C++ Code

In the following listing, the line

\[ \text{for (j=0; j<n; j++) } Y[j] = X[\text{random(n)}]; \]

selects a random sample from \( X \) with replacement. Replace the call to \text{compute_statistic}(Y)\) with
a call to the function that computes the statistic whose precision you wish to estimate.

```cpp
#include <stdlib.h>

get_data(); // put the variable of interest in the first n elements of the array X[].
randomize(); // initializes random number generator

for (i=0; i<100; i++){
    for (j=0; j<n; j++) Y[j]=X[\text{random(n)}];
    Z[i]=\text{compute_statistic}(Y);
    // compute the statistic for the array Y and store it in Z
    compute_stats(Z);
}
```

EViews

```eviews
create u 1 22
series x
x.fill 141,156.5,162,159,157,143.5,154,158,140,142,150,148.5,138.5,161,153,145,147,158.5,160.5,167.5,155,137

!rep=50 ' number of replications
vector(!rep) var_boot
For li = 1 to !rep
    X.resample XB
    var_boot(li) = @var(XB)
Next

scalar q5=@quantile(var_boot,.05) '5% quantile of the bootstrap
scalar q95=@quantile(var_boot,.95) '95% quantile of the bootstrap
```
GAUSS

(This routine only generates a single resample.)

\[ n = \text{rows}(Y); \]
\[ U = \text{rnds}(n, 1, \text{integer seed}); \]
\[ I = \text{trunc}(n \times U + \text{ones}(n, 1, 1)); \]
\[ Y_{\star} = Y[I, .]; \]

MatLab

%% Bootstrap Method
% This function takes N bootstrap samples with replacement of the
% original data and stores the calculation of the desired statistic
% in the returned array.

function [stat] = bootstrap(data, N)
    \%reset random generator to different state at each time
    \textbf{rand('state', sum(100*clock))};
    \%initialize array to be returned of bootstrap estimations
    \textbf{stat} = \textbf{zeros(1, N)};
    \%initialize array to hold bootstrap sample
    \textbf{sample} = \textbf{zeros(1, n)};

    \textbf{for} \textbf{i} = \textbf{1:N}
        \%to be ran before executing the function:
        \%\textbf{clear all;}
        \%\textbf{data=} \{141, 156.5, 162, 159, 157, 143.5, 158, 140, 142, 150, 148.5, 138.5, 161, 153, 145, 147, 158.5, 160.5, 167.5, 155, 137\};
        \%to run: \textbf{bootstrap(data, 100)}

        \textbf{end}

    \textbf{end}
choose=round(((n-1)*rand(1,n))+1);
%choose an array length n of random #'s between [1,n]
%the sampled data will come from the values at these random indices.

%fill the sample array with values at randomized indices
for j=1:n
    sample(j)=data(choose(j));
end;
stat(i)=mean(sample); %fill stat array with bootstrap estimations
end;

R
#This R program selects 50 bootstrap samples from the classroom data
#and then produces a boxplot and stripchart of their variances.
class=c(141,156.5,162,159,157,143.5,154,158,140,142,150,148.5,138.5,
161,153,145,147,158.5,160.5,167.5,155,137)
#record group size
n = length(class)
#set number of bootstrap samples
N =50
stat = numeric(N) #create a vector in which to store the results
    #the elements of the vector will be numbered from 1 to N
#Set up a loop to generate a series of bootstrap samples
for (i in 1:N){
    #bootstrap sample counterparts to observed samples are denoted with "B"
    classB= sample (class, n, replace=T)
    stat[i] = var(classB)
}
boxplot (stat)
stripchart(stat)

Resampling Stats
The following Resampling Stats program yields a histogram of median values derived from 100 bootstrap samples of the classroom data.

```
DATA (141 156.5 162 157 154 158 140 142 150 148.5 138.5 161 153 145 147 158.5 160.5 167.5 155 137) A
MEDIAN A med_orig
REPEAT 100
   SAMPLE 21 A B
   MEDIAN B med
   SCORE med scrboard
END
HISTOGRAM scrboard
```

**SAS Code**

```
%macro boot(      /* Bootstrap resampling analysis */
   data=,         /* Input data set, not a view or a tape file. */
   samples=200,   /* Number of resamples to generate. */
   residual=,     /* Name of variable in the input data set that contains residuals; may not be used with SIZE= */
   equation=,     /* Equation (in the form of an assignment statement) for computing the response variable */
   size=,         /* Size of each resample; default is size of the input data set. The SIZE= argument may not be used with BALANCED=1 or with a nonblank value for RESIDUAL= */
   balanced=,     /* 1 for balanced resampling; 0 for uniform resampling. By default, balanced resampling is used unless the SIZE= argument is specified, in which case uniform resampling is used. */
   random=0,      /* Seed for pseudorandom numbers. */
   stat=_numeric_/* Numeric variables in the OUT= data set created by the %ANALYZE macro that contain the values of statistics for which you want to compute bootstrap distributions. */
   */
```
id=,  /* One or more numeric or character variables that
uniquely identify the observations of the OUT=
data set within each BY group. No ID variables
are needed if the OUT= data set has only one
observation per BY group.
The ID variables may not be named _TYPE_, _NAME_,
or _STAT_ */
biascorr=1,  /* 1 for bias correction; 0 otherwise */
alpha=.05,  /* significance (i.e., one minus confidence) level
for confidence intervals; blank to suppress normal
confidence intervals */
print=1,  /* 1 to print the bootstrap estimates;
0 otherwise. */
chart=1  /* 1 to chart the bootstrap resampling distributions;
0 otherwise. */
);

%if %bquote(&data)= %then %do;
  %put ERROR in BOOT: The DATA= argument must be specified.;
  %goto exit;
%end;

%global _bootdat; %let _bootdat=&data;

%local by useby;
%let useby=0;

%global usevardf vardef;
%let usevardf=0;

