.8	1.7	1.6	0.9	2.2
10,000,000	12	500	500	45.0	ex	cr	cr	11.0	7.1	36.8	ex	cr	cr	0.4	1.8
10,000	2	1000	1000	5.3	7.3	7.2	5.7	11.2	12.4	3.6	7.0	3.7	2.8	5.3	7.2
100,000	2	1000	1000	25.7	80.4	8.5	7.4	14.3	16.0	19.9	78.1	4.7	3.6	6.4	8.4
1,000,000	2	1000	1000	50.0	183.9	3.3	2.5	8.9	4.4	39.9	179.7	1.9	1.6	1.6	2.7
10,000,000	2	1000	1000	65.2	1428.7	2.5	2.3	9.1	6.2	52.1	1218.7	1.2	1.1	0.5	1.6
10,000	6	1000	1000	8.2	10.7	18.3	17.9	16.9	26.8	5.3	10.4	5.4	4.3	7.6	10.7
100,000	6	1000	1000	28.3	87.8	14.5	13.6	15.6	25.5	21.5	86.4	5.3	4.4	6.3	9.5
1,000,000	6	1000	1000	56.8	214.4	5.2	4.8	4.2	8.5	46.0	211.3	2.2	2.0	1.7	3.2
10,000,000	6	1000	1000	63.6	ex	cr	cr	9.5	6.9	51.0	ex	cr	cr	0.5	1.5
10,000	12	1000	1000	11.2	15.8	19.6	18.0	24.3	37.8	8.0	15.3	8.4	6.5	10.7	16.4
100,000	12	1000	1000	15.1	47.7	6.1	6.0	8.7	12.2	11.8	47.2	2.9	2.2	3.6	5.3
1,000,000	12	1000	1000	61.1	213.6	4.4	4.8	10.1	8.8	50.4	210.6	2.3	2.0	1.8	3.3
10,000,000	12	1000	1000	63.1	ex	cr	cr	10.6	6.0	51.7	ex	cr	cr	0.5	1.4
10,000	2	2000	2000	10.6	8.8	27.0	25.6	28.6	49.2	7.6	8.6	9.3	7.1	12.4	22.2
100,000	2	2000	2000	14.2	39.1	12.5	14.8	14.4	20.8	10.8	38.2	4.5	3.5	5.8	10.5
1,000,000	2	2000	2000	45.6	167.9	5.3	5.6	6.5	11.6	36.5	166.2	2.6	2.1	2.6	4.8
10,000,000	2	2000	2000	87.2	ex	3.4	2.8	8.5	6.1	69.4	ex	1.3	1.1	0.7	1.8
10,000	6	2000	2000	13.5	10.0	29.6	20.8	32.0	92.9	8.5	9.8	11.0	8.1	13.4	26.1
100,000	6	2000	2000	23.9	63.7	19.5	17.0	23.1	50.5	18.0	62.8	7.4	5.9	8.9	17.6
1,000,000	6	2000	2000	62.6	230.3	7.4	6.6	8.8	32.1	50.0	224.2	3.6	3.0	3.3	7.0
10,000,000	6	2000	2000	85.9	ex	cr	cr	7.8	12.7	68.2	ex	cr	cr	0.7	1.8
10,000	12	2000	2000	13.1	10.2	27.1	24.3	33.1	85.6	9.1	10.0	11.6	8.6	14.0	28.1
100,000	12	2000	2000	26.4	63.5	17.3	15.9	25.4	55.4	18.5	62.5	8.2	6.2	10.1	19.0
1,000,000	12	2000	2000	62.2	207.9	7.0	6.6	8.1	21.0	49.7	205.2	3.6	2.9	3.0	6.2
10,000,000	12	2000	2000	50.2	ex	cr	cr	6.0	3.8	40.5	ex	cr	cr	0.4	1.0

cases where the absolute runtimes are not trivial (i.e. under one minute).[6] This runtime efficiency gap grows

[6] In the only case where any of the algorithms is faster than OPDY (relatively few records (#strata = 2, *N* = 10,000,000) and relatively few bootstrap samples required (*n* = *m* = 500)), DA is faster by less than 15 seconds of real runtime. In absolute terms, however, total runtimes for both OPDY and DA are under a minute, making this irrelevant to solving the problem of developing fast, scalable

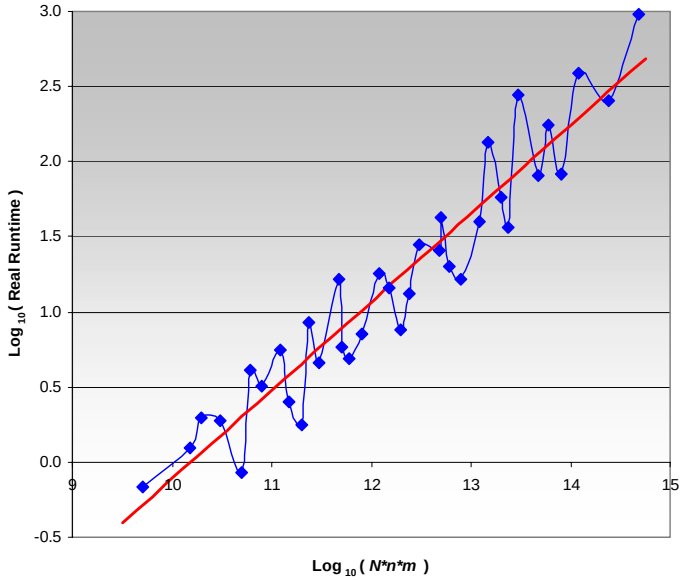
dramatically in absolute terms for OPDY vs. most of the other algorithms as #strata, *N*, *n*, and/or *m* increase (to take just one example, when the input dataset contains 6 strata with only *N* =1,000,000 records each, and *n* = *m* = 2,000, the real runtime of OPDY is 36 seconds, compared to real runtimes of PSS (almost 38 minutes), A4.8 (about 2 hours, 19 minutes), HTPS (almost 5 minutes), HTPI (almost 4 minutes), DA (almost five and a half minutes), and Out-SM (over 19 minutes)). That OPDY so convincingly beats DA is mildly surprising given how widely known, how used, and how old DA is. Similarly, PSS, A4.8, and Out-SM never are serious competitors against OPDY. And even the algorithms that keep up better with OPDY, namely the two hashing algorithms, crash under dataset sizes that OPDY handles easily. As mentioned above, using input datasets with two "by variables" defining twelve strata, OPDY can handle multiple strata with slightly over 100 million records each, while both hashing algorithms often crash when the input dataset contains only three to four million records per strata. And of course, since the size constraint on OPDY is the number of records in the largest stratum and *not* the number of records overall in the dataset, it can actually handle datasets orders of magnitude larger than those the hashing algorithms can handle, as long every stratum is below a fixed size. Obviously memory constraints for the hashing algorithms can be avoided with more memory, but it is the *relative* performance for a fixed amount of memory that matters, and OPDY's more efficient use of memory clearly trumps that of the hashing algorithms.

