# Bootstraps, Permutation Tests, and Sampling Orders of Magnitude Faster Using SAS®

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

## Abstract

While permutation tests and bootstraps have very wide-ranging application, both share a common potential drawback: as data-intensive resampling methods, both can be runtime prohibitive when applied to large or even medium-sized data samples drawn from large datasets. The data explosion over the past few decades has made this a common occurrence, and it highlights the increasing need for faster, and more efficient and scalable, permutation test and bootstrap algorithms.

Seven bootstrap and six permutation test algorithms coded in SAS (the largest privately owned software firm globally) are compared. The fastest algorithms ("OPDY" for the bootstrap, "OPDN" for permutation tests) are new, use no modules beyond Base SAS, and achieve speed increases orders of magnitude faster than the relevant "built-in" SAS procedures (OPDY is over 200x faster than Proc SurveySelect; OPDN is over 240x faster than Proc SurveySelect, over 350x faster than NPAR1WAY (which crashes on datasets less than a tenth the size OPDN can handle), and over 720x faster than Proc Multtest). OPDY also is much faster than hashing, which crashes on datasets smaller – sometimes by orders of magnitude – than OPDY can handle. OPDY is easily generalizable to multivariate regression models, and OPDN, which uses an extremely efficient draw-by-draw random-sampling-without-replacement algorithm, can use virtually any permutation statistic, so both have a very wide range of application. And the time complexity of both OPDY and OPDN is sub-linear, making them not only the fastest, but also the only truly scalable bootstrap and permutation test algorithms, respectively, in SAS.

Keywords:
Bootstrap, Permutation, SAS, Scalable, Hashing, With Replacement, Without Replacement, Sampling

JEL Classifications: C12, C13, C14, C15, C63, C88
Mathematics Subject Classifications: 62F40, 62G09, 62G10

* J.D. Opdyke is Managing Director, DataMineit, LLC, a consultancy specializing in advanced statistical and econometric modeling, risk analytics, and algorithm development for the banking, finance, and consulting sectors. J.D. has been a SAS user for over 20 years and routinely writes SAS code faster (often orders of magnitude faster) than SAS Procs (including but not limited to Proc Logistic, Proc MultTest, Proc Summary, Proc NPAR1WAY, Proc Freq, Proc Plan, and Proc SurveySelect). He earned his undergraduate degree with honors from Yale University, his graduate degree from Harvard University where he was both a Kennedy Fellow and a Social Policy Research Fellow, and he has completed post-graduate work as an ASP Fellow in the graduate mathematics department at MIT. His peer reviewed publications span number theory/combinatorics, robust statistics and high-convexity VaR modeling for regulatory and economic capital estimation, statistical finance, statistical computation, applied econometrics, and hypothesis testing for statistical quality control. Most are available upon request from J.D. at jdopdyke@datamineit.com.



## Acknowledgments

I sincerely thank Toyo Johnson and Nicole Ann Johnson Opdyke for their support and belief that SAS could produce better bootstraps, better permutation tests, and better sampling with and without replacement.

**Appendix A**

SAS v.9.2 code for the OPDN, OPDN-Alt, PROCSS, PROCMT, PROCNPAR, and Bebb-Sim algorithms, and the SAS code that generates the datasets used to test them in this paper, is presented below.

```
**************************************************************************
**************************************************************************
PROGRAM:  MFPTUS.SAS

DATE:  12/30/10

CODER:       J.D. Opdyke

PURPOSE:  Run and compare 6 different SAS Permutation Tests algorithms
          including OPDN, OPDN_alt, PROCSS, PROCMT, PROCNPAR, and BEBB_SIM.
             See Opdyke J.D., "Permutation Tests (and Sampling with Replacement) Orders of
             Magnitude Faster Using SAS," InterStat, January, 2011, for detailed
explanations
             of the different algorithms.

INPUTS:      Each macro is completely modular and accepts 6 macro parameter
          values (PROCMT accepts 7):
                indata        = the input dataset (including libname)
                outdata       = the input dataset (including libname)
                byvars        = the "by variables" defining the strata (these are
                                character variables)
                samp2var      = the (character) variable defining the two samples in
each
                                stratum: TEST ("T") and CONTROL ("C"), with those
values
                permvar       = the variable to be tested with permutation tests
                num_psmps     = number of Permutation Test samples
                seed          = and optional random number seed (must be an integer or
                                blank)
             left_or_right = "left" or "right" depending on whether the user wants
                                lower or upper one-tailed p-values, respectively, in
                                addition to the two-tailed p-value (only for PROCMT)

OUTPUTS:  A SAS dataset, named by the user via the macro variable outdata,
          which contains the following variables:
                1) the "by variables" defining the strata
                2) the name of the permuted variable
                3) the size (# of records) of the permutation samples (this should
                   always be the smaller of the two samples (TEST v. CONTROL))
                4) the number of permutation samples
                5) the left, right, and two-tailed p-values corresponding to the
                   following hypotheses:
                      p_left  = Pr(x>X | Ho: Test>=Control)
                      p_right = Pr(x<X | Ho: Test<=Control)
                      p_both  = Pr(x=X | Ho: Test=Control)
                   where x is the sample permutation statistic from the Control
                   sample (or the additive inverse of the Test sample if the Test
                   sample is smaller, which is atypical) and X is the distribution of
                   permutation statistic values.

                   Following standard convention, the two-tailed p-value is
                   calculated based on the reflection method.
**************************************************************************
**************************************************************************
```

```
***;


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


libname MFPTUS "c:\";

%MACRO makedata(strata_size=, testproportion=, numsegs=, numgeogs=);

*** Since generating all the datasets below once takes only a couple of minutes,
    time was not wasted trying to optimize runtimes for the creation of mere
       test data.
***;

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

*** The variable "cntrl_test" identifies the test and control samples with
    values of "T" and "C" respectively.
***;

data MFPTUS.pricing_data_&numstrata.strata_&strata_size.(keep=geography segment cntrl_test
price sortedby=geography segment);
  format segment geography $8. cntrl_test $1.;
  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;
              if mod(i,&testproportion.)=0 then do;
                 cntrl_test="T";
                 price=(rand('UNIFORM')+0.175/(&testproportion./(&testproportion./10)));
              end;
              else do;
                 cntrl_test="C";
                 price=rand('UNIFORM');
              end;
              output;
           end;
           else if j=2 then do i=1 to strata_size;
              if mod(i,&testproportion.)=0 then do;
                 cntrl_test="T";
                 price=rand('POISSON',1-0.75/(&testproportion./(&testproportion./10)));
              end;
              else do;
                 cntrl_test="C";
                 price=rand('POISSON',1.0);
              end;
              output;
           end;
```

```
          else if j=3 then do i=1 to strata_size;
*** Make Control smaller to test code in algorithms.;
             if mod(i,&testproportion.)=0 then cntrl_test="C";
             else cntrl_test="T";
             price=rand('NORMAL');
             output;
          end;
     end;
  end;
  run;
%MEND makedata;

%makedata(strata_size=10000, testproportion=10, numsegs=2, numgeogs=1);
%makedata(strata_size=10000, testproportion=10, numsegs=2, numgeogs=3);
%makedata(strata_size=10000, testproportion=10, numsegs=3, numgeogs=4);

%makedata(strata_size=100000, testproportion=100, numsegs=2, numgeogs=1);
%makedata(strata_size=100000, testproportion=100, numsegs=2, numgeogs=3);
%makedata(strata_size=100000, testproportion=100, numsegs=3, numgeogs=4);

%makedata(strata_size=1000000, testproportion=1000, numsegs=2, numgeogs=1);
%makedata(strata_size=1000000, testproportion=1000, numsegs=2, numgeogs=3);
%makedata(strata_size=1000000, testproportion=1000, numsegs=3, numgeogs=4);

%makedata(strata_size=10000000, testproportion=10000, numsegs=2, numgeogs=1);
%makedata(strata_size=10000000, testproportion=10000, numsegs=2, numgeogs=3);
%makedata(strata_size=10000000, testproportion=10000, numsegs=3, numgeogs=4);


%makedata(strata_size=7500000, testproportion=7500, numsegs=2, numgeogs=1);
%makedata(strata_size=7500000, testproportion=7500, numsegs=2, numgeogs=3);


%makedata(strata_size=25000000, testproportion=25000, numsegs=2, numgeogs=1);
%makedata(strata_size=50000000, testproportion=50000, numsegs=2, numgeogs=1);
%makedata(strata_size=100000000, testproportion=100000, numsegs=2, numgeogs=1);
```

