"monte": When is n Sufficiently Large?

Jeffrey H. Gove*
Research Forester
USDA Forest Service
Northern Research Station
271 Mast Road
Durham, New Hampshire 03824 USA
e-mail: jgove@fs.fed.us or e-mail: jhgove@unh.edu

Wednesday 1st August, 2012
4:02pm

Contents

1 Introduction

In designing surveys or even the sampling methods themselves, one is often presented with the question: How many samples should be taken in practice? The answer is a function of the variability of the population and the time it takes to execute the respective sampling technique on the population. Barrett and Goldsmith (1976) addressed the question "When is n sufficiently large" in their Monte Carlo (MC) analysis, based on normal theory confidence interval convergence under the Central Limit Theorem (CLT). Given a population of individuals, drawing repeated samples in the Monte Carlo sense and monitoring the percentage of times the associated confidence intervals catch the population mean for different sample sizes (n) is certainly a useful and instructive way to address this question. In what follows, we use this concept to look at sample size issues for populations of sample (grid) points in sampling surface "sampSurf" objects (Gove, 2011b). More

*Phone: (603) 868-7667; Fax: (603) 868-7604.

information on this approach can be found, for example in Barrett and Nutt (1979). Recently, van Buuren (2012, p. 47) has also advocated this approach for assessing the performance of various multiple imputation methods.

The "monte" class allows for repeated sampling from objects of class "sampSurf" (Gove, 2011b) in the Monte Carlo sense with performance results calculated for, e.g., the percentage of times the sample confidence intervals caught the population mean. Both normal theory and bootstrap intervals can be requested in the analysis. Requesting different samples sizes (e.g., $n = 10, 20, 50, 100, \ldots$) in the simulation allows the comparison of convergence to the nominal confidence level desired (e.g., the 95% level). This type of analysis then provides a method for assessing the performance of different sampling methods on a given population, or for assessing how a given method performs across disparate populations with various degrees of inherent population variability. In the former, we wish to see how methods compare with each other, in the latter, we perhaps wish to assess how consistently a new method performs across different types of populations of standing trees or downed logs.

While currently part of the R package `sampSurf` (Gove, 2012a,b), the "monte" concept is really more of a subpackage within `sampSurf`, and can be used in its current form for more general populations than those from "sampSurf" objects. Therefore, the classes detailed here may be separated from the `sampSurf` package in the future and be located in a more general sampling package, with perhaps only "sampSurf" extensions residing in `sampSurf`.[1]

The following documents the class structure within the `SampSurf` package for objects related to confidence interval assessment, and provides some simple examples of their use. It is assumed that the reader has some familiarity with the workings of the `sampSurf` package,[2] and of course, with R itself.

2 "monte" Class Structure Overview

There are several classes associated with the Monte Carlo CLT simulations implemented here. These are...

- "montePop:" This class stores the population that the MC samples will be drawn from for confidence interval estimation.
- "monteSample:" This class holds the means, confidence intervals and other information from the individual MC samples drawn from the population. There are subclasses for normal theory ("monteNTSample") and bootstrap ("monteBSSample") methods.

[1] In other words, "monte" support will always be available for "sampSurf" objects, but don't count on the code residing within the package name space (other than as an imported required package for `sampSurf`) in the future.

[2] For a very brief introduction, see: http://sampsurf.r-forge.r-project.org/.

- "monte:" The class the keeps tract of objects of the above classes for a give set of MC runs. There can be normal theory or bootstrap components, or both.

In what follows, each class will be detailed, including the class structures and generics used to create objects of the respective classes.

3 The "montePop" Class

Each object of class "monte" must have a population of values that form the basis for repeated sampling in the Monte Carlo sense. The "montePop" class holds the information necessary for such populations in the context of "monte" objects. The class is defined as follows...

```
R> showClass('montePop')
```

```
Class "montePop" [package "sampSurf"]

Slots:

Name:           mean            var         stDev              N          total
Class:       numeric        numeric       numeric        numeric        numeric

Name:        popVals zeroTruncated              n            fpc        varMean
Class:       numeric        logical       numeric        numeric        numeric

Name:          stErr    description
Class:       numeric      character
```

- *mean*: The population mean: $\mu = \frac{1}{N}\sum_{i=1}^{N} y_i$.
- *var*: The population variance: $\sigma^2 = \frac{1}{N-1}\sum_{i=1}^{N}(y_i - \mu)^2$.
- *stDev*: The population standard deviation: $\sigma = \sqrt{\sigma^2}$
- N: The number of observations in the population—the population size (N).
- *total*: The total for whatever attribute the population is concerned with: $\tau_y = \sum_{i=1}^{N} y_i$.
- *popVals*: A numeric vector containing the values, $y_i, i = 1, \ldots, N$, in the population.
- *zeroTruncated*: A logical scalar: `TRUE` if the population is zero-truncated (no zeros); `FALSE` otherwise. This was added to deal with "sampSurf" objects, which can be largely zero-inflated

in all of the background grid cells where no inclusion zones exist, depending on how the tract size, population size, sampling design, etc., are chosen.

Note: The following four slots can either contain the contents as described below, or be `NA` *if no sample size information was provided...*

- n: A numeric vector listing the different sample sizes that will be drawn from this object. If we will be drawning samples of size $n = 10, 20, 30$, then this would hold `c(10,20,30)`.
- *fpc*: The finite population correction factors for each sample size `n`. The correction is: $f_c = (N - n)/N$.
- *varMean*: The population variance of the mean, which is sample size dependent; *viz.*, $\sigma^2_{\bar{y}} = \frac{\sigma^2}{n} \times f_c$.
- *stErr*: The population standard error of the mean: $\sigma_{\bar{y}} = \sqrt{\sigma^2_{\bar{y}}}$.
- *description*: Some descriptive text about the object.

Note that each object of class "montePop" is designed to hold one, and only one population.

3.1 Object creation

There is a constructor generic to create objects of class "montePop". While it is simple enough to create objects using `new`, it is recommended to use the constructor to minimize the chances of creating an invalid object. The constructor has the same name as the class and can create an object from an **R** numeric vector, or from a "sampSurf" object...

```
R> x = rnorm(100)
R> x.mp = montePop(x)
R> x.mp
```

```
Population...
  Mean = 0.10287108
  Variance = 0.79935093
  Standard Deviation = 0.89406428
  Total = 10.287108
  Size (N) = 100
  Zero-truncated = FALSE
```

The result of the above run is a "montePop" population object with the slots defined above assigned when the signature object of the constructor is of class "numeric" (vector).

In the following, we generate a population object from a "sampSurf" object...

```
R> smTract = Tract(c(x=30,y=30), cellSize=0.5)
R> smbuffTr = bufferedTract(8,smTract)
R> agauge = angleGauge(6)
R> SS.hps = sampSurf(10, smbuffTr, 'horizontalPointIZ', angleGauge=agauge,
+                    estimate='