Aside from speed, note, too, that PSS, the two hashing algorithms, and Out-SM have the added disadvantage of requiring storage space for the bootstrap sample datasets, which can become prohibitively large if #strata, *N*, *n*, and/or *m* are large. All of these results combined arguably make OPDY the only truly scalable bootstrap algorithm in SAS®.

Focusing on the real runtime of OPDY and its relationship to #strata, *N*, *n*, and *m* (see Graph 1), the empirical runtimes in Table B1 yield a reasonably accurate approximation in (1), so OPDY appears to be O(*N***n***m*), which is sensible.

$$\text{Log}_{10}(\text{Real Runtime}) = -5.99228 + 0.588164 * \text{Log}_{10}(N*n*m) \text{ (where } N = \text{all } N \text{ across strata)} \quad (1)$$

Graph 1: OPDY Real Runtime by *N*n*m* (*N* = all strata)



bootstraps. DA fails on this front when absolute runtimes are large enough to actually matter. To take just one example, when the input dataset contains 12 strata with only 100,000 records each, but 2,000 bootstrap samples of size *n* = 2,000 are drawn, OPDY runs in under 26 seconds real time, while DA runs in about 11 minutes real time; if hundreds of such datasets require the application of bootstraps, DA becomes runtime prohibitive and is excluded as a viable option.

Conclusions

The goal of this paper was to develop a non-resource intensive, with-replacement sampling algorithm to execute bootstraps faster than its competitors on arguably the most widely used statistical software platform globally. The OPDY algorithm accomplishes this objective. It uses negligible storage space, and on the SAS® platform, it is much faster than any other alternative, including the built-in SAS® Procedure designed (with Floyd's algorithm) specifically for this purpose. It is even much faster than hashing, and simultaneously does not have the large storage requirements of hashing. Most importantly, the hashing algorithms crash under dataset sizes much smaller than those which OPDY can handle, making OPDY the only truly scaleable bootstrap algorithm in SAS®.

That said, it should be noted that the algorithm runtimes presented herein, and in any empirical algorithm research for that matter, obviously very much depend on the hardware configurations on which the algorithms are run. Very fast I/O speeds on grid platforms, for example, may well allow the hashing algorithms to close the speed gap on OPDY. However, it is also likely that such platforms will also have comparably souped-up memory resources, and thus, OPDY may retain or even increase its speed lead: all else equal, as a rule, in-memory processing will always be faster than I/O processing, and OPDY does not need to create and write to disk the bootstrap sampling dataset that the hashing algorithms "merge" with the original input dataset. In fact, preliminary runs on such systems (with large amounts of memory) show dramatic (relative) speed gains for OPDY, as well as for A4.8. The code provided herein allows for testing and comparisons on different platforms, and this author welcomes feedback from readers regarding the relative performance of the algorithms across an array of configurations.

Acknowledgments

I sincerely thank Nicole Ann Johnson Opdyke and Toyo Johnson for their support and belief that SAS® could produce a better bootstrap.