*** compute the actual values of the statistics;
%let vardef=DF;
%let by=;
%analyze(data=&data,out=_ACTUAL_);
%if &syserr>4 %then %goto exit;

*** compute plug-in estimates;
%if &usevardf %then %do;
  %let vardef=N;
  %analyze(data=&data,out=_PLUGIN_);
  %let vardef=DF;
  %if &syserr>4 %then %goto exit;
%end;

%if &useby=0 %then %let balanced=0;

%if %bquote(&size)^= %then %do;
  %if %bquote(&balanced)= %then %let balanced=0;
  %else %if %bquote(&balanced)= %then %let balanced=0;
  %put %cmpres(ERROR in BOOT: The SIZE= argument may not be used with BALANCED=1.);
  %goto exit;
%end;
%end;

%if %bquote(&residual)^= %then %do;
  %put %cmpres(ERROR in BOOT: The SIZE= argument may not be used with RESIDUAL=.);
  %goto exit;
%end;
%end;
%else %if %bquote(&balanced)= %then %let balanced=1;

*** find number of observations in the input data set;
%global _nobs;
data _null_;  
call symput('_nobs',trim(left(put(_nobs,12.))));
if 0 then set &data nobs=_nobs;
stop;
run;
%if &syserr>4 %then %goto exit;

%if &balanced %then
   %bootbal(data=&data,samples=&samples,
           random=&random,print=0);

%else %if &useby %then
   %bootby(data=&data,samples=&samples,
           random=&random,size=&size,print=0);

%if &syserr>4 %then %goto exit;

%if &balanced | &useby %then %do;
   %let by=_sample_;
   %analyze(data=BOOTDATA,out=BOOTDIST);
%end;

%else
   %bootslow(data=&data,samples=&samples,
             random=&random,size=&size);

%if &syserr>4 %then %goto exit;

%if &chart %then %do;
   %if %bquote(&id)^= %then %do;
      proc sort data=BOOTDIST; by &id; run;
      proc chart data=BOOTDIST(drop=_sample_);
      vbar &stat;
      by &id;
      run;
   %end;
%else %do;
      proc chart data=BOOTDIST(drop=_sample_);
      vbar &stat;
run;
%end;
%end;

%bootse(stat=&stat,id=&id,alpha=&alpha,biascorr=&biascorr,print=&print)

%exit:;

%mend boot;

**S-Plus**
Download the S+Resample package.

boot = bootstrap(urdata, median)

boxplot(boot)

plot(boot, 0*boot)

**Stata**
Once the height data is entered, the following line of code invokes the bootstrap to produce a 95% confidence interval for the interquartile deviation, plus a point estimate of its bias.

```
bootstrap "summarize height,detail" (r(p75)-r(p25)), reps(100) nobc nonormal
```

**Computer Code : The BCₐ Bootstrap**

**R**

```r

f.median<- function(y,id){
+    median( y[id])
+  }

boot.ci(boot(classdata, f.median, 400), conf = 0.90)
```

**SAS**
**S-Plus**

```splus
boot = bootstrap(data, median)
limits.bca(boot)
```

**Stata**

Once the height data is entered, the following line of code invokes the bootstrap to produce a 95% confidence interval for the interquartile deviation, plus a point estimate of its bias.

```stata
bootstrap "summarize height,detail" (r(p75)-r(p25)), reps(100) nonormal nopercentile
```

**Variance-Stabilized Bootstrap-t**

**R**

```r
class=c(141,156.5,162,159,157,143.5,154,158,140,142,150,148.5,138.5,
161,153,145,147,158.5,160.5,167.5,155,137)

# download the bootstrap package from http://rweb.stat.umn.edu/R/library/bootstrap/R/
#install manually, for example, source("/temp/Documents/RData/bootstrap")
library(bootstrap)
# find an 80% CI for the population variance
results=boott(class, var, VS=TRUE, v.nbootg=50, v.nbootsd=50, v.nboott=200, perc=c(.10,.90))
results[1]
```

**Parametric Bootstrap**

**MatLab**

```matlab
% Parametric Bootstrap
% This function calculates desired estimations
% using the parametric bootstrap method.
% This problem uses the exponential distribution
% to calculate an array of IQR's of the bootstrapped data.
% To be ran before the function:
% clear all;
% class=[141,156.5,162,159,157,143.5,154,158,140,142,150,148.5,138.5,161,153,145,147,158.5,160.5,167.5,155,137];
% To run function:
% parametric(class,100)
```
function [stat]=parametric(data,N)
    rand('state', sum(100*clock)); %reset random generator to a different state
    n=length(data);
    stat=zeros(1,N);
    samp=zeros(1,n);
    for i=1:N
        choose=round(((n-1)*rand(1,n))+1); %randomize indices for bootstrap sample
        for j=1:n
            samp(j)=data(choose(j)); %set bootstrap sample from choose indices
        end
        lambda=1/mean(samp);
        stat(i)=log(1/3)/(-lambda); %IQR for an exponential function
    end

R
#The following R program fits an exponential distribution to the data set A
#Then uses a parametric bootstrap to get a 90% confidence interval for the IQR of the population
#from which the data set A was taken.
#n=length(A)
#create a vector in which to store the IQR’s
IQR = numeric(1000)
#Set up a loop to generate the 1000 IQR’s
for (i in 1:1000) {
    bA=sample (A, n, replace=T)
    IQR[i]  = qexp(.75,1/mean(bA))–qexp(.25, 1/mean(bA))
}
quantile (IQR , probs = c(.05,.95))

Resampling Stats
‘The following program fits an exponential distribution to the data set A
‘Then uses a parametric bootstrap to get a 90% confidence interval for the IQR of the ‘population
from which the data set A was taken.
MAXSIZE board 1000
MEAN A x_bar
REPEAT 1000
    EXPONENTIAL 100 x_bar X
PERCENTILE X (25 75) P
Once the height data is entered, the following line of code invokes the parametric bootstrap with a normal distribution to produce a 95% confidence interval for the interquartile range.

```stata
bstrap "summarize height,detail" (r(p75)-r(p25)), reps(100) nobc nopercentile
```