***** OPDN ***;**
***** OPDN ***;**
***** OPDN ***;**

```
%MACRO OPDN(num_psmps=,
            indata=,
            outdata=,
            byvars=,
            samp2var=,
            permvar=,
            seed=
            );

*** The only assumption made within this macro is that the byvars are all
    character variables and the input dataset is sorted by the "by variables"
       that define the strata.  Also, the "TEST" and "CONTROL" samples are defined
    by a variable "cntrl_test" formatted as $1. containing "T" and "C"
       character strings, respectively, as values.
***;
```

```
*** Obtain the last byvar, count the byvars, and assign each byvar into numbered
    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;

*** If user does not pass a value to the optional macro variable "seed," use -1
    based on the time of day.
***;
%if %sysevalf(%superq(seed)=,boolean) %then %let seed=-1;

*** Obtain counts and cumulated counts for each strata.;

proc summary data=&indata. nway;
  class &byvars. &samp2var.;
  var &permvar.;
  output out=byvar_sum(keep = _FREQ_ &byvars. &samp2var. sumpvar)
                      sum = sumpvar
                     ;
  run;

*** Identify and keep the smaller of the two samples for more efficient
    sampling.  Below, invert the empirical permutation distribution if CONTROL
       sample is smaller that TEST sample (which is not typical).  That will
       remain consistent with output variables corresponding to:
            p_left  = Pr(x>X | Ho: Test>=Control)
            p_right = Pr(x<X | Ho: Test<=Control)
            p_both  = Pr(x=X | Ho: Test=Control)
       where x is the sample permutation statistic and X is the distribution of
       permutation statistic values.
***;

data byvar_sum_min(keep=tot_FREQ _FREQ_ &byvars. &samp2var. sumpvar sortedby=&byvars.);
  set byvar_sum;
  format tot_FREQ _FREQ_ 16.;
  by &byvars.;
  retain lag_FREQ lag_sum lag_samp2;
  if first.&last_byvar. then do;
        lag_FREQ = _FREQ_;
        lag_sum  = sumpvar;
        lag_samp2  = &samp2var.;
  end;
  else do;
     tot_FREQ = sum(lag_FREQ,_FREQ_);
     if _FREQ_<=lag_FREQ then output;
        else do;
           _FREQ_  = lag_FREQ;
           sumpvar = lag_sum;
           &samp2var. = lag_samp2;
           output;
        end;
  end;
  run;


*** Obtain number of strata, and use this number to count and separately permute
    each strata below.
```

```
***;

%let dsid = %sysfunc(open(byvar_sum_min));
%let n_byvals = %sysfunc(ifc(&dsid.
                                   ,%nrstr(%sysfunc(attrn(&dsid.,nobs))
                                           %let rc = %sysfunc(close(&dsid.));
                                              )
                                   ,%nrstr(0)
                                )
                        );
*** In case number is large, avoid scientific notation.;
%let n_byvals = %sysfunc(putn(&n_byvals.,16.));


*** Cumulate counts of strata for efficient (re)use of _TEMPORARY_ array.;

data byvar_sum_min(drop=prev_freq);
  set byvar_sum_min;
  format cum_prev_freq _FREQ_ 16.;
  retain cum_prev_freq 0;
  prev_freq = lag(tot_FREQ);
  if _n_=1 then prev_freq = 0;
  cum_prev_freq = sum(cum_prev_freq, prev_freq);
  run;


*** For access in data step below (that uses data _null_ for memory
    conservation), put into macro strings a) smaller-of-two-sample counts,
       b) total counts for each strata, c) cumulated total counts, d) summed
       permutation variable, d) which of two samples is smaller, and e) byvar
       values.
***;

proc sql noprint;
  select _freq_ into :freqs separated by ' ' from byvar_sum_min;
  quit;
proc sql noprint;
  select tot_FREQ into :tot_FREQs separated by ' ' from byvar_sum_min;
  quit;
proc sql noprint;
  select cum_prev_freq into :cum_prev_freqs separated by ' ' from byvar_sum_min;
  quit;
proc sql noprint;
  select sumpvar into :sumpvar separated by ' ' from byvar_sum_min;
  quit;
proc sql noprint;
  select &samp2var. into :samp2var separated by ' ' from byvar_sum_min;
  quit;

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


*** Get size of largest stratum for efficient (re)use of _TEMPORARY_ array.;

proc sql noprint;
  select max(tot_FREQ) into :max_tot_freq separated by ' ' from byvar_sum_min;
  quit;
```

```
*** In case number is large, avoid scientific notation.;
%let max_tot_freq = %sysfunc(putn(&max_tot_freq.,16.));


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

*** Initialize macro variables used below.;
%let p_left =;
%let p_right =;
%let p_both =;
%let samp_small_size =;

data _null_;
  set &indata.;
  by &byvars.;
  format which_sample $1.;
*** To view permutation distributions for each strata in .log file.,
    comment out first line below this comment, uncomment the two lines below
       it, and uncomment "put _ALL_" 37 lines below it.  Note that this will slow
       program execution and often create large .log files.
***;

  array temp{&max_tot_freq.} _TEMPORARY_;
%if &sysver >= 9.2 %then %do;
  array psums{&num_psmps.} _TEMPORARY_;
  retain byval_counter 0 cum_prev_freq 0;
%end;
%if &sysver < 9.2 %then %do;
  array psums{&num_psmps.} psmp1-psmp&num_psmps.;
  retain byval_counter 0 cum_prev_freq 0 psmp1-psmp&num_psmps.;
%end;

  temp[_n_-cum_prev_freq]=&permvar.;

  if last.&last_byvar. then do;
     byval_counter+1;
     num_psmps = &num_psmps.*1;
     psmp_size = 1 * scan("&freqs.", byval_counter,' ');
        which_sample = COMPRESS(UPCASE(scan("&samp2var.", byval_counter,' ')),' ');
     tot_FREQ_hold = 1 * scan("&tot_FREQs.", byval_counter,' ');
        seed = 1*&seed.;

     do m=1 to num_psmps;
        x=0;
        tot_FREQ = tot_FREQ_hold;
        do n=1 to psmp_size;
              cell = floor(ranuni(seed)*tot_FREQ) + 1;
              x = temp[cell] + x;
              hold = temp[cell];
              temp[cell]=temp[tot_FREQ];
              temp[tot_FREQ] = hold;
              tot_FREQ+(-1);
           end;
           psums[m] = x;
     end;
     psum = 1*scan("&sumpvar.", byval_counter,' ');
     p_right = 0;
```

```
        p_left  = 0;
        p_both  = 0;
        call sortn(of psums[*]);
     pmed = median(of psums[*]);
     pmean = mean(of psums[*]);

* put _ALL_;

*** Efficiently handle extreme test sample values.;

     IF psum<psums[1] THEN DO;
           p_left=0;
           p_right=num_psmps;
           p_both=0;
        END;
        ELSE IF psum>psums[num_psmps] THEN DO;
           p_left=num_psmps;
           p_right=0;
           p_both=0;
        END;
        ELSE DO;

*** For non-extreme cases, start with shorter tail for less looping.;

     if pmed>=psum then do;
           do z=1 to num_psmps;
              if psum>=psums[z] then p_left+1;
              else do;
                 lastbinnum = z-1;
                 distance_left = pmean - psums[z-1];
                 leave;
              end;
           end;

*** Avoid loop for other (larger) p-value.
    If psum equals last bin, p_right = 1 - p_left + lastbinsize.
    Otherwise, p_right = 1 - p_left.
***;
           if psum = psums[lastbinnum] then do;
              lastbinsize=1;
              do k=lastbinnum to 1 by -1;
                 if psums[k]=psums[k-1] then lastbinsize+1;
                 leave;
              end;
              p_right = num_psmps - p_left + lastbinsize;
           end;
           else p_right = num_psmps - p_left;
        end;

        else do;
           do z=num_psmps to 1 by -1;
              if psum<=psums[z] then p_right+1;
              else do;
                 lastbinnum = z+1;
                 distance_right = psums[z+1] - pmean;
                 leave;
              end;
           end;

*** Avoid loop for other (larger) p-value.
    If psum equals last bin, p_left = 1 - p_right + lastbinsize.
```

```
    Otherwise, p_left = 1 - p_right.
***;
            if psum = psums[lastbinnum] then do;
               lastbinsize=1;
               do k=lastbinnum to num_psmps;
                  if psums[k]=psums[k+1] then lastbinsize+1;
                  else leave;
               end;
               p_left = num_psmps - p_right + lastbinsize;
            end;
            else p_left = num_psmps - p_right;
         end;

*** Base 2-sided p-value on distance from mean of last (i.e. least extreme) bin
    of smaller p-value.  This is common practice.
***;

         if p_left<p_right then do;
            p_both = p_left;
         do z=num_psmps to 1 by -1;
               if (psums[z] - pmean) >= distance_left then p_both+1;
               else leave;
            end;
         end;
         else if p_left>p_right then do;
            p_both = p_right;
         do z=1 to num_psmps;
               if (pmean - psums[z]) >= distance_right then p_both+1;
               else leave;
            end;
         end;
         else p_both=num_psmps;

*** Account for possibility, due to psum=a particular bin value, that
    p_both>num_psmps.
***;
         p_both = min(p_both,num_psmps);

        END;

        p_left  = p_left  / num_psmps;
        p_right = p_right / num_psmps;
        p_both  = p_both  / num_psmps;