Appendix A

SAS® v.9.2 code for the OPDY, PSS, A4.8, DA, and Out-SM algorithms, and the SAS® code that generates the datasets used to test them in this paper, is presented below.

```
************************************************************************
************************************************************************
PROGRAM:  MFBUS.SAS

DATE:     9/13/10

CODER:    J.D. Opdyke
          Managing Director, Quantitative Strategies
          DataMineIt

PURPOSE:  Run and compare different SAS bootstrap algorithms including OPDY, PSS, DA,
          Out-SM, and A4.8.  See "Much Faster Bootstraps Using SAS" by J.D. Opdyke for
          detailed explanations of the different algorithms.

INPUTS:   Each macro is completely modular and accepts 5 macro parameters:
          bsmp_size = number of observations in each of the bootstrap samples
          num_bsmps = number of bootstrap samples
          indata    = the input dataset (including libname)
          byvars    = the "by variables" defining the strata
          bootvar   = the variable to be bootstrapped

OUTPUTS:  A uniquely named SAS dataset, the name of which contains the name of the
          algorithm, bsmp_size, and num_bsmps.  Variables in the output dataset include
          the mean of the bootstrap statistics, the standard deviation of the bootstrap
          statistics, and the 2.5th and 97.5th percentiles of the bootstrap statistics.
          Additional or different bootstrap statistics are easily added.

CAVEATS:  The "by variables" defining the strata in the input datasets are, in OPDY and
          DA, assumed to be character variables.

          The directory c:\MFBUS must be created before the program is run.

************************************************************************
************************************************************************
***;

options
label
symbolgen
fullstimer
yearcutoff=1950
nocenter
ls = 256
ps = 51
msymtabmax=max
mprint
mlogic
minoperator mindelimiter=' '
cleanup
;


libname MFBUS "c:\MFBUS";
```

```
%macro makedata(strata_size=, numsegs=, numgeogs=);

%let numstrata = %eval(&numsegs.*&numgeogs.);

*** For the price variable, multiplying the random variates by the loop counter
    dramatically skews the values of the sample space, thus ensuring that any
    erroneous non-random sampling will be spotted quickly and easily.
***;

data MFBUS.price_data_&numstrata.strata_&strata_size.(keep=geography segment price
sortedby=geography segment);
  format segment geography $8.;
  array seg{3} $ _TEMPORARY_ ('segment1' 'segment2' 'segment3');
  array geog{4} $ _TEMPORARY_ ('geog1' 'geog2' 'geog3' 'geog4');
  strata_size = 1* &strata_size.;
  do x=1 to &numgeogs.;
     geography=geog{x};
     do j=1 to &numsegs.;
        segment=seg{j};
        if j=1 then do i=1 to strata_size;
           price=rand('UNIFORM')*10*i;
           output;
        end;
        else if j=2 then do i=1 to strata_size;
           price=rand('NORMAL')*10*i;
           output;
        end;
        else if j=3 then do i=1 to strata_size;
           price=rand('LOGNORMAL')*10*i;
           output;
        end;
     end;
  end;
  run;

%mend makedata;


%makedata(strata_size=10000, numsegs=2, numgeogs=1);
%makedata(strata_size=10000, numsegs=2, numgeogs=3);
%makedata(strata_size=10000, numsegs=3, numgeogs=4);

%makedata(strata_size=100000, numsegs=2, numgeogs=1);
%makedata(strata_size=100000, numsegs=2, numgeogs=3);
%makedata(strata_size=100000, numsegs=3, numgeogs=4);

%makedata(strata_size=1000000, numsegs=2, numgeogs=1);
%makedata(strata_size=1000000, numsegs=2, numgeogs=3);
%makedata(strata_size=1000000, numsegs=3, numgeogs=4);

%makedata(strata_size=10000000, numsegs=2, numgeogs=1);
%makedata(strata_size=10000000, numsegs=2, numgeogs=3);
%makedata(strata_size=10000000, numsegs=3, numgeogs=4);
```

```
*** OPDY_Boot ***;
*** OPDY_Boot ***;
*** OPDY_Boot ***;

%macro OPDY_Boot(bsmp_size=, num_bsmps=, indata=, byvars=, bootvar=);


*** the only assumption made within this macro is that the byvars are all character
variables;


*** obtain last byvar, count byvars, and assign each byvar into macro variables for easy
access/processing;

%let last_byvar = %scan(&byvars.,-1);
%let num_byvars = %sysfunc(countw(&byvars.));
%do i=1 %to &num_byvars.;
  %let byvar&i. = %scan(&byvars.,&i.);
%end;


*** macro obtains number of observations in a dataset;

%macro nobs(dset);
   %if %sysfunc(exist(&dset)) %then %do;
      %let dsid=%sysfunc(open(&dset));
      %let nobs=%sysfunc(attrn(&dsid,nobs));
      %let dsid=%sysfunc(close(&dsid));
   %end;
   %else %let nobs=0;
   &nobs
%mend nobs;


*** initialize macro variables used later;
%let bmean =;
%let bstd =;
%let b975 =;
%let b025 =;


*** obtain counts and cumulated counts for each strata;

proc summary data=&indata. nway;
  class &byvars.;
  var &bootvar.;
  output out=byvar_nobs(keep=_FREQ_ &byvars.) n=junk;
  run;

%let n_byvals = %nobs(byvar_nobs);

data cum_temp(keep=_FREQ_ cum_prev_freq);
  set byvar_nobs(keep=_FREQ_);
  retain cum_prev_freq 0;
  prev_freq = lag(_FREQ_);
  if _n_=1 then prev_freq = 0;
  cum_prev_freq = sum(cum_prev_freq, prev_freq);
  run;


*** put counts, cumulated counts, and byvar values into macro strings;
```

```
proc sql noprint;
  select cum_prev_freq into :cum_prev_freqs separated by ' ' from cum_temp;
  quit;

proc sql noprint;
  select _freq_ into :freqs separated by ' ' from cum_temp;
  quit;

%do i=1 %to &num_byvars.;
   proc sql noprint;
     select &&byvar&i. into :byvals&i. separated by ' ' from byvar_nobs;
     quit;
%end;


*** get size of largest stratum;

proc summary data=byvar_nobs(keep=_FREQ_) nway;
  var _FREQ_;
  output out=byvar_nobs(keep=max_freq) max=max_freq;
  run;