### Chapter 3

#### Two-sample Comparison

3.4.1. Program Code

To compare teaching methods, 20 school children were randomly assigned to one of two groups. The following are the test results:

| conventional | 65 79 90 75 61 85 98 80 97 75 |
| new          | 90 98 73 79 84 81 98 90 83 88 |

Are the two teaching methods equivalent in result?

```c++
int Choose (int lo, int hi)
{
    int z = rand()%(hi - lo +1) + lo;
    return (z);
}

//Pack all the observations in a single vector Data.

float CompTwo(float *X){
```
float sum =0, temp;
int k;
for (int i=0; i< n[1]; ++i){
k=Choose (i, n[1]+n[2]-1);
temp = *(X+k); *(X+k)=*(X+i); *(X+i)= temp;
sum =sum +temp;
}
return (sum);
}

float stat0=0, stat;
int cnt;
for (int i =0; i < n[1]; ++i)stat0=stat0 +Data[i];

cnt=0;
for (int i =0; i < MC; ++i){
stat=Comp2(Data);
if (stat >=stat0) cnt++;
}
float p=float(cnt)/MC;
cout << p << endl;

EViews

!n = 400 ' number of rearrangements to be examined
create u 1 10
series conventional
conventional.fill 65,79,90,75,61,85,98,80,97,75
series new
new.fill 90,98,73,79,84,81,98,90,83,88
!sumorig =@sum(new)
!count = 0

range 1 20 ' expande worfile space
' place data into a new data series A
series a = new
smpl 11 20
a = conventional(-10)
smpl @all

for !i  = 1 to !n
' draw from A without replacement 10 elements
a.resample(outsmpl=1 10,permute) boot
if (@sum(boot)<=!sumorig) then
!count=!count+1
endif
next
scalar pvalue=!count/!n
show pvalue
Excel – Using Resampling Statistics for Excel

Place the two sets of observations in adjoining columns. Outline them and use the S or Shuffle command.

MatLab

% Monte Carlo for Calculating p-value
% This function takes N permuted samples of the original
% data, without replacement and calculated the desired
% statistic on the permuted sample. It then counts how
% many cases as or more extreme occured to calculate the
% p-value for the test.
%
% montecarlo(samp1,samp2,NMCS)
% samp1, samp2 are the two arrays of data to be tested
% NMCS is the number of Monte Carlo Simulations to be performed.
%
%To be ran before executing the function:
%clear all;
%conventional=[65,79,90,75,61,85,98,80,97,75];
%new=[90,98,73,79,84,81,98,90,83,88];
%to run: montecarlo(conventional,new,1000)

function [p]=montecarlo(samp1,samp2,NMCS)
    rand('state',sum(100*clock)); %reset random generator to a different state
    n=length(samp1);
    m=length(samp2);
    data=cat(2,samp1,samp2); %concatinate data arrays

    stat0=sum(samp1); %calculate test statistic
    cnt=0; %initialize count for p calculation
    sample=zeros(1,n); %initialize array to hold permutation

    for i=1:NMCS
        choose=randperm(n+m); %randomize indices
        for j=1:n
            %fill sample with random data, up to first n elements
            sample(j)=data(choose(j));
        end;
        stat1=sum(sample);
        if stat1<=stat0
            cnt=cnt+1;
        end;
    end;
    p=cnt/NMCS; %p-value
R
N=400  #number of rearrangements to be examined
  conventional =c(65, 79, 90, 75, 61, 85, 98, 80, 97, 75)
  new  = c (90, 98, 73, 79, 84, 81, 98, 90, 83, 88)
n=length (new)
sumorig = sum(new)
cnt= 0  #zero the counter
#Stick both sets of observations in a single vector
A = c(new, conventional)
for (i in 1:N){
  D= sample (A,n)
  if (sum(D) <= sumorig)cnt=cnt+1
}
cnt/N  #pvalue
[1] 0.9025

Resampling Statistics
DATA (65 79 90 75 61 85 98 80 97 75) A
SIZE A n
SUM A sumorig
DATA (90 98 73 79 84 81 98 90 83 88) B
LET cnt=0
CONCAT A B C
REPEAT 400
  SHUFFLE C D
  TAKE D 1,n E
  SUM E sumperm
  ` we count only rearrangements in the lower tail
  IF sumperm <= sumorig
  LET cnt=cnt+1
END
END
DIVIDE cnt 400 pvalue
PRINT pvalue

SPLUS
Using S+Resample, we may use the R code or do:
perm = permutationTest2(new,sum,data2 = Other,alternative=“less”)
perm  # this prints the p-value, among other things
plot(perm)  # plot shows the relationship between the observed value and the null distribution

STATATA
Enter the classification as a separate variable.
in   score   method
    score method
and so forth

```
permute score "sum score if method" sum=r(sum), reps(1000) left nowarn
command:   sum score if method
statistic: sum      = r(sum)
permute var: score
```

Matched Pairs

**EViews**

create u 8
series New
new.fill 48722, 28965, 36581, 40543, 55423, 38555, 31778, 45643
series standard
standard.fill 46555, 28293, 37453, 38324, 54989, 35687, 32000, 43289
!N=400
!sumorig=@sum(new)
!count = 0
for !i  = 1 to !N
    series stat = @recode(rnd>=0.5,standard,new)
    if (@sum(stat)<=!sumorig) then
        !count=!count+1
    endif
next
scalar pvalue=!count!/!N
show pvalue

**Excel – Using Resampling Statistics for Excel**
Place the two sets of observations in adjoining columns. Outline them and use the S or Shuffle command with the “Shuffle Within Rows” option checked.