*** If CONTROL sample is smaller than TEST (which is atypical), reverse
*** p-values, as empirical distribution is mirror of itself.;

     if "C"=COMPRESS(UPCASE(scan("&samp2var.", byval_counter,' ')),' ') then do;
           hold = p_left;
           p_left = p_right;
           p_right = hold;
        end;

*** Cumulate key macro variables to save results.;

     call symput('p_left',symget('p_left')||" "||compress(p_left));
     call symput('p_right',symget('p_right')||" "||compress(p_right));
     call symput('p_both',symget('p_both')||" "||compress(p_both));
        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 &outdata.(sortedby=&byvars. drop=n_byvals i);
  n_byvals = 1*&n_byvals.;
  format %assign_formats;
  do i=1 to n_byvals;
     length permvar $32;
     permvar = "&permvar.";
     n_psamp = 1 * scan("&freqs.", i,' ');
     num_psmps = &num_psmps.;
     p_left = 1*scan("&p_left.",i,' ');
     p_right = 1*scan("&p_right.",i,' ');
     p_both = 1*scan("&p_both.",i,' ');
        %assign_byvar_vals(which_strata = i)
     label permvar   = "Permuted Variable"
           n_psamp   = "Size of Permutation Samples"
           num_psmps = "# of Permutation Samples"
           p_left    = "Left p-value"
           p_right   = "Right p-value"
           p_both    = "Two-Tailed p-value"
           ;
     output;
  end;
  run;


*** Optional.;
* proc datasets lib=work memtype=data kill nodetails;
*   run;
```

```
%MEND OPDN;

%OPDN(num_psmps = 1000,
      indata    = MFPTUS.pricing_data_2strata_100000,
      outdata   = MFPTUS.OPDN_100000_2strata,
      byvars    = geography segment,
      samp2var  = cntrl_test,
      permvar   = price,
      seed      =
     );
```

```
*** OPDN_alt ***;
*** OPDN_alt ***;
*** OPDN_alt ***;

%MACRO OPDN_alt(num_psmps=,
                indata=,
                   outdata=,
                   byvars=,
                   samp2var=,
                   permvar=,
                   seed=
                   );

*** If user does not pass a value to the optional macro variable "seed," use -1
    based on the time of day.
***;
%if %sysevalf(%superq(seed)=,boolean) %then %let seed=-1;

*** To minimize intermediate memory requirements, initialize output data set
    with missing variable values.
***;

data &outdata.(sortedby=&byvars.);
   stop;
   set &indata(keep=&byvars.);
   length permvar $32 n_psamp num_psmps p_left p_right p_both
          distance_right distance_left lastbinnum lastbinsize
          pmed pmean tot_FREQ_incr x 8;
   call missing(of _all_);
   run;

*** Obtain counts and cumulated counts for each strata.;

proc summary data=&indata. nway;
  class &byvars. &samp2var.;
  var &permvar.;
  output out=byvar_sum(keep = _FREQ_ &byvars. &samp2var. sumpvar)
                        sum = sumpvar;
  run;

*** Identify and keep the smaller of the two samples for more efficient
    sampling.  Below, invert the empirical permutation distribution if CONTROL
       sample is smaller that TEST sample (which is not typical).  That will
       remain consistent with output variables corresponding to:
            p_left  = Pr(x>X | Ho: Test>=Control)
            p_right = Pr(x<X | Ho: Test<=Control)
            p_both  = Pr(x=X | Ho: Test=Control)
```

```
       where x is the sample permutation statistic and X is the distribution of
       permutation statistic values.
***;

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

data byvar_sum_min(keep=tot_FREQ _FREQ_ &byvars. &samp2var. sumpvar sortedby=&byvars.);
  set byvar_sum;
  format tot_FREQ _FREQ_ 16.;
  by &byvars.;
  retain lag_FREQ lag_sum lag_samp2;
  if first.&last_byvar. then do;
        lag_FREQ = _FREQ_;
        lag_sum  = sumpvar;
        lag_samp2  = &samp2var.;
  end;
  else do;
     tot_FREQ = sum(lag_FREQ,_FREQ_);
     if _FREQ_<=lag_FREQ then output;
        else do;
           _FREQ_  = lag_FREQ;
           sumpvar = lag_sum;
           &samp2var. = lag_samp2;
           output;
        end;
  end;
  run;


*** Get size of largest stratum for efficient (re)use of _TEMPORARY_ array.;

proc sql noprint;
   select max(tot_FREQ) into :max_tot_freq from byvar_sum_min;
   quit;

*** In case number is large, avoid scientific notation.;
%let max_tot_freq = %sysfunc(putn(&max_tot_freq.,16.));

data &outdata.;
   if 0 then modify &outdata.;

*** To view permutation distributions for each strata in .log file.,
    comment out first line below this comment, uncomment the line below it, and
    uncomment "put _ALL_" 35 lines below it.  Note that this will slow program
       execution and often create large .log files.
***;
  array temp{&max_tot_freq.} _TEMPORARY_;
%if &sysver >= 9.2 %then %do;
  array psums{&num_psmps.} _TEMPORARY_;
  retain num_psmps &num_psmps.;
%end;
%if &sysver < 9.2 %then %do;
  array psums{&num_psmps.} psmp1-psmp&num_psmps.;
  retain num_psmps &num_psmps. psmp1-psmp&num_psmps.;
%end;

   do _n_ = 1 by 1 until(last.&last_byvar.);
      merge &indata.(keep=&byvars. &permvar. &samp2var.)
               byvar_sum_min;
      by &byvars.;
      temp[_n_]=&permvar.;
```

```
   end;
   seed = 1*&seed.;
   do m=1 to num_psmps;
      x=0;
      tot_FREQ_incr = tot_FREQ;
      do n=1 to _FREQ_;
            cell = floor(ranuni(seed)*tot_FREQ_incr) + 1;
            x = temp[cell] + x;
            hold = temp[cell];
            temp[cell]=temp[tot_FREQ_incr];
            temp[tot_FREQ_incr] = hold;
            tot_FREQ_incr+(-1);
         end;
      psums[m] = x;
   end;

   n_psamp = _FREQ_;

   p_right = 0;
   p_left  = 0;
      p_both  = 0;
      call sortn(of psums[*]);
   pmed = median(of psums[*]);
   pmean = mean(of psums[*]);

* put _ALL_;

*** Efficiently handle extreme test sample values.;

   IF sumpvar<psums[1] THEN DO;
      p_left=0;
      p_right=num_psmps;
      p_both=0;
   END;
   ELSE IF sumpvar>psums[num_psmps] THEN DO;
      p_left=num_psmps;
      p_right=0;
      p_both=0;
   END;
   ELSE DO;

*** For non-extreme cases, start with shorter tail for less looping.;

   if pmed>=sumpvar then do;
      do z=1 to num_psmps;
         if sumpvar>=psums[z] then p_left+1;
         else do;
            lastbinnum = z-1;
            distance_left = pmean - psums[z-1];
            leave;
         end;
      end;

*** Avoid loop for other (larger) p-value.
    If sumpvar equals last bin, p_right = 1 - p_left + lastbinsize.
    Otherwise, p_right = 1 - p_left.
***;
      if sumpvar = psums[lastbinnum] then do;
         lastbinsize=1;
         do k=lastbinnum to 1 by -1;
            if psums[k]=psums[k-1] then lastbinsize+1;
```

```
            leave;
         end;
         p_right = num_psmps - p_left + lastbinsize;
      end;
      else p_right = num_psmps - p_left;
   end;
   else do;
      do z=num_psmps to 1 by -1;
         if sumpvar<=psums[z] then p_right+1;
         else do;
            lastbinnum = z+1;
            distance_right = psums[z+1] - pmean;
            leave;
         end;
      end;

*** Avoid loop for other (larger) p-value.
    If psum equals last bin, p_left = 1 - p_right + lastbinsize.
    Otherwise, p_left = 1 - p_right.
***;
      if sumpvar = psums[lastbinnum] then do;
         lastbinsize=1;
         do k=lastbinnum to num_psmps;
            if psums[k]=psums[k+1] then lastbinsize+1;
            else leave;
         end;
         p_left = num_psmps - p_right + lastbinsize;
      end;
      else p_left = num_psmps - p_right;
   end;

*** Base 2-sided p-value on distance from mean of last (i.e. least extreme) bin
    of smaller p-value.  This is common practice.
***;

   if p_left<p_right then do;
      p_both = p_left;
      do z=num_psmps to 1 by -1;
         if (psums[z] - pmean) >= distance_left then p_both+1;
         else leave;
      end;
   end;
   else if p_left>p_right then do;
      p_both = p_right;
      do z=1 to num_psmps;
         if (pmean - psums[z]) >= distance_right then p_both+1;
         else leave;
      end;
   end;
   else p_both=num_psmps;

*** Account for possibility, due to psum=a particular bin value, that p_both>num_psmps.
***;
   p_both = min(p_both,num_psmps);

   END;

*** If CONTROL sample is smaller than TEST (which is atypical), reverse
*** p-values, as empirical distribution is mirror of itself.;

   if &samp2var.="C" then do;
```

```
      hold = p_left;
         p_left = p_right;
         p_right = hold;
      end;

   p_left  = p_left  / num_psmps;
   p_right = p_right / num_psmps;
   p_both  = p_both  / num_psmps;

   length permvar $32;
   retain permvar "&permvar";
   output;
   run;

data &outdata.;
   set &outdata.(keep=&byvars. permvar num_psmps n_psamp p_left p_right p_both);
   label permvar   = "Permuted Variable"
         n_psamp   = "Size of Permutation Samples"
         num_psmps = "# of Permutation Samples"
         p_left    = "Left p-value"
         p_right   = "Right p-value"
         p_both    = "Two-Tailed p-value"
         ;
   run;


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

%MEND OPDN_alt;


%OPDN_alt(num_psmps = 1000,
          indata    = MFPTUS.pricing_data_2strata_100000,
          outdata   = MFPTUS.OPDN_alt_100000_2strata,
          byvars    = geography segment,
          samp2var  = cntrl_test,
          permvar   = price,
          seed      =
         );
```