data _null_;
  set byvar_nobs;
  call symputx('max_freq',max_freq);
  run;


*** save results of each stratum in cumulated macro variables instead of outputting to a
    dataset on the data step to lessen intermediate memory requirements
***;

data _null_;
  set &indata.(keep=&byvars. &bootvar.);
  by &byvars.;
  array bmeans{&num_bsmps.} bm1-bm&num_bsmps.;
  array temp{&max_freq.} _TEMPORARY_;
  retain byval_counter 0 cum_prev_freq 0;
  temp[_n_-cum_prev_freq]=&bootvar.;
  if last.&last_byvar. then do;
     byval_counter+1;
     freq = 1* scan("&freqs.", byval_counter,' ');
     num_bsmps = &num_bsmps.*1;
     bsmp_size = &bsmp_size.*1;
     do m=1 to num_bsmps;
        x=0;
        do n=1 to bsmp_size;
           x = temp[floor(ranuni(-1)*freq) + 1] + x;
        end;
        bmeans[m] = x/bsmp_size;
     end;
     bmean = mean(of bm1-bm&num_bsmps.);
     bstd = std(of bm1-bm&num_bsmps.);
     b975 = pctl(97.5, of bm1-bm&num_bsmps.);
     b025 = pctl(2.5, of bm1-bm&num_bsmps.);
     call symput('bmean',symget('bmean')||" "||compress(bmean));
     call symput('bstd',symget('bstd')||" "||compress(bstd));
     call symput('b975',symget('b975')||" "||compress(b975));
     call symput('b025',symget('b025')||" "||compress(b025));
     cum_prev_freq = 1*scan("&cum_prev_freqs.",byval_counter+1,' ');
  end;
```

```
  run;


*** obtain and assign the format of each byvar, all of which are assumed to be character
variables;

data lens(keep=lens);
  set &indata.(keep=&byvars. firstobs=1 obs=1);
  do i=1 to &num_byvars.;
     lens = vlengthx(scan("&byvars.",i));
     output;
  end;
  run;
proc sql noprint;
  select lens into :alllens separated by ' ' from lens;
  quit;
%macro assign_formats;
   %do i=1 %to &num_byvars.;
      &&byvar&i. $%scan(&alllens.,&i.).
   %end;
%mend assign_formats;


*** assign each byvar value for each stratum;

%macro assign_byvar_vals(which_strata=);
   %do j=1 %to &num_byvars.;
      &&byvar&j. = scan("&&byvals&j.",&which_strata.,' ');
   %end;
%mend assign_byvar_vals;


*** unwind and assign all the cumulated macro variables;

data MFBUS.OPDY_boots_&bsmp_size._&num_bsmps.(sortedby=&byvars. drop=n_byvals i);
  n_byvals = 1*&n_byvals.;
  format %assign_formats;
  do i=1 to n_byvals;
     bmean = 1*scan("&bmean.",i,' ');
     bstd = 1*scan("&bstd.",i,' ');
     b025 = 1*scan("&b025.",i,' ');
     b975 = 1*scan("&b975.",i,' ');
     %assign_byvar_vals(which_strata = i)
     output;
  end;
  run;


*** optional;
proc datasets lib=work memtype=data kill nodetails;
  run;

%mend OPDY_Boot;


%OPDY_Boot(bsmp_size=500,
           num_bsmps=500,
           indata=MFBUS.price_data_6strata_100000,
           byvars=geography segment,
           bootvar=price
          );
```

```
*** PSS ***;
*** PSS ***;
*** PSS ***;

%macro PSS(bsmp_size=, num_bsmps=, indata=, byvars=, bootvar=);

proc surveyselect data=&indata. method=urs sampsize=&bsmp_size. rep=&num_bsmps. seed=-1
out=Boot_PSS_Samps(drop=expectedhits samplingweight) noprint;
  strata &byvars.;
  run;

proc summary data=Boot_PSS_Samps nway;
  class &byvars. replicate;
  weight numberhits;
  var &bootvar.;
  output out=Boot_PSS_avgs(sortedby=&byvars. keep=&byvars. &bootvar.) mean=;
  run;

proc univariate data=Boot_PSS_avgs noprint;
  by &byvars.;
  var &bootvar.;
  output out=MFBUS.Boot_PSS_&bsmp_size._&num_bsmps.
        mean=bmean
         std=bstd
    pctlpts = 2.5 97.5
     pctlpre=b
  ;
  run;


*** optional;
proc datasets lib=work memtype=data kill nodetails;
  run;

%mend PSS;


%PSS(bsmp_size=500,
     num_bsmps=500,
     indata=MFBUS.price_data_6strata_100000,
     byvars=geography segment,
     bootvar=price
    );


*** Boot_DA ***;
*** Boot_DA ***;
*** Boot_DA ***;

%macro Boot_DA(bsmp_size=, num_bsmps=, indata=, byvars=, bootvar=);

*** the only assumption made within this macro is that the byvars are all character
variables;

*** obtain last byvar, count byvars, and assign each byvar into macro variables for easy
access/processing;

%let last_byvar = %scan(&byvars.,-1);
%let num_byvars = %sysfunc(countw(&byvars.));
%do i=1 %to &num_byvars.;
  %let byvar&i. = %scan(&byvars.,&i.);
```

```
%end;


*** macro obtains number of observations in a dataset;

%macro nobs(dset);
   %if %sysfunc(exist(&dset)) %then %do;
      %let dsid=%sysfunc(open(&dset));
      %let nobs=%sysfunc(attrn(&dsid,nobs));
      %let dsid=%sysfunc(close(&dsid));
   %end;
   %else %let nobs=0;
   &nobs
%mend nobs;


*** obtain counts and cumulated counts for each strata;

proc summary data=&indata. nway;
  class &byvars.;
  var &bootvar.;
  output out=byvar_nobs(keep=_FREQ_ &byvars.) n=junk;
  run;

%let n_byvals = %nobs(byvar_nobs);

data cum_temp(keep=_FREQ_ cum_prev_freq);
  set byvar_nobs(keep=_FREQ_);
  retain cum_prev_freq 0;
  prev_freq = lag(_FREQ_);
  if _n_=1 then prev_freq = 0;
  cum_prev_freq = sum(cum_prev_freq, prev_freq);
  run;


*** put counts, cumulated counts, and byvar values into macro strings;

proc sql noprint;
  select cum_prev_freq into :cum_prev_freqs separated by ' ' from cum_temp;
  quit;
proc sql noprint;
  select _freq_ into :freqs separated by ' ' from cum_temp;
  quit;

%do i=1 %to &num_byvars.;
   proc sql noprint;
     select &&byvar&i. into :byvals&i. separated by ' ' from byvar_nobs;
     quit;
%end;


*** obtain and assign the format of each byvar, all of which are assumed to be character
variables;

data lens(keep=lens);
  set &indata.(keep=&byvars. firstobs=1 obs=1);
  do i=1 to &num_byvars.;
     lens = vlengthx(scan("&byvars.",i));
     output;
  end;
  run;
```

```
proc sql noprint;
  select lens into :alllens separated by ' ' from lens;
  quit;
%macro assign_formats;
   %do i=1 %to &num_byvars.;
      &&byvar&i. $%scan(&alllens.,&i.).
   %end;