**MatLab**

```
% % Matched Pairs
% This function tests two samples by using matched pairs.
% The function chooses between the ith element of the
% two samples, computes and compares the test statistics
% to calculate the p-value.
```
% match(samp1,samp2,N)
% samp1,samp2 are arrays containing data to be tested/
% N is the number times to perform mathed pairs.
%

% Before executing function:
% clear all;
% new=[48722,28965,36581,40543,55423,38555,31778,45643];
% standard=[46555,28293,37453,35324,54989,35687,32000,43289];
% To run function:
% match(new,standard,1000)

function [p]=match(samp1,samp2,N)
    rand('state', sum(100*clock)); %reset random generator to a different state
    n=length(samp1);
    stat0=sum(samp1); %test statistic
    samp=zeros(1,n);  %initialize sample array
    cnt=0;
    for i=1:N
        choose=rand(1,n);
        for j=1:n  %will choose between ith element of samp1 or samp2
            if choose(j)<.5
                samp(j)=samp1(j);
            else
                samp(j)=samp2(j);
            end
        end
        stat1=sum(samp); %test statistic of sample
        if stat1>=stat0
            cnt=cnt+1;
        end
    end
    p=cnt/N; %p-value

R Code
New = c(48722, 28965, 36581, 40543, 55423, 38555, 31778, 45643)
Standard=c(46555, 28293, 37453, 38324, 54989, 35687, 32000, 43289)
Diff=New–Standard
N=400   #number of rearrangements to be examined
sumorig = sum(Diff)
n=length(Diff)
stat = numeric (n)
cnt= 0  #zero the counter
for (i in 1:N){
    for (j in 1:n) stat[j]=ifelse(runif(1) < 0.5, Diff[j], -Diff[j])
        if (sum(stat) >= sumorig)cnt=cnt+1
}
cnt/N #one-sided p-value
[1] 0.032

**Stata**

drop _all
input new stand
48722 46555
28965 28293
36581 37453
40543 38324
55423 54989
38555 35687
31778 32000
45643 43289
end

set seed 1234
local reps 400
tempvar which
quietly gen `which' = 0
sum new, meanonly
scalar sumorig = r(sum)
local cnt 0
forval i = 1/`reps' {
    quietly replace `which' = cond(uniform() < 0.5, new, stand)
    sum `which', meanonly
    if r(sum) <= scalar(sumorig) {
        local ++cnt
    }
}
di `cnt'/`reps'
exit

**Unequal Variances**

**EViews**

create u 1 9
series treatmt
treatmt.fill 94, 38, 23, 197, 99, 16, 141
series control
control.fill 52, 10, 40, 104, 51, 27, 146, 30, 46
' Find observed difference
scalar obsdif = @mean(treatmt) - @mean(control)
!N = 1000
vector(!N) stat
for !i = 1 to !N
    treatmt.resample(outsmpl=1 7) treatmtB
    control.resample(outsmpl=1 9) controlB
    stat(!i) = @mean(treatmtB) - @mean(controlB)
next
scalar quant05=@quantile(stat,.05)
scalar quant95=@quantile(stat,.95)
MatLab

% Unequal Variances
% Calculates difference of means
% using the bootstrap method, with replacement.
% uneqvar(samp1,samp2,N)
% samp1,samp2 are arrays of data being tested.
% N is the number of times to perform bootstrap
%

% clear all;
% samp1=[94,38,23,197,99,16,141];
% samp2=[52,10,40,104,51,27,146,30,46];
% to run function:
% uneqvar(samp1,samp2,100)

function [stat]=uneqvar(samp1,samp2,N)
    rand('state', sum(100*clock));
    n=length(samp1);
    m=length(samp2);
    samp1B=zeros(1,n);
    samp2B=zeros(1,m);
    stat=zeros(1,N);
    for i=1:N
        for j=1:n
            samp1B(j)=samp1(round((n-1)*rand+1));
        end
        for j=1:m
            samp2B(j)=samp2(round((m-1)*rand+1));
        end
        stat(i)=mean(samp1B)-mean(samp2B);
    end

R

# Two Samples
# Efron & Tibshirani, 1993, p. 11
# The observed difference in survival times between treatment mice
# and control mice is 30.63 days. Determine a 90% confidence interval
# around this estimate using the percentile bootstrap.
treatmt = c(94, 38, 23, 197, 99, 16, 141)  # treatment group
control = c(52, 10, 40, 104, 51, 27, 146, 30, 46)  # control group
n = length (treatmt)
m = length (control)
# Find observed difference
obsdif = mean(treatmt) - mean (control)
# We want to determine whether obsdif is too large to have occurred
# solely by chance
N = 1000
stat = numeric(N)  # create a vector in which to store the results
for (i in 1:N){
    # bootstrap sample counterparts to observed samples are denoted with "B"
treatmtB = sample (treatmt, replace = T)
controlB = sample (control, replace = T)
stat[i] = mean(treatmtB) - mean(controlB)
}
quantile (stat, c(0.05, 0.95))
# If the interval does not include obsdif, reject the null hypothesis.

Resampling Stats
'Two Samples
'Efron & Tibshirani, 1993, p. 11
'The observed difference in survival times between treatment mice
'and control mice is 30.63 days. Determine a 90% confidence interval
'around this estimate. Employ the bootstrap-t.
DATA (94 38 23 197 99 16 141) treatmt  'treatment group
DATA (52 10 40 104 51 27 146 30 46) control  'control group
'Record group sizes
SIZE treatmt n
SIZE control m
'Find observed difference
MEAN treatmt tmean
MEAN control cmean
LET obsdif = tmean - cmean
'We want to determine whether obsdif is too large to have occurred
'solely by chance
'Compute std of observed diff
VARIANCE treatmt vt
VARIANCE control vc
LET den = (((n-1)*vt+(m-1)*vc))*(1/n+1/m)/(n+m-2)
SQRT den std
LET t = obsdif/std
'Bootstrap
REPEAT 1000
'Bootstrap sample counterparts to observed samples are denoted with "$"
SAMPLE n treatmt treatmt$
SAMPLE m control control$
'Find the numerator for first Hall-Wilson correction
MEAN treatmt$ tmean$
MEAN control$ cmean$
'Parentheses in next statement are essential
LET dif$ = (tmean$-cmean$)-obsdif
'Find the denominator for second Hall-Wilson correction
VARIANCE treatmt$ vt$
VARIANCE control$ vc$
LET den2 = (((n-1)*vt$+(m-1)*vc$))*(1/n+1/m)/(n+m-2)
SQRT den2 den
LET stat = dif$/den
SCORE stat board
END
'Rescale to use as CI for difference in population means
MULTIPLY std board board
ADD obsdif board board
HISTOGRAM board
PERCENTILE board (5 95) interval
PRINT interval obsdif
'If the interval does not include zero, reject the null hypothesis.