***** PROC_SS ***;**
***** PROC_SS ***;**
***** PROC_SS ***;**

```
%MACRO PROCSS(num_psmps=,
              indata=,
              outdata=,
              byvars=,
              samp2var=,
              permvar=,
              seed=
             );


*** If user does not pass a value to the optional macro variable "seed," use -1
```

```
    based on the time of day.
***;
%if %sysevalf(%superq(seed)=,boolean) %then %let seed=-1;

*** Obtain counts and cumulated counts for each strata.;

proc summary data=&indata. nway;
  class &byvars. &samp2var.;
  var &permvar.;
  output out=byvar_sum(keep = _FREQ_ &byvars. &samp2var. sumpvar)
                        sum = sumpvar
                       ;
  run;

*** Identify and keep the smaller of the two samples for more efficient
    sampling.  Below, invert the empirical permutation distribution if CONTROL
       sample is smaller that TEST sample (which is not typical).  That will
       remain consistent with output variables corresponding to:
            p_left  = Pr(x>X | Ho: Test>=Control)
            p_right = Pr(x<X | Ho: Test<=Control)
            p_both  = Pr(x=X | Ho: Test=Control)
       where x is the sample permutation statistic and X is the distribution of
       permutation statistic values.
***;

%let last_byvar = %scan(&byvars.,-1);
data byvar_sum(keep=&byvars. _NSIZE_ sumpvar &samp2var. sortedby=&byvars.);
  set byvar_sum(rename=(_FREQ_=_NSIZE_));
  by &byvars.;
  retain lag_NSIZE lag_sum lag_samp2;
  if first.&last_byvar. then do;
     lag_NSIZE = _NSIZE_;
        lag_sum  = sumpvar;
        lag_samp2  = &samp2var.;
  end;
  else do;
     if _NSIZE_<=lag_NSIZE then output;
        else do;
              _NSIZE_  = lag_NSIZE;
              sumpvar = lag_sum;
              &samp2var. = lag_samp2;
              output;
        end;
  end;
  run;

*** From SAS Online Documentation:
    For simple random sampling without replacement, if there is enough memory
    for it PROC SURVEYSELECT uses Floyds ordered hash table algorithm (see
       Bentley and Floyd (1987) and Bentley and Knuth (1986) for details).  If
       there is not enough memory available for Floyds algorithm, PROC
       SURVEYSELECT switches to the sequential algorithm of Fan, Muller, and
       Rezucha (1962), which requires less memory but might require more time to
       select the sample.
***;

proc surveyselect data     = &indata.(drop=&samp2var.)
                  method   = srs
                     sampsize = byvar_sum(keep=&byvars. _NSIZE_)
                     rep      = &num_psmps.
                     seed     = &seed.
```

```
                     out      = PSS_perm_Samps(drop=SamplingWeight SelectionProb)
                     noprint;
  strata &byvars.;
  run;

proc summary data=PSS_perm_Samps nway;
  class &byvars. replicate;
  var &permvar.;
output out=PSS_perm_sums(sortedby=&byvars. replicate keep=&byvars. replicate &permvar.)
sum=;
  run;

proc transpose data=PSS_perm_sums out=PSS_perm_sums_t(rename=(_NAME_=permvar))
prefix=psmp;
  var &permvar.;
  by &byvars.;
  id replicate;
  run;

data &outdata.(keep=&byvars. permvar n_psamp num_psmps p_left p_right p_both)
     error
        ;
  merge PSS_perm_sums_t(in=insamps)
        byvar_sum(in=insummary)
           ;
  by &byvars.;
  if insamps & insummary then do;
     array psums[&num_psmps.] psmp1-psmp&num_psmps.;
     n_psamp = _NSIZE_;
     num_psmps = 1*&num_psmps.;
     p_left  = 0;
     p_right = 0;
     p_both  = 0;
     call sortn(of psums[*]);
     pmed = median(of psums[*]);
     pmean = mean(of psums[*]);

*** Efficiently handle extreme test sample values.;

     IF sumpvar<psums[1] THEN DO;
        p_left=0;
        p_right=num_psmps;
        p_both=0;
     END;
     ELSE IF sumpvar>psums[num_psmps] THEN DO;
        p_left=num_psmps;
        p_right=0;
        p_both=0;
     END;
     ELSE DO;

*** For non-extreme cases, start with shorter tail for less looping.;

        if pmed>=sumpvar then do;
           do z=1 to num_psmps;
              if sumpvar>=psums[z] then p_left+1;
              else do;
                 lastbinnum = z-1;
                 distance_left = pmean - psums[z-1];
                 leave;
              end;
```

```
         end;

*** Avoid loop for other (larger) p-value.
    If sumpvar equals last bin, p_right = 1 - p_left + lastbinsize.
    Otherwise, p_right = 1 - p_left.
***;
         if sumpvar = psums[lastbinnum] then do;
            lastbinsize=1;
            do k=lastbinnum to 1 by -1;
               if psums[k]=psums[k-1] then lastbinsize+1;
               leave;
            end;
            p_right = num_psmps - p_left + lastbinsize;
         end;
         else p_right = num_psmps - p_left;
      end;

      else do;
         do z=num_psmps to 1 by -1;
            if sumpvar<=psums[z] then p_right+1;
            else do;
               lastbinnum = z+1;
               distance_right = psums[z+1] - pmean;
               leave;
            end;
         end;

*** Avoid loop for other (larger) p-value.
    If psum equals last bin, p_left = 1 - p_right + lastbinsize.
    Otherwise, p_left = 1 - p_right.
***;
         if sumpvar = psums[lastbinnum] then do;
            lastbinsize=1;
            do k=lastbinnum to num_psmps;
               if psums[k]=psums[k+1] then lastbinsize+1;
               else leave;
            end;
            p_left = num_psmps - p_right + lastbinsize;
         end;
         else p_left = num_psmps - p_right;
      end;

*** Base 2-sided p-value on distance from mean of last (i.e. least extreme) bin
    of smaller p-value.  This is common practice.
***;

      if p_left<p_right then do;
         p_both = p_left;
         do z=num_psmps to 1 by -1;
            if (psums[z] - pmean) >= distance_left then p_both+1;
            else leave;
         end;
      end;
      else if p_left>p_right then do;
         p_both = p_right;
         do z=1 to num_psmps;
            if (pmean - psums[z]) >= distance_right then p_both+1;
            else leave;
         end;
      end;
      else p_both=num_psmps;
```

```
*** Account for possibility, due to psum=a particular bin value, that
    p_both>num_psmps.
***;
        p_both = min(p_both,num_psmps);

     END;

     p_left  = p_left  / num_psmps;
     p_right = p_right / num_psmps;
     p_both  = p_both  / num_psmps;

*** If CONTROL sample is smaller than TEST (which is atypical), reverse
*** p-values, as empirical distribution is mirror of itself.;

     if &samp2var.="C" then do;
           hold = p_left;
           p_left = p_right;
           p_right = hold;
        end;

     label permvar   = "Permuted Variable"
           n_psamp   = "Size of Permutation Samples"
           num_psmps = "# of Permutation Samples"
           p_left    = "Left p-value"
           p_right   = "Right p-value"
           p_both    = "Two-Tailed p-value"
           ;
     output &outdata.;
  end;
  else output error;
  run;

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

%MEND PROCSS;

%PROCSS(num_psmps = 1000,
        indata    = MFPTUS.pricing_data_2strata_100000,
        outdata   = MFPTUS.PROCSS_100000_2strata,
        byvars    = geography segment,
        samp2var  = cntrl_test,
        permvar   = price,
        seed      =
        );
```