%mend assign_formats;


*** assign each byvar value for each stratum;

%macro assign_byvar_vals(which_strata=);
   %do j=1 %to &num_byvars.;
      &&byvar&j. = scan("&&byvals&j.",&which_strata.,' ');
   %end;
%mend assign_byvar_vals;


data MFBUS.boot_da_&bsmp_size._&num_bsmps.(keep=&byvars. bmean bstd b975 b025);
  n_byvals=&n_byvals.;
  bsmp_size = 1* &bsmp_size.;
  num_bsmps = 1* &num_bsmps.;
  format %assign_formats;
  do byval_counter=1 to n_byvals;
     freq          = 1* scan("&freqs.", byval_counter,' ');
     cum_prev_freq = 1* scan("&cum_prev_freqs.", byval_counter,' ');
     %assign_byvar_vals(which_strata = byval_counter)
     array bmeans{&num_bsmps.} bm1-bm&num_bsmps. (&num_bsmps.*0);
     do bsample=1 to num_bsmps;
        xsum=0;
        do obs=1 to bsmp_size;
           obsnum = floor(freq*ranuni(-1))+1+cum_prev_freq;
           set &indata.(keep=&bootvar.) point=obsnum;
           xsum = xsum + &bootvar.;
        end;
        bmeans[bsample] = xsum/bsmp_size;
     end;
     bmean = mean(of bm1-bm&num_bsmps.);
     bstd = std(of bm1-bm&num_bsmps.);
     b025 = pctl(2.5, of bm1-bm&num_bsmps.);
     b975 = pctl(97.5, of bm1-bm&num_bsmps.);
     output;
  end;
  stop;
  run;


*** optional;
proc datasets lib=work memtype=data kill nodetails;
  run;

%mend Boot_DA;


%Boot_DA(bsmp_size=500,
         num_bsmps=500,
         indata=MFBUS.price_data_6strata_100000,
         byvars=geography segment,
         bootvar=price
        );
```

```
*** Boot_SM ***;
*** Boot_SM ***;
*** Boot_SM ***;

%macro Boot_SM(bsmp_size=, num_bsmps=, indata=, byvars=, bootvar=);

*** obtain counts by strata;

proc summary data=&indata. nway;
  class &byvars.;
  var &bootvar.;
  output out=byvar_nobs(keep=_FREQ_ &byvars.) n=junk;
  run;

*** output bootstrap observations to sample in a nested loop;

data bsmp;
  set byvar_nobs(keep=&byvars. _FREQ_);
  num_bsmps = 1*&num_bsmps.;
  bsmp_size = 1*&bsmp_size.;
  do sample=1 to num_bsmps;
     do k=1 to bsmp_size;
        obsnum = floor(_FREQ_*ranuni(-1))+1;
        output;
     end;
  end;
  run;

proc sort data=bsmp;
  by &byvars. obsnum;
  run;

proc sort data=&indata. out=price_data;
  by &byvars.;
  run;


*** create record counter on input dataset;

%let last_byvar = %scan(&byvars.,-1);

data boot;
  set price_data;
  retain obsnum 0;
  by &byvars.;
  if first.&last_byvar. then obsnum=1;
  else obsnum+1;
  run;

proc sort data=boot;
  by &byvars. obsnum;
  run;


*** merge bootstrap sample observations with input dataset to obtain bootstrap samples;

data boot
     error
        ;
  merge boot(in=inboot)
        bsmp(in=inbsmp)
```

```
        ;
  by &byvars. obsnum;
  if inboot & inbsmp then output boot;
  else if inboot~=1 then output error;
  run;


*** summarize bootstrap samples;

proc summary data=boot(keep=&byvars. sample &bootvar.) nway;
  class &byvars. sample;
  var &bootvar.;
  output out=boot_means(keep=&byvars. &bootvar. sortedby=&byvars.) mean=;
  run;

proc univariate data=boot_means(keep=&byvars. &bootvar.) noprint;
  by &byvars.;
  var &bootvar.;
  output out=MFBUS.boot_Out_SM_&bsmp_size._&num_bsmps.
                                  mean = b_mean
                                   std = b_std
                                   pctlpts = 2.5 97.5 pctlpre=b
                                   ;
      run;

*** optional;
proc datasets lib=work memtype=data kill nodetails;
  run;

%mend Boot_SM;

%Boot_SM(bsmp_size=500,
         num_bsmps=500,
         indata=MFBUS.price_data_6strata_100000,
         byvars=geography segment,
         bootvar=price
        );


*** ALGO4p8 ***;
*** ALGO4p8 ***;
*** ALGO4p8 ***;

%macro Algo4p8(bsmp_size=, num_bsmps=, indata=, byvars=, bootvar=);

*** obtain counts by strata and merge onto input dataset;

proc summary data=&indata. nway;
  class &byvars.;
  var &bootvar.;
  output out=byvar_nobs(keep=_FREQ_ &byvars.) n=junk;
  run;

proc sort data=&indata. out=price_data;
  by &byvars.;
  run;

data price_data
     error
        ;
  merge byvar_nobs(in=innobs keep=&byvars. _FREQ_)
```

```
        &indata.(in=infull)
              ;
  by &byvars.;
  if innobs & infull then output price_data;
  else output error;
  run;


*** simultaneously create all bootstrap samples and summarize results of each as the end
of stratum is reached;

%let last_byvar = %scan(&byvars.,-1);

data MFBUS.boot_algo4p8_&bsmp_size._&num_bsmps.(keep=&byvars. bstd bmean b025 b975);
  set price_data end=lastrec;
  by &byvars.;
  num_bsmps = &num_bsmps.;
  array bsummeans{&num_bsmps.} bsummean_1-bsummean_&num_bsmps.;
  array bcds{&num_bsmps.} bcd_1-bcd_&num_bsmps.;
  retain bsummean_1-bsummean_&num_bsmps. 0 bcd_1-bcd_&num_bsmps. &bsmp_size. counter 0;
  counter+1;
  p = 1/(_FREQ_-counter+1);
  do i=1 to num_bsmps;
     if bcds[i]>0 then do;
        x = rand('BINOMIAL',p,bcds[i]);
        bsummeans[i]=x*&bootvar. + bsummeans[i];
        bcds[i]=bcds[i]-x;
     end;
  end;
  if last.&last_byvar. then do;
     bsmp_size = 1*&bsmp_size.;
     do h=1 to num_bsmps;
        bsummeans[h] = bsummeans[h]/bsmp_size;
     end;
     bmean = mean(of bsummean_1-bsummean_&num_bsmps.);
     bstd = std(of bsummean_1-bsummean_&num_bsmps.);
     b025 = pctl(2.5, of bsummean_1-bsummean_&num_bsmps.);
     b975 = pctl(97.5, of bsummean_1-bsummean_&num_bsmps.);
     output;
     if lastrec~=1 then do;
        counter = 0;
        do x=1 to num_bsmps;
           bsummeans[x]=0;
           bcds[x]=bsmp_size;
        end;
     end;
  end;
  run;

*** optional;
proc datasets lib=work memtype=data kill nodetails;
  run;

%mend Algo4p8;

%Algo4p8(bsmp_size=500,
         num_bsmps=500,
         indata=MFBUS.price_data_6strata_100000,
         byvars=geography segment,
         bootvar=price
        );
```