**Stata**
A dummy "treat" variable is used to distinguish the two samples enabling a stratified bootstrap to be used
```
drop _all
input treat value
  1 94  1 38  1 23  1 197  1 99  1 16  1 141
  0 52  0 10  0 40  0 104  0 51  0 27  0 146  0 30  0 46
end
```
capture program drop mydiff
program mydiff, rclass
args treat value
  sum `value' if `treat', meanonly
  return scalar mean_treat = r(mean)
  sum `value' if !`treat', meanonly
  return scalar mean_contr = r(mean)
  return scalar diff = return(mean_treat) - return(mean_contr)
end
set seed 1234
bootstrap "mydiff treat value" r(diff), strata(treat) nowarn
exit

**Comparing Variances**

**MatLab**
```
function [p]=vartest(samp1,samp2,N)
  % Variance test using Aly's statistic
  % This function takes in two data arrays
  % and tests their variance using Aly's statistic.
  %
  % vartest(samp1,samp2,N)
  % samp1, samp2 are arrays of data to be compared
  % N is the number of times to permute
  %

  % To be run before executing the program
  clear all;
  % samp1=[129,123,126,128.5,121];
  % samp2=[153,154,155,156,158];
  % To run program:
  % vartest(samp1,samp2,6400)

  %

  function [p]=vartest(samp1,samp2,N)
```
n=length(samp1);
m=length(samp2);
diff1=diff(samp1); % get difference vector of samp1
diff2=diff(samp2); % get difference vector of samp2
aly0=aly(diff1); % compute original test statistic
cnt=0;

samp=zeros(1,n);

for i=1:N
    samp=choose(diff1,diff2); % finds sampled array
    aly1=aly(samp); % compute aly statistic of sample
    if aly1<=aly0
        cnt=cnt+1;
    end
end
p=cnt/N; % p-value

% this function finds the difference vector used in aly's calculation
function [d]=diff(samp)
    s=sort(samp);
    n=length(samp);
    d=zeros(1,n-1);

    for i=1:(n-1)
        d(i)=s(i+1)-s(i);
    end

% this function computes the aly statistic with the difference vector
function [aly]=aly(diff)
    aly=0;
    n=length(diff);
    m=n+1;
    for i=1:n
        aly=aly+(i*(m-i)*diff(i));
    end

% this function chooses between the ith element if the to difference functions.
function [c]=choose(samp1,samp2)
    rand('state',sum(100*clock));
    n=length(samp1);
    c=zeros(1,n);
    for i=1:n
        if rand<.5
            c(i)=samp1(i);
        else
            c(i)=samp2(i);
        end
    end
R

diff=function(samp){
  s=sort(samp)
  l= length(samp)
  d=1:(l-1)
  for(k in 2:l){
    d[k-1]=s[k]-s[k-1]
  }
  return(d)
}

aly=function(samp){
  stat=0
  l=length(samp)
  for (k in 1:l)
    stat=stat+k*(l+1-k)*samp[k]
  return(stat)
}

vartest=function(samp1,samp2, NMonte){
  d1=diff(samp1)
  d2=diff(samp2)
  l=length(d1)
  stat0=aly(d1)
  pd=d1
  cnt=0
  for(j in 1:NMonte){
    r=rbinom(l,1,.5)
    for (k in 1:l)pd[k]=ifelse(r[k],d1[k],d2[k])
    if (aly(pd)>=stat0)cnt=cnt+1
  }
  return(cnt/NMonte) #one-sided p-value
}

x1 = c(129, 123, 126, 128.5, 121)
y1 = c(153, 154, 155, 156, 158)

vartest(x1,y1,1600)

EViews test of Equality of Variance between series

cREATE U 1 5
series x1
x1.fill 129, 123, 126, 128.5, 121
series y1
y1.fill 153, 154, 155, 156, 158
Chapter 5

Combining data from several samples

#Combine and weight samples to obtain a CI for population median

first = c(3.87, 4.48, 5.00, 3.60, 3.89, 2.50, 4.72, 3.15, 2.94, 4.65, 4.59, 3.60, 3.67, 3.16)
second = c(8.67, 2.64, 4.01, 4.31, 2.75, 2.44, 2.95, 7.86, 3.36, 6.97, 1.97, 5.31)

#record group sizes
n1 = length(first)
n2 = length(second)
n = n1 + n2

#record standard deviations
s1 = sqrt(var(first))
s2 = sqrt(var(second))

ratio = (n1/s1)/(n1/s1 + n2/s2)

#set number of bootstrap samples
N = 100

stat = numeric(N)  #create a vector in which to store the results
#the elements of the vector will be numbered from 1 to N

#Set up a loop to generate a series of bootstrap samples
for (i in 1:N){
    choice = sort(runif(n,0,1))
    cnt = 0
    while (choice[cnt+1]<ratio) cnt = cnt + 1
    #bootstrap sample counterparts to observed samples are denoted with "B"
    firstB = sample(first, cnt, replace=T)
    secondB = sample(second, n-cnt, replace=T)
    stat[i] = median(c(firstB, secondB))
}

quantile(stat,c(.05,.95))