***** PROC_MT ***;**
***** PROC_MT ***;**
***** PROC_MT ***;**

```
%MACRO PROCMT(num_psmps=,
              indata=,
              outdata=,
              byvars=,
              samp2var=,
```

```
              permvar=,
              seed=,
                 left_or_right=
              );


*** If user does not pass a value to the optional macro variable "seed,"
    generate a random seed and use it for all three proc multtests below
       (although SAS OnlineDoc says seed=-1 should work, it does not).
***;
%if %sysevalf(%superq(seed)=,boolean)
%then %let seed=%sysfunc(ceil(%sysevalf(1000000000*%sysfunc(ranuni(-1)))));

*** Un/comment test statements below to perform permutation tests based
    on different assumptions about the variance structure of the samples.;
***;

proc multtest    data = &indata.
              nsample = &num_psmps.
                    seed = &seed.
                     out = mt_output_results_2t(keep=&byvars. perm_p
rename=(perm_p=xp_both))
                    permutation
                      noprint;
  by &byvars.;
  class &samp2var.;
*  test mean (&permvar. / DDFM=SATTERTHWAITE);
  test mean (&permvar.);
  run;


*** To make runtime results comparable to PROC NPAR1WAY, which provides only
    two-tailed p-value and the smaller of the right or left p-values, run
       two of the three PROC MULTTESTs and calculate the second tail as one
       minus the given tail, which will usually be very close to the actual
       value unless the data is highly discretized.
***;

proc multtest    data = &indata.
              nsample = &num_psmps.
                    seed = &seed.
%if %UPCASE(%sysfunc(compress(&left_or_right.)))=RIGHT %then %do;
                     out = mt_output_results_up(keep=&byvars. perm_p
rename=(perm_p=xp_right))
%end;
%if %UPCASE(%sysfunc(compress(&left_or_right.)))=LEFT %then %do;
               out = mt_output_results_low(keep=&byvars. perm_p rename=(perm_p=xp_left))
%end;
                    permutation
                      noprint;
  by &byvars.;
  class &samp2var.;
%if %UPCASE(%sysfunc(compress(&left_or_right.)))=RIGHT %then %do;
*  test mean (&permvar. / upper DDFM=SATTERTHWAITE);
  test mean (&permvar. / upper);
%end;
%if %UPCASE(%sysfunc(compress(&left_or_right.)))=LEFT %then %do;
*  test mean (&permvar. / lower DDFM=SATTERTHWAITE);
  test mean (&permvar. / lower);
%end;
  run;
```

```
proc summary data=&indata. nway;
  class &byvars. &samp2var.;
  var &permvar.;
  output out=byvar_frq(keep = _FREQ_ &byvars.)
                         n = toss
                         ;
  run;

%let last_byvar = %scan(&byvars.,-1);
data byvar_frq(keep=&byvars. xn_psamp sortedby=&byvars.);
  set byvar_frq(rename=(_FREQ_=xn_psamp));
  by &byvars.;
  retain lag_FREQ;
  if first.&last_byvar. then lag_FREQ = xn_psamp;
  else do;
     if xn_psamp<=lag_FREQ then output;
        else do;
           xn_psamp  = lag_FREQ;
           output;
        end;
  end;
  run;


data &outdata.(drop=xn_psamp xp_left xp_both)
     error
        ;
%if %UPCASE(%sysfunc(compress(&left_or_right.)))=LEFT %then %do;
  merge mt_output_results_low(in=inlow)
              mt_output_results_2t(in=in2t)
        byvar_frq(in=infrq)
        ;
  if in2t & inlow & infrq then do;
     format permvar $32.;
     permvar = "&permvar.";
        n_psamp = xn_psamp;
     num_psmps = 1*&num_psmps.;
        p_left = xp_left;
        p_right = .;
%end;
%if %UPCASE(%sysfunc(compress(&left_or_right.)))=RIGHT %then %do;
  merge mt_output_results_up(in=inup)
              mt_output_results_2t(in=in2t)
        byvar_frq(in=infrq)
        ;
  if in2t & inup & infrq then do;
     format permvar $32.;
     permvar = "&permvar.";
        n_psamp = xn_psamp;
     num_psmps = 1*&num_psmps.;
        p_right = xp_right;
        p_left = .;
%end;
        p_both = xp_both;
     label permvar   = "Permuted Variable"
           n_psamp   = "Size of Permutation Samples"
           num_psmps = "# of Permutation Samples"
           p_left    = "Left p-value"
           p_right   = "Right p-value"
           p_both    = "Two-Tailed p-value"
```

```
            ;
      output &outdata.;
   end;
   else output error;
   run;


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

%MEND PROCMT;

%PROCMT(num_psmps     = 1000,
        indata        = MFPTUS.pricing_data_2strata_100000,
        outdata       = MFPTUS.PROCMT_100000_2strata,
        byvars        = geography segment,
        samp2var      = cntrl_test,
        permvar       = price,
        seed          = ,
           left_or_right = left
        );
```