Appendix B

Table B1:
Real and CPU Runtimes (minutes) of the Algorithms for Various *N*, #strata, *n*, and *m*

(ex = excessive, cr = crashed)

				REAL							CPU						
N ***(per stratum)***	***#*** ***strata***	***n***	***m***	**OPDY**	**PSS**	**A4.8**	**HTPS**	**HTIT**	**DA**	**Out-SM**	**OPDY**	**PSS**	**A4.8**	**HTPS**	**HTIT**	**DA**	**Out-SM**
10,000	2	500	500	0.01	0.06	0.12	0.10	0.05	0.10	0.09	0.00	0.03	0.12	0.04	0.02	0.04	0.07
100,000	2	500	500	0.01	0.33	1.19	0.09	0.07	0.11	0.14	0.01	0.26	1.17	0.10	0.04	0.05	0.06
1,000,000	2	500	500	0.10	3.00	11.70	0.24	0.19	0.14	0.41	0.07	2.41	11.51	0.12	0.11	0.07	0.17
10,000,000	2	500	500	0.71	29.96	117.14	1.80	1.93	0.47	5.37	0.60	23.75	115.03	0.97	0.97	0.30	1.28
10,000	6	500	500	0.02	0.18	0.35	0.21	0.14	0.29	0.30	0.01	0.12	0.35	0.09	0.08	0.12	0.16
100,000	6	500	500	0.04	0.99	3.50	0.20	0.18	0.31	0.36	0.03	0.79	3.44	0.12	0.10	0.14	0.18
1,000,000	6	500	500	0.24	8.95	34.83	0.56	0.49	1.07	1.08	0.19	7.30	34.38	0.35	0.34	0.20	0.53
10,000,000	6	500	500	2.24	89.17	830.52	cr	cr	17.51	19.78	1.83	72.76	781.98	cr	cr	0.97	4.00
10,000	12	500	500	0.03	0.36	0.75	0.57	0.89	0.57	0.53	0.02	0.27	0.73	0.18	0.15	0.25	0.31
100,000	12	500	500	0.08	2.39	7.35	0.72	0.68	0.66	0.65	0.06	1.88	7.19	0.25	0.20	0.26	0.38
1,000,000	12	500	500	0.46	21.78	74.56	1.78	1.55	4.67	1.96	0.38	17.67	72.27	0.80	0.75	0.44	1.04
10,000,000	12	500	500	4.67	210.19	ex	cr	cr	51.39	33.01	3.64	171.66	ex	cr	cr	1.91	8.28
10,000	2	1000	1000	0.03	0.18	0.24	0.24	0.19	0.37	0.41	0.01	0.12	0.23	0.12	0.09	0.18	0.24
100,000	2	1000	1000	0.03	0.75	2.36	0.25	0.22	0.42	0.47	0.02	0.58	2.29	0.14	0.11	0.19	0.25
1,000,000	2	1000	1000	0.13	6.39	23.51	0.42	0.32	1.13	0.56	0.11	5.10	22.98	0.24	0.21	0.21	0.34
10,000,000	2	1000	1000	0.96	62.69	1372.74	2.44	2.20	8.73	5.95	0.85	50.07	1170.95	1.15	1.10	0.46	1.53
10,000	6	1000	1000	0.07	0.56	0.73	1.25	1.23	1.16	1.83	0.03	0.36	0.71	0.37	0.30	0.52	0.73
100,000	6	1000	1000	0.08	2.30	7.14	1.18	1.11	1.27	2.07	0.07	1.75	7.03	0.43	0.36	0.51	0.77
1,000,000	6	1000	1000	0.33	18.80	71.00	1.72	1.59	1.39	2.81	0.29	15.23	69.97	0.74	0.67	0.58	1.06
10,000,000	6	1000	1000	2.93	186.14	ex	cr	cr	27.81	20.24	2.51	149.18	ex	cr	cr	1.43	4.53
10,000	12	1000	1000	0.09	1.04	1.46	1.81	1.67	2.25	3.50	0.06	0.74	1.42	0.78	0.60	0.99	1.52
100,000	12	1000	1000	0.30	4.51	14.26	1.82	1.80	2.58	3.65	0.13	3.52	14.08	0.88	0.67	1.07	1.60
1,000,000	12	1000	1000	0.67	40.72	142.47	2.92	3.21	6.77	5.85	0.58	33.64	140.50	1.53	1.37	1.22	2.20
10,000,000	12	1000	1000	6.47	408.29	ex	cr	cr	68.52	39.03	5.28	334.25	ex	cr	cr	3.04	9.32
10,000	2	2000	2000	0.05	0.57	0.47	1.44	1.37	1.53	2.62	0.04	0.40	0.46	0.50	0.38	0.66	1.19
100,000	2	2000	2000	0.12	1.68	4.62	1.47	1.75	1.71	2.46	0.07	1.28	4.52	0.53	0.41	0.69	1.24
1,000,000	2	2000	2000	0.27	12.37	45.58	1.44	1.52	1.77	3.16	0.19	9.91	45.12	0.70	0.58	0.70	1.30
10,000,000	2	2000	2000	1.38	119.97	ex	4.70	3.92	11.63	8.38	1.26	95.43	ex	1.75	1.57	0.91	2.48
10,000	6	2000	2000	0.14	1.92	1.43	4.21	2.96	4.55	13.23	0.12	1.22	1.40	1.57	1.15	1.91	3.72
100,000	6	2000	2000	0.22	5.25	13.98	4.27	3.73	5.08	11.07	0.20	3.94	13.78	1.62	1.29	1.95	3.87
1,000,000	6	2000	2000	0.60	37.76	138.84	4.46	3.97	5.32	19.38	0.54	30.11	135.16	2.17	1.79	1.99	4.24
10,000,000	6	2000	2000	4.26	365.80	ex	cr	cr	33.27	54.24	3.80	290.18	ex	cr	cr	2.80	7.80
10,000	12	2000	2000	0.27	3.59	2.79	7.45	6.67	9.10	23.49	0.24	2.51	2.75	3.18	2.35	3.85	7.72
100,000	12	2000	2000	0.43	11.40	27.36	7.46	6.87	10.93	23.87	0.40	7.97	26.96	3.53	2.66	4.37	8.18
1,000,000	12	2000	2000	1.33	82.82	276.65	9.35	8.80	10.76	28.00	1.13	66.17	273.12	4.75	3.89	4.00	8.30
10,000,000	12	2000	2000	15.84	794.49	ex	cr	cr	95.52	59.85	7.88	640.85	ex	cr	cr	5.62	15.30