Comparing Samples From Two Populations

For C++, R, and Resampling Methods, comparing samples from two populations is merely a question of embedding a second loop inside the original loop of Section 3.4.2. For example, in R

for (k in 1:MC){
stat=0
for (j in 1:blocks) {
    D = sample (A[j],n)
    stat = stat + sum(D)
}
if (stat <= sumorig) cnt = cnt + 1
}
cnt/MC  # pvalue

With Stata, one needs to enter the block variable along with the other data, then make use of the stratified permute command as follows:

<table>
<thead>
<tr>
<th>block</th>
<th>regimen</th>
<th>wtloss</th>
</tr>
</thead>
<tbody>
<tr>
<td>man</td>
<td>diet</td>
<td>50</td>
</tr>
<tr>
<td>man</td>
<td>diet</td>
<td>25</td>
</tr>
<tr>
<td>man</td>
<td>diet</td>
<td>30</td>
</tr>
<tr>
<td>man</td>
<td>exer</td>
<td>22</td>
</tr>
<tr>
<td>man</td>
<td>exer</td>
<td>34</td>
</tr>
<tr>
<td>man</td>
<td>exer</td>
<td>28</td>
</tr>
<tr>
<td>wom</td>
<td>diet</td>
<td>30</td>
</tr>
<tr>
<td>wom</td>
<td>diet</td>
<td>20</td>
</tr>
<tr>
<td>wom</td>
<td>exer</td>
<td>28</td>
</tr>
<tr>
<td>wom</td>
<td>exer</td>
<td>25</td>
</tr>
<tr>
<td>wom</td>
<td>exer</td>
<td>24</td>
</tr>
</tbody>
</table>

. encode regimen, gen(treat)
. permute wtloss "sum wtloss if treat==1" sum=r(sum), reps(400) strata(block) nowarn

K-Unordered Samples

C++

// set global variables
// n[0]=0, n[1]=n1, n[2]=n1+n2, ..., n[columns+1]=#observations=N
int columns, n[columns+1], N;
float data[N];

// compute statistic for original observations
float f2Orig(float *data) {
    float Bsum = 0, sum, temp;
    int k, m = 0;
    for (int j = 0; j < columns; ++j) {
        sum = 0;
        for (int i = n[j]; i < n[j+1]; ++i) sum += *(data+i);
Bsum=Bsum +sum*sum;

return (Bsum);
}

//select observations and compute statistic in a single pass
float f2(float X){
    float Bsum =0, sum, temp;
    int k, m=0;
    for (int j=1; j<columns; ++j){
        sum = 0;
        for (int i=n[j]; i< n[j+1]; i++){
            k=Choose (m, N–1);
            temp = X[k]; X[k]= X[m]; X[m]= temp;
            sum =sum +temp;
            m++;
        }
        Bsum=Bsum +sum*sum;
    }
    return (Bsum);
}

S = f2Orig(data);
for (int i =0; i < MC; ++i)if (f2(data) >=S) cnt++;
float p=float(cnt)/MC;
cout << p << endl;

R

F1=function(size,data){
#size is a vector containing the sample sizes
#data is a vector containing all the data in the same order as the sample sizes
    stat=0
    start=1
grandMean = mean(data)
for (i in 1:length(size)){
    groupMean = mean(data[seq(from = start, length = size[i])])
    stat = stat + abs(groupMean - grandMean)
    start = start + size[i]
}
return(stat)
}

We use this function repeatedly in the following R program:

# One-way analysis of unordered data via a Monte Carlo
data = c(FastGro, NewGro, WunderGro)
size = c(4,6,5,4)

f1=F1(size,data)

#Number MC of simulations determines precision of p-value
MC = 1600
cnt = 0
for (i in 1:MC){
    pdata = sample (data)
    f1p=F1(size,pdata)
    # counting number of rearrangements for which F1 greater than or equal to original
    if (f1 <= f1p) cnt=cnt+1
}
cnt/N

---

**Stata**

*** program to calculate F2 due to Lynn Markham

capture program drop f2test

program define f2test, rclass

tempvar xij sq sumxj obsnum f2 bcount

sort brand
qui by brand: gen `obsnum'=_n
qui by brand: gen `bcount'= _N
gen `sumxj'= sum(growth), by(brand)
replace `sumxj' = . if `obsnum'>1
gen `xijsq' = (`sumxj' * `sumxj')/`bcount'
egen `f2'= sum(`xijsq')
return scalar F2=`f2'[1]
end

Calculating Pitman Correlation

R

# One-way analysis of ordered data via a Monte Carlo
doce <- c(0, 0, 0, 0, 5, 5, 5, 5, 5, 5, 20, 20, 20, 20, 80, 80, 80, 80, 80)
b breaks <- c (0, 1, 1, 2, 0, 1, 2, 3, 5, 3, 5, 7, 7, 6, 7, 8, 9, 9)
temp <- 1 + dose
logdose <- log(dose)
rho0 <- cor(logdose, breaks)
# Number N of simulations determines precision of p-value
N <- 400
cnt <- 0
for (i in 1:N) {
    D <- sample (breaks)
    rho <- cor(D, logdose)
    # counting correlations larger than original by chance
    if (rho0 <= rho) cnt<-cnt+1
}
cnt/N                                    # pvalue

Resampling Stats

`One-way analysis of ordered data via a Monte Carlo
DATA (0 0 0 0 5 5 5 5 5 5 20 20 20 20 80 80 80 80 80) dose
DATA (0 1 1 2 0 1 2 3 5 3 5 7 7 6 7 8 9 9) breaks
ADD 1 dose temp
LOG temp logdose
CORR logdose breaks rho0
`N determines precision of rho-value
COPY 400 N
REPEAT N
    SHUFFLE breaks B
    CORR logdose B rho
    SCORE rho scrboard
Is this a one- or two-sided test?