***** PROCNPAR ***;**
***** PROCNPAR ***;**
***** PROCNPAR ***;**

```
%MACRO PROCNPAR(num_psmps=,
                indata=,
                outdata=,
                byvars=,
                samp2var=,
                permvar=,
                seed=
                );

*** If user does not pass a value to the optional macro variable "seed," use -1
    based on the time of day.
***;
%if %sysevalf(%superq(seed)=,boolean) %then %let seed=-1;

ods listing close;
proc npar1way    data = &indata.
               scores = data
                              ;
  var &permvar.;
  by &byvars.;
  class &samp2var.;
  exact scores=data / n=&num_psmps. seed=&seed.;
  ods output DataScoresMC = hold(keep = &byvars. Name1 Label1 nValue1
                                where = (Label1 = "Estimate")
                                );
  run;
ods listing;

proc transpose data = hold(drop=Label1)
```

```
                out = &outdata.(drop = _NAME_);
  by &byvars.;
  id Name1;
  var nValue1;
  run;

proc summary data=&indata. nway;
  class &byvars. &samp2var.;
  var &permvar.;
  output out=byvar_frq(keep = _FREQ_ &byvars. &samp2var.)
                          n = toss
                          ;
  run;

%let last_byvar = %scan(&byvars.,-1);
data byvar_frq(keep=&byvars. permvar n_psamp num_psmps &samp2var. sortedby=&byvars.);
  set byvar_frq;
  by &byvars.;
  format permvar $32.;
  retain permvar "&permvar." lag_FREQ lag_samp2 ;
  n_psamp = _FREQ_;
  num_psmps = 1*&num_psmps.;
  if first.&last_byvar. then do;
     lag_FREQ = n_psamp;
        lag_samp2  = &samp2var.;
  end;
  else do;
     if n_psamp<=lag_FREQ then output;
        else do;
        n_psamp  = lag_FREQ;
           &samp2var. = lag_samp2;
           output;
        end;
  end;
  run;

data &outdata.(drop=hold mcpl_data mcpr_data mcp2_data &samp2var.)
     error
        ;
  merge byvar_frq(in=infrq)
           &outdata.(in=inresults)
        ;
  by &byvars.;
  if inresults & infrq then do;
  p_left  = mcpl_data;
  p_right = mcpr_data;
  p_both  = mcp2_data;
  label permvar   = "Permuted Variable"
        n_psamp   = "Size of Permutation Samples"
        num_psmps = "# of Permutation Samples"
        p_left    = "Left p-value"
        p_right   = "Right p-value"
        p_both    = "Two-Tailed p-value"
        ;

*** If CONTROL sample is smaller than TEST (which is atypical), reverse
*** p-values, as empirical distribution is mirror of itself.;

   if &samp2var.="C" then do;
      hold = p_left;
         p_left = p_right;
```

```
        p_right = hold;
  end;
     output &outdata.;
     end;
     else output error;
  run;

%MEND PROCNPAR;

%PROCNPAR(num_psmps = 1000,
          indata    = MFPTUS.pricing_data_2strata_100000,
          outdata   = MFPTUS.PROCNPAR_100000_2strata,
          byvars    = geography segment,
          samp2var  = cntrl_test,
          permvar   = price,
          seed      =
          );
```

```
*** BEBB_SIM ***;
*** BEBB_SIM ***;
*** BEBB_SIM ***;

%MACRO BEBB_SIM(num_psmps=,
                indata=,
                outdata=,
                byvars=,
                samp2var=,
                permvar=,
                seed=
                );

*** If user does not pass a value to the optional macro variable "seed," use -1
    based on the time of day.
***;
%if %sysevalf(%superq(seed)=,boolean) %then %let seed=-1;


*** Obtain counts for each strata.;

proc summary data=&indata. nway;
  class &byvars. &samp2var.;
  var &permvar.;
  output out=byvar_sum(keep = _FREQ_ &byvars. &samp2var. sumpvar)
                       sum = sumpvar
                       ;
  run;

%let last_byvar = %scan(&byvars.,-1);
data byvar_sum_min(keep=tot_FREQ _FREQ_ &byvars. &samp2var. sumpvar sortedby=&byvars.);
  set byvar_sum;
  by &byvars.;
  retain lag_FREQ lag_sum lag_samp2;
  if first.&last_byvar. then do;
       lag_FREQ = _FREQ_;
       lag_sum  = sumpvar;
       lag_samp2  = &samp2var.;
  end;
```

```
  else do;
     tot_FREQ = sum(lag_FREQ,_FREQ_);
     if _FREQ_<=lag_FREQ then output;
        else do;
              _FREQ_  = lag_FREQ;
              sumpvar = lag_sum;
              &samp2var. = lag_samp2;
              output;
        end;
  end;
  run;

*** Slightly faster to use macro variable value strings and scan() than to merge
    _FREQ_ etc. onto the "indata" dataset, especially for large "indata"
       datasets.
*** ;
proc sql noprint;
  select _freq_ into :freqs separated by ' ' from byvar_sum_min;
  quit;
proc sql noprint;
  select tot_FREQ into :tot_FREQs separated by ' ' from byvar_sum_min;
  quit;
proc sql noprint;
  select sumpvar into :sumpvar separated by ' ' from byvar_sum_min;
  quit;
proc sql noprint;
  select &samp2var. into :samp2var separated by ' ' from byvar_sum_min;
  quit;


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

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

data &outdata.(keep=&byvars. permvar n_psamp num_psmps p_left p_right p_both);
  set &indata.(keep=&byvars. &permvar.) end=lastrec;
  by &byvars.;
  retain permvar "&permvar" n_psamp 0 num_psmps &num_psmps.;

  array smalln_counter{&num_psmps.} _TEMPORARY_;
  array psums{&num_psmps.} _TEMPORARY_;

  if first.&last_byvar. then do;

        byval_counter+1;

     _freq_ = 1*scan("&freqs.",byval_counter,' ');
     n_psamp = _FREQ_;
     tot_FREQ = 1*scan("&tot_FREQs.",byval_counter,' ');

        bigN_counter = tot_FREQ+1;
        do i=1 to num_psmps;
           smalln_counter[i] = _FREQ_;
           psums[i] = 0;
        end;
  end;
  bigN_counter+(-1);

  min_mac_res_inloop = &permvar.;
```

```
  seed=1*&seed.;
  if last.&last_byvar.~=1 then do i=1 to num_psmps;
     if ranuni(seed) <= smalln_counter[i]/bigN_counter then do;
        psums[i] = min_mac_res_inloop + psums[i];
           smalln_counter[i]+(-1);
        end;
  end;
  else do i=1 to num_psmps;
     if smalln_counter[i]>0 then psums[i] = min_mac_res_inloop + psums[i];
  end;
  if last.&last_byvar. then DO;


*** If CONTROL sample is smaller than TEST (which is atypical), reverse
*** order of empirical distribution.;

     sumpvar = 1*scan("&sumpvar.",byval_counter,' ');

     p_left  = 0;
     p_right = 0;
     p_both  = 0;
     call sortn(of psums[*]);
     pmed = median(of psums[*]);
     pmean = mean(of psums[*]);

*** Efficiently handle extreme test sample values.;

     IF sumpvar<psums[1] THEN DO;
        p_left=0;
        p_right=num_psmps;
        p_both=0;
     END;
     ELSE IF sumpvar>psums[num_psmps] THEN DO;
        p_left=num_psmps;
        p_right=0;
        p_both=0;
     END;
     ELSE DO;

*** For non-extreme cases, start with shorter tail for less looping.;

     if pmed>=sumpvar then do;
        do z=1 to num_psmps;
           if sumpvar>=psums[z] then p_left+1;
           else do;
              lastbinnum = z-1;
              distance_left = pmean - psums[z-1];
              leave;
           end;
        end;

*** Avoid loop for other (larger) p-value.
    If sumpvar equals last bin, p_right = 1 - p_left + lastbinsize.
    Otherwise, p_right = 1 - p_left.
***;
        if sumpvar = psums[lastbinnum] then do;
           lastbinsize=1;
           do k=lastbinnum to 1 by -1;
              if psums[k]=psums[k-1] then lastbinsize+1;
              leave;
```

```
         end;
         p_right = num_psmps - p_left + lastbinsize;
      end;
      else p_right = num_psmps - p_left;
   end;

   else do;
      do z=num_psmps to 1 by -1;
         if sumpvar<=psums[z] then p_right+1;
         else do;
            lastbinnum = z+1;
            distance_right = psums[z+1] - pmean;
            leave;
         end;
      end;

*** Avoid loop for other (larger) p-value.
    If psum equals last bin, p_left = 1 - p_right + lastbinsize.
    Otherwise, p_left = 1 - p_right.
***;
      if sumpvar = psums[lastbinnum] then do;
         lastbinsize=1;
         do k=lastbinnum to num_psmps;
            if psums[k]=psums[k+1] then lastbinsize+1;
            else leave;
         end;
         p_left = num_psmps - p_right + lastbinsize;
      end;
      else p_left = num_psmps - p_right;
   end;

*** Base 2-sided p-value on distance from mean of last (i.e. least extreme) bin
    of smaller p-value.  This is common practice.
***;
   if p_left<p_right then do;
      p_both = p_left;
      do z=num_psmps to 1 by -1;
         if (psums[z] - pmean) >= distance_left then p_both+1;
         else leave;
      end;
   end;
   else if p_left>p_right then do;
      p_both = p_right;
      do z=1 to num_psmps;
         if (pmean - psums[z]) >= distance_right then p_both+1;
         else leave;
      end;
   end;
   else p_both=num_psmps;

*** Account for possibility, due to psum=a particular bin value, that
    p_both>num_psmps.
***;
   p_both = min(p_both,num_psmps);
      END;

   p_left  = p_left  / num_psmps;
   p_right = p_right / num_psmps;
   p_both  = p_both  / num_psmps;

*** If CONTROL sample is smaller than TEST (which is atypical), reverse
```

```
*** p-values, as empirical distribution is mirror of itself.;

     if "C"=COMPRESS(UPCASE(scan("&samp2var.", byval_counter,' ')),' ') then do;
           hold = p_left;
           p_left = p_right;
           p_right = hold;
     end;

     label permvar   = "Permuted Variable"
           n_psamp   = "Size of Permutation Samples"
           num_psmps = "# of Permutation Samples"
           p_left    = "Left p-value"
           p_right   = "Right p-value"
           p_both    = "Two-Tailed p-value"
           ;
      output &outdata.;
  END;
  run;

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

%MEND BEBB_SIM;

%BEBB_SIM(num_psmps = 1000,
          indata    = MFPTUS.pricing_data_2strata_100000,
          outdata   = MFPTUS.BEBB_SIM_100000_2strata,
          byvars    = geography segment, samp2var=cntrl_test,
          permvar   = price,
          seed      =
           );
```