Appendix C

To prove the validity of Algorithm 4.8 as an equal-probability, with-replacement sampling algorithm, it must be shown that all possible samples of *n* items have equal probability of being selected. The probability of drawing any specific item in any draw from among the *N* items is 1/*N*, so the probability of a particular set of *n* items being drawn, in a specific order, is simply $(1/N)^n$ or $1/N^n$.[7] However, the order of selection does not matter for these purposes,[8] so we must use the probability of drawing a particular sample of *n* items, in any order, and show that this is identical to the probability of drawing any sample of *n* items using Algo4.8.

The probability of drawing any particular sample of *n* items, in any order, when sampling with replacement is given by Efron and Tibshirani (1993, p. 58) as

$$\frac{n!}{b_1!b_2!\cdots b_n!}\cdot\prod_{i=1}^{n}\left(\frac{1}{n}\right)^{b_i} \tag{C1}$$

when $n = N$, and b_i indicates the number of times item *i* is drawn. Note that, by definition, $n=\sum_{i=1}^{n} b_i$, so $\prod_{i=1}^{n}(1/n)^{b_i} = (1/n)^n$. So when $n \le N$, (C1) is simply

$$\frac{n!}{b_1!b_2!\cdots b_N!}\cdot\left(\frac{1}{N}\right)^{n} \tag{C2}$$

To show that the probability that any sample of *n* items drawn from *N* items ($n \le N$) using Algo4.8 is equal to (C2), note that the probability that any of the *n* items in the sample is drawn *b* times, where *b* is 0 ≤ positive integers ≤ *n'*, is by definition the binomial probability (using *n'* and *p* as defined in Algo4.8)

$$\Pr\left(b_i = b_i^*\right) = \binom{n'}{b_i^*} p^{b_i^*}\left(1-p\right)^{n'-b_i^*} \quad \text{for } 0 < p < 1 \text{ [9]} \tag{C3}$$

This makes the probability of drawing any *n* items with Algo4.8, with possible duplicates, in any order, the following:

[7] This corresponds with there being N^n possible sample-orderings for a with-replacement sample of *n* items drawn from *N* items, $n \le N$.

[8] Note that given the sequential nature of the algorithm, the order of the *n* items will be determined by, and is the same as, the order of the *N* population items, which of course can be sorted in any way without affecting the validity of the Algo4.8 algorithm.

[9] In Algo4.8, as mentioned above, *p* will never equal zero, and if $p = 1$, $b_i^* = n'$, which is correct.

Pr(Algo4.8 sample = any particular with-replacement sample) =

$$\frac{n!}{b_1!(n-b_1)!}\left(\frac{1}{N}\right)^{b_1}\left(1-\frac{1}{N}\right)^{n-b_1}\cdot$$

$$\frac{(n-b_1)!}{b_2!(n-b_1-b_2)!}\left(\frac{1}{N-1}\right)^{b_2}\left(1-\frac{1}{N-1}\right)^{n-b_1-b_2}\cdot$$

$$\frac{(n-b_1-b_2)!}{b_3!(n-b_1-b_2-b_3)!}\left(\frac{1}{N-2}\right)^{b_3}\left(1-\frac{1}{N-2}\right)^{n-b_1-b_2-b_3}\cdot$$

$$\vdots$$

$$\frac{(n-b_1-b_2-\cdots-b_{N-1})!}{b_N!(n-b_1-b_2-b_3-\cdots-b_N)!}\left(\frac{1}{N-(N-1)}\right)^{b_N}\left(1-\frac{1}{N-(N-1)}\right)^{n-b_1-b_2-b_3-\cdots-b_N} \quad \text{(C4)}$$

Reordering terms gives

Pr(Algo4.8 sample = any particular with-replacement sample) =

$$\frac{n!}{b_1!(n-b_1)!}\cdot\frac{(n-b_1)!}{b_2!(n-b_1-b_2)!}\cdot\frac{(n-b_1-b_2)!}{b_3!(n-b_1-b_2-b_3)!}\cdot\cdots\cdot\frac{(n-b_1-b_2-\cdots-b_{N-1})!}{b_N!(n-b_1-b_2-b_3-\cdots-b_N)!}\cdot$$

$$\left(\frac{1}{N}\right)^{b_1}\left(\frac{N-1}{N}\right)^{n-b_1}\cdot\left(\frac{1}{N-1}\right)^{b_2}\left(\frac{N-2}{N-1}\right)^{n-b_1-b_2}\cdot\left(\frac{1}{N-2}\right)^{b_3}\left(\frac{N-3}{N-2}\right)^{n-b_1-b_2-b_3}\cdot\cdots\cdot$$

$$\left(\frac{1}{N-(N-1)}\right)^{b_N}\left(\frac{N-N}{N-(N-1)}\right)^{n-b_1-b_2-b_3-\cdots-b_N} \quad \text{(C5)}$$