COUNT scrboard >= rho0 extremes
LET pvalue = extremes/N
PRINT pvalue

Resampling Stats for Excel

Outline the two columns you wish to shuffle. When completing the Matrix Shuffle form, specify “Shuffle within Columns.” Compute Excel’s Correl() function repeatedly.

Stata

gen logdose = log(dose+1)
permute breaks “corr logdose breaks”, teststat=r(rho)

Synchronized Permutations

The following C++ code can be used to obtain the desired tests for a 2x2 balanced design with n observations per cell. Assume data is stored in the vector X.

```c++
void Compute_Bal2 (float *X, n){
    float S1=0, S2=0, S12=0, s1=0, s2=0, s12=0, *Y;
    Y = new float[4*n];
    int begin, chng, chng2, cnt1=0, cnt2=0, cnt12=0;
    /* compute the row, column, and interaction statistics for the original sample */
    Stats2x2 (X, &S1, &S2, &S12);
    for (int i =0; i < MC; ++i){
        s1=0; s2=0; s12=0;
        for (int i=0; i< 4*n; ++i) Y[i]=X[i];
        //compute row main effect
        chng= ShuffleR (Y, &s1);
        RearrangR(Y, &s1, chng);
        //compute interaction
        chng2=ShuffleC (Y, &s2, chng);
        RearrangC (Y, &s2, chng, chng2);
        for (begin =0; begin < n; ++begin)
            s12 = s12 + *(Y + begin) + *(Y + 3*n + begin);
        //compute column main effect
        for (int i=0; i< 4*n; ++i) Y[i]=X[i];
        chng2=ShuffleC (Y, &s2,0);
        RearrangC (Y, &s2, 0, chng2);
        if (s1 >= S1)cnt1++;
        if (s2 >= S2)cnt2++;
    }
}
```
if (s12 >= S12) cnt12++;
}
float p1=float(cnt1)/MC, p2=float(cnt2)/MC, p12=float(cnt12)/MC;
cout << p1 << ' ' << p2 << ' ' << p12;
}

void Stats2x2 (float *X, float *S1, float *S2, float *S12)
{
    for (int begin =0; begin < n; ++begin){
        *S1 = *S1 + *(X+ begin) + *(X+n+begin);
        *S2 = *S2 + *(X+ begin) + *(X+2*n+begin);
        *S12 = *S12 + *(X+ begin) + *(X+3*n+begin);
    }
}

int ShuffleR (float *X, float *s1)
{
    // interchange elements in 1st column between rows
    int chng=0, z, v, i;
    float templ1, temp2;
    for (i =0; i < n; ++i){
        z=Choose (chng, 2*n-1-chng);
        if (z>n-1){
            v = z-n;
            z = Choose (chng, n-1);
            templ1 = *(X+z);
            *(X+z) = *(X+chng);
            temp2= *(X+2*n+chng+v);
            *(X+2*n+chng+v) = *(X+2*n+ chng);
            *(X+chng)= temp2;
            *(X+2*n + chng) = templ1;
            chng++;
        }
    }
    // sum contents of R1C1
    for (i =0; i < n; ++i)*s1 = *s1 + *(X+i);
    // cout << " s1=" << *s1;
    return (chng);
}

void RearrangingR (float *X, float *s1, int chng)
{
    // interchange elements in 2nd column between rows
    float templ1, temp2;
    int begin, z1, z2;
    for (begin =0; begin< chng; ++begin){
        z1=Choose (begin, n-1);
        z2=Choose (begin, n-1);
        templ1= *(X+n+z1);
        temp2= *(X+3*n+z2);
        *(X+n+z1)= *(X+ begin);
        *(X+3*n+ z2)= *(X+ 3*n+ begin);
        *(X+n+ begin)=temp2;
        *(X+3*n+ begin)=templ1;
    }
}
int ShuffleC (float *X, float *s2, int chng)
{
    //interchange elements in 1st row between columns
    int chngC=0, z, v, gap, i;
    float temp1, temp2;
    for (i =chng; i < n; ++i){
        gap = chng+chngC; // gap contains previously changed elements
        z=Choose (gap, 2*n-1-gap);
        if (z>n-1){
            v=z-n;
            z = Choose (gap, n-1);
            temp1 = *(X+z);
            *(X+z) = *(X+chng);
            temp2= *(X+ n + gap + v);
            *(X+ n + gap + v) = *(X+ n + gap);
            *(X+chng)= temp2;
            *(X+ n + gap) = temp1;
            chngC++;
        }
    }
    //sum contents of R1C1
    *s2=0;
    for (i =0; i < n; ++i)*s2 = *s2 + *(X+i);
    return (chngC);
}

void RearrangC (float *X, float *s2, int chng, int chngC)
{
    int z1, z2, begin;
    float temp1, temp2;
    for (begin =chng; begin < chngC + chng; ++begin){
        z1=Choose (begin, n-1);
        z2=Choose (begin, n-1);
        temp1= *(X+2*n+ z1);
        temp2= *(X+3*n+ z2);
        *(X+2*n+ z1)= *(X+2*n+ begin);
        *(X+3*n+ z2)= *(X+3*n+ begin);
        *(X+2*n+ begin)=temp2;
        *(X+3*n+ begin)=temp1;
    }
    for (int begin = 0; begin < n; ++begin)*s2 = *s2 + *(X+2*n+ begin);
}

Chapter 6

Computing Fisher’s Exact Test
R
To simplify the programming, assume that the smallest marginal is in the first row and the smallest cell frequency is located in the first column, and that we have the actual cell frequencies

- \( \text{data} = c(f_{11}, f_{12}, f_{21}, f_{22}) \)
- \( m = \text{data}[2] + \text{data}[4] \)
- \( n = \text{data}[1] + \text{data}[3] \)
- \( t = \text{data}[1] + \text{data}[2] \)
- \( \text{ntab} = 0 \)
- \( \text{for } (k \in 0:\text{data}[1]) \text{ ntab = ntab + comb(m, t-k)*comb(n, k)} \)
- \( \text{ntab/combin(m+n,t)} \)  

where \( \text{fact} = \text{function(n)prod (1:n)} \)

and \( \text{comb} = \text{function (n,t) fact(n)/(fact(t)*fact(n-t))} \).