## Appendix B

For the convenience of the reader, a proof is provided below for the validity of the SRSWOR procedure used by OPDN, which is essentially the approach of Goodman & Hedetniemi (1977). The algorithm as implemented in OPDN is shown again below.

OPDN implementation of Goodman & Hedetniemi (1977) for Permutation Tests:

*** temp[] is the array filled with all the data values, for current stratum, of the variable being permuted
*** psums[] is the array containing the permutation sample statistic values for every permutation sample

```
do m = 1 to #permutation tests
   x ← 0
   tot_FREQ_hold ← # records in current stratum
   tot_FREQ ← tot_FREQ_hold
   do n = 1 to # records in smaller of Control and Treatment samples
      cell ← uniform random variate on 1 to tot_FREQ
      x    ← temp[cell] + x
      hold ← temp[cell]
      temp[cell] ← temp[tot_FREQ]
      temp[tot_FREQ] ← hold
      tot_FREQ ← tot_FREQ -1
   end;
   psums[m] ← x
end;
```

For a sampling algorithm to be a valid SRSWOR procedure, the probability of selecting any without-replacement sample of $n$ items from a population of $N$ items ($N > n$) needs to equal the probability of selecting any other without-replacement sample of $n$ items, and that probability is:

Pr(drawing any particular sample of $n$ items from larger group of $N$ items) = $1\Big/\binom{N}{n} = n!(N-n)!\big/N!$ (C1)

because there are $N$ – choose – $n$ possible without-replacement samples.

Using the algorithm shown above, the probability of drawing the first item is $1/N$, and the probability of drawing the second item is $1/(N\text{-}1)$ and the probability of drawing the third item is $1/(N\text{-}2)$, and so on. Because each of these draws is independent of the others, the probability of drawing a sample of any $n$ items is the product of these probabilities:

Pr(drawing any particular sample of $n$ items from larger group of $N$ items) =
$\frac{1}{N}\cdot\frac{1}{(N-1)}\cdot\frac{1}{(N-2)}\cdot\ldots\cdot\frac{1}{(N-n+2)}\cdot\frac{1}{(N-n+1)}$ * the number of permutations of these $n$ items (because we do not care about the ordering of the $n$ items, only that a particular set of $n$ items is drawn), which is $n!$.

So Pr(drawing any particular sample of $n$ items from larger group of $N$ items) = $n!\Big/\prod_{i=0}^{n-1}(N-i)$ (B2)

But in fact, (B1) = (B2), as shown below.

(B1) = $$\frac{n!\left(N-n\right)!}{N!}=\frac{\left[n\left(n-1\right)\left(n-2\right)\cdot\ldots\cdot 2\cdot 1\right]\left[\left(N-n\right)\left(N-n-1\right)\left(N-n-2\right)\cdot\ldots\cdot 2\cdot 1\right]}{N\left(N-1\right)\left(N-2\right)\cdot\ldots\cdot\left(N-n+1\right)\left(N-n\right)\left(N-n-1\right)\cdot\ldots\cdot 2\cdot 1}=$$

$$=\frac{\left[n\left(n-1\right)\left(n-2\right)\cdot\ldots\cdot 2\cdot 1\right]}{N\left(N-1\right)\left(N-2\right)\cdot\ldots\cdot\left(N-n+1\right)}=\frac{n!}{\prod_{i=0}^{n-1}\left(N-i\right)}=$$

= (B2)

## Appendix C

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

PURPOSE:  Run and compare different SAS bootstrap algorithms including OPDY, PSS, DA,
          Out-SM, and A4.8.  See Opdyke, J.D., "Much Faster Bootstraps Using SAS,"
          InterStat, October, 2101, for detailed explanations.

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 temp{&max_freq.} _TEMPORARY_;

  %if &sysver >= 9.2 %then %do;
    array bmeans{&num_bsmps.} _TEMPORARY_;
    retain byval_counter 0 cum_prev_freq 0;
  %end;
  %if &sysver < 9.2 %then %do;
    array bmeans {&num_bsmps.} bmean1-bmean&num_bsmps.;
    retain byval_counter 0 cum_prev_freq 0 bmean1-bmean&num_bsmps.;
  %end;
  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 bmeans[*]);
     bstd = std(of bmeans[*]);
     b975 = pctl(97.5, of bmeans[*]);
     b025 = pctl(2.5, of bmeans[*]);
     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 D**

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 $\left(1/N\right)^n$ or $1/N^n$.[1] However, the order of selection does not matter for these purposes,[2] 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

$$\frac{n!}{b_1!b_2!\ldots b_n!}\cdot\prod_{i=1}^{n}\left(\frac{1}{n}\right)^{b_i} \qquad \text{(D1)}$$

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 $\prod_{i=1}^{n}\left(1/n\right)^{b_i}=\left(1/n\right)^n$ . So when $n \le N$, (D1) is simply

$$\frac{n!}{b_1!b_2!\ldots b_N!}\cdot\left(\frac{1}{N}\right)^n \qquad \text{(D2)}$$

To show that the probability that any sample of $n$ items drawn from $N$ items ($n \le N$) using Algo4.8 is equal to (D2), note that the probability that any of the $n$ items in the sample is drawn $b$ times, where $b$ is $0 \le$ positive integers $\le n'$, is by definition the binomial probability (using $n'$ and $p$ as defined in Algo4.8)

$$\Pr\left(b_i=b_i^*\right)=\binom{n'}{b_i^*}p^{b_i^*}\left(1-p\right)^{n'-b_i^*} \quad \text{for } 0<p<1 \text{ [3]} \qquad \text{(D3)}$$

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

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

[2] 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.

[3] 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) =**

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

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

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

$$\vdots$$

$$\frac{(n-b_1-b_2-\ldots-b_{N-1})!}{b_N!(n-b_1-b_2-\ldots-b_N)!}\left(\frac{1}{N-(N-1)}\right)^{b_N}\left(1-\frac{1}{N-(N-1)}\right)^{n-b_1-b_2-b_3-\ldots-b_N} \quad \text{(D4)}$$

Reordering terms gives

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

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

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

$$\left(\frac{1}{N-(N-1)}\right)^{b_N}\left(\frac{N-N}{N-(N-1)}\right)^{n-b_1-b_2-b_3-\ldots-b_N} \quad \text{(D5)}$$

Because $n=\sum_{i=1}^{n} b_i$ , the denominator in the last "combinatoric" term in (D5) 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 $n!/b_1!b_2!\ldots b_N!$. Of the remaining "probability" terms, the final term can be written as

$$\left(\frac{1}{N-(N-1)}\right)^{b_N}\left(\frac{1}{N-(N-1)}\right)^{n-b_1-b_2-b_3-\ldots-b_N}\left(\frac{0}{1}\right)^0$$

. 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),[4] 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

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

, which is (D2).

Excel workbook examples of calculating this probability using both (D2) directly and the steps of Algo4.8, for both $n \leq N$ and $n = N$, are available from the author upon request.

## References


[1] 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.

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

[3] Bentley, J. L. and Knuth, D. (1986), “Literate Programming,” *Communications of the Association for Computing Machinery*, 29, 364–369.

[4] 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.

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

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

[7] Edgington, E.S., and Onghena, P. (2007), Randomization Tests, Fourth Edition, Chapman & Hall/CRC.

[8] Efron, B. (1979), “Bootstrap Methods: Another Look at the Jackknife,” *Annals of Statistics*, 7, 1–26.

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

[10] Ernvall, J., & O. Nevalainen (1982), “An Algorithm for Unbiased Random Sampling,” *The Computer Journal*, Vol.25 (1), p.45-47.

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

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

[13] Fan, C. T., Muller, M. E., and Rezucha, I. (1962), “Development of Sampling Plans by Using Sequential (Item by Item) Selection Techniques and Digital Computers, ” *Journal of the American Statistical Association*, 57, 387–402.


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

[14] Fisher, Sir R.A. (1935), *Design of Experiments*, Edinburgh, Oliver & Boyd.
[15] Goodman, S. & S. Hedetniemi (1977), *Introduction to the Design and Analysis of Algorithms*, McGraw-Hill, New York.
[16] Good, P. (2004), Permutation, Parametric, and Bootstrap Tests of Hypotheses, Third Edition, Springer Verlag.
[17] Graham, R., Knuth, D., and Patashnik, O. (1994), *Concrete Mathematics: A Foundation for Computer Science*, 2nd ed., Reading, MA: Addison-Wesley Publishing Company.
[18] Knuth, D. (May, 1992), "Two Notes on Notation," *The American Mathematical Monthly*, Vol. 99, No. 5.
[19] Manly, B. F. J., (2007), Randomization, Bootstrap and Monte Carlo Methods in Biology, Third Edition, Chapman & Hall/CRC.
[20] Mielke, P. & K. Berry (2001), *Permutation Methods: A Distance Function Approach*, Springer-Verlag, New York.
[21] Opdyke, J.D. (2010), "Much Faster Bootstraps Using SAS®," *InterStat*, October, 2010.
[22] Opdyke, J.D. (2011), "Permutation Tests (and Sampling Without Replacement) Orders of Magnitude Faster Using SAS®," *InterStat*, January, 2011.
[23] Pesarin, F. (2001), *Multivariate Permutation Tests with Applications in Biostatistics*, John Wiley & Sons, Ltd., New York.
[24] SAS Institute Inc. (2007), *SAS OnlineDoc 9.2*, Cary, NC: SAS Institute Inc.
[25] Secosky, J., and Bloom, J. (May, 2007), "Getting Started with the DATA Step Hash Object," SAS Institute, Inc.
[26] Secosky, J., and Bloom, J. (2008), "Better Hashing in SAS® 9.2," SAS Institute, Inc., Paper 306-2008.
[27] Tillé, Y. (2006), *Sampling Algorithms*, New York, NY, Springer.
[28] Tillé, Y. (2010), *eMail Correspondence*, September 30, 2010.
[29] Vaughan, H. (February 1970), "The Expression of $0^0$," *The Mathematics Teacher*, Vol. 63, p.111.

**Related Articles**

| Article ID | Article title |
|---|---|
| 098 | Bootstrap |
| 096 | Resampling |
| 551 | Bootstrapping Regression |