Because $n=\sum_{i=1}^{n}b_i$, the denominator in the last "combinatoric" term in (C5) is $b_N!0!=b_N!$, and except for the first numerator *n*! and b_i! in each denominator, the rest of the numerators and denominators in the combinatoric terms cancel leaving $n!/(b_1!b_2!\cdots b_N!)$. Of the remaining "probability" terms, the final term can be written as $\left(\frac{1}{N-(N-1)}\right)^{b_N}\left(\frac{1}{N-(N-1)}\right)^{n-b_1-b_2-b_3-\cdots-b_N}\left(\frac{0}{1}\right)^{0}$. If we avoid the centuries old debate (at least since the time of Euler) regarding the value of 0^0 and define $0^0 = 1$, as is accepted convention by numerous

august mathematicians (including Euler),[10] then all the (N - x) numerators and denominators cancel here as well, leaving only, from the first term, $(1/N)^{b_1}(1/N)^{n-b_1} = (1/N)^n$, which yields

$$\frac{n!}{b_1!b_2!\cdots b_N!}\cdot\left(\frac{1}{N}\right)^n$$, which is (C2).

Examples of calculating this probability using both (C2) directly and the steps of Algo4.8, for both $n \le N$ and $n = N$, are shown in Table C1 below.

Table C1:
Algo4.8 vs. (2): Probability of Drawing Any Particular With-Replacement Sample of n items, $n \le N$

	Algo4.8										**(C2)**			
for $n=N$										Cum.		Cum.		Cum.
Variable Value	Record #	N	n	N'	n'	p'	b^*	Cum. Sample	Pr(b=b^*)	Product Pr.	$(1/n)^{\wedge}b^*$	Product $(1/n)^{\wedge}b^*$	$b^*!$	Product $b^*!$
a	1	3	3	3	3	0.33	2	a,a	0.222	0.222	0.111	0.111	2	2
b	2	3	3	2	1	0.50	0	a,a	0.500	0.111	1.000	0.111	1	2
c	3	3	3	1	1	1.00	1	a,a,c	1.000	0.111	0.333	0.037	1	2
End Result								a,a,c	**Pr(any 3) = Pr(a,a,c) =**	**0.111**			**Pr(any 3) = (2) =**	**0.111**

	Algo4.8										**(C2)**			
for $n<N$										Cum.		Cum.		Cum.
Variable Value	Record #	N	n	N'	n'	p'	b^*	Cum. Sample	Pr(b=b^*)	Product Pr.	$(1/N)^{\wedge}b^*$	Product $(1/N)^{\wedge}b^*$	$b^*!$	Product $b^*!$
a	1	5	3	5	3	0.20	2	a,a	0.096	0.096	0.040	0.040	2	2
b	2	5	3	4	1	0.25	0	a,a	0.750	0.072	1.000	0.040	1	2
c	3	5	3	3	1	0.33	0	a,a	0.667	0.048	1.000	0.040	1	2
d	4	5	3	2	1	0.50	1	a,a,d	0.500	0.024	0.200	0.008	1	2
e	5	5	3	1	0			a,a,d			1.000	0.008	1	2
End Result								a,a,d	**Pr(any 3) = Pr(a,a,d) =**	**0.024**			**Pr(any 3) = (2) =**	**0.024**

[10] See Euler (1748), Euler (1770), Graham et al., (1994), Knuth (1992), and Vaughan (1970).

References

Bebbington, A. (1975), "A Simple Method of Drawing a Sample Without Replacement," *Journal of the Royal Statistical Society, Series C (Applied Statistics)*, Vol. 24, No. 1, 136.

Bentley, J. L. and Floyd, R. (1987), "A Sample of Brilliance," *Communications of the Association for Computing Machinery*, 30, 754–757.

CherNick, M. (2007), *Bootstrap Methods: A Guide for Practitioners and Researchers*, 2nd ed., Hoboken, NJ, John Wiley & Sons, Inc.

Cassell, D. (2007), "Don't Be Loopy: Re-Sampling and Simulation the SAS® Way," Proceedings of the SAS Global Forum 2007 Conference, Cary, NC: SAS Institute Inc.

Davison, A., and Hinkley, D. (1997), *Bootstrap Methods and their Application*, Cambridge, UK, Cambridge University Press.

Efron, B. (1979), "Bootstrap Methods: Another Look at the Jackknife," *Annals of Statistics*, 7, 1–26.

Efron, B., and Tibshirani, R. (1993), *An Introduction to the Bootstrap*, New York, Chapman & Hall, LLC.

Euler, L. (1748), translated by J.D. Blanton (1988), *Introduction to Analysis of the Infinite*, New York, NY, Springer-Verlag.

Euler, L. (1770), translated by Rev. John Hewlett (1984), *Elements of Algebra*, New York, NY, Springer-Verlag.

Knuth, D. (May, 1992), "Two Notes on Notation," *The American Mathematical Monthly*, Vol. 99, No. 5.

Graham, R., Knuth, D., and Patashnik, O. (1994), *Concrete Mathematics: A Foundation for Computer Science*, 2nd ed., Reading, MA: Addison-Wesley Publishing Company.

Secosky, J., and Bloom, J. (May, 2007), "Getting Started with the DATA Step Hash Object," SAS Institute, Inc.

Secosky, J., and Bloom, J. (2008), "Better Hashing in SAS® 9.2," SAS Institute, Inc., Paper 306-2008.

SAS Institute Inc. (2007), *SAS OnlineDoc® 9.2*, Cary, NC: SAS Institute Inc.

Tillé, Y. (2006), *Sampling Algorithms*, New York, NY, Springer.

Tillé, Y. (2010), *eMail Correspondence*, September 30, 2010.

Vaughan, H. (February 1970), "The Expression of 0^0," *The Mathematics Teacher*, Vol. 63, p.111.