In S-PLUS, we would substitute \text{choose()} for \text{comb()}.

**Chapter 8**

Bivariate Dependence

**R**

```
#Calculating a p-value via a Monte Carlo
armspan <- c(139, 140, 141, 142.5, 143.5)
height <- c(137, 138.5, 140, 141, 142)
rho0 <- cor(armspan, height)
cnt<- 0
for (i in 1:400){
  D <- sample (armspan)
  rho <- cor(D, height)
  # counting correlation larger than original by chance
  if (rho0 <= rho ) cnt<-cnt+1
}
cnt/400        #pvalue
```

**Resampling Stats**

*Calculating a p-value via a Monte Carlo*

```
DATA (139 140 141 142.5 143.5) armspan
```
DATA (137 138.5 140 141 142) height
CORR armspan height rho0
REPEAT N
   SHUFFLE height H
   CORR armspan H rho
   SCORE rho scrboard
END
COUNT scrboard>=rho0 extremes
LET pvalue = extremes/N
PRINT pvalue

**Stata**
permute armspan "corr armspan height" teststat=r(rho)

Confidence Interval

**R**
armspan <- c(139, 140, 141, 142.5, 143.5)
height <- c (137, 138.5, 140, 141, 142)
n = length(armspan)
#collect all variables in a single frame so as to sample as a unit
data=cbind(armspan,height)
#set number of bootstrap samples
N =400
stat = numeric(N) #create a vector in which to store the results
for(i in 1:N){
   ind=sample(n,n, replace=T)
   boot= data[ind,]
   stat[i]=cor(boot[,1],boot[,2])
}
quantile(stat,prob=c(0.05,0.95))

**Stata**
bootstrap “corr armspan height” r(rho), reps(1000)

**Linear Regression**
R
# obtain LAD regression coefficients and test slope to see if greater than zero
library("quantreg")
Guests = c(289,391,482,358,365,561,339,479,500,160,319,331)
Meals = c(235,355,475,275,345,522,315,399,441,158,305,225)
N=400
f = coef(rq(formula = Meals ~ Guests))
names(f)=NULL
stat0=f[2]
cnt=0
for(i in 1:N){
  guestP=sample(Guests)
  fp= coef(rq(formula = Meals ~ guestP))
  names(fp)=NULL
  if (fp[2] >= stat0)
    cnt=cnt+1
}
f

cnt/N

# obtain bootstrap confidence intervals for LAD regression coefficients
library("quantreg")
Guests = c(289,391,482,358,365,561,339,479,500,160,319,331)
Meals = c(235,355,475,275,345,522,315,399,441,158,305,225)
n = length(Guests)
data=cbind(Guests,Meals)
# set number of bootstrap samples
N =400
stat = numeric(N) # create a vector in which to store the results
for(i in 1:N){
  ind=sample(n,n, replace=T)
  guestP= data[ind,]
  fp= coef(rq(formula = Meals ~ guestP))
  stat[i]= fp[2]
}
quantile(stat,prob=c(0.05,0.95))

SAS
// code uses the wrapper method of David L. Cassell

%macro rand_gen(
  indata=_last_,
  outdata=outrand,
)
depvar=y,
numreps=1000,
seed=0);
/* Get size of input dataset into macro variable &NUMRECS */
proc sql noprint;
    select count(*) into :numrecs from &INDATA;
quit;
/* Prepare for sorting by generating random numbers */
data __temp_1;
    retain seed &SEED ; drop seed;
    set &INDATA;
    do replicate = 1 to &NUMREPS;
        call ranuni(seed,rand_dep);
        output;
    end;
run;
proc sort data=__temp_1;
by replicate rand_dep;
run;
data &OUTDATA ;
    array deplist{ &NUMRECS } _temporary_ ;
    set &INDATA(in=in_orig)
        __temp_1(drop=rand_dep);
    if in_orig then do;
        replicate=0;
        deplist{ _n_ } = &DEPVAR ;
    end;
    else &DEPVAR = deplist{ 1+ mod(_n_,&NUMRECS) };
run;
%mend rand_gen;

%rand_gen(indata=nudata,outdata=outrand,
    depvar=Meals,numreps=1600,seed=12345678)
data nudata;
input Guests Meals;
datalines;
289 235
391 355
482 475
358 275
365 345
561 522
339 315
479 399
500 441
160 158
319 305
331 225
;

proc glm data=outrand noprint outstat=outstat1;
by replicate;
model Meals = Guests;
run;

%rand_anl(randdata=outstat1,
where=_source_='Guests' and _Type_='SS3',
testprob=prob,testlabel=Model F test)

%macro rand_anl(
   randdata=outrand,
   where=,
   testprob=probf,
   testlabel=F test,);
   data _null_; 
   retain pvalue numsig numtot 0; 
   set &RANDDATA end=endofile; 
   %if "&WHERE" ne "" 
   %then where &WHERE %str(;) ;
   if Replicate=0 then pvalue = &TESTPROB ;
   else do;
      numtot+1;
      numsig + ( &TESTPROB < pvalue ); 
   end;
if endofile then do;
    ratio = numsig/numtot;
    put "Randomization test for &TESTLABEL ");
    %if "&WHERE" ne "" %then "where &WHERE";
    " has significance level of ";
    ratio 6.4 ;
end;
run;
%mend rand_anl;

Stata
permute Meals "regress Meals Guests" _b, reps(400) left
Resampling Methods
A Practical Guide to Data Analysis
Good, P.I.
2006, XX, 218 p., Hardcover
ISBN: 978-0-8176-4386-7
A product of Birkhäuser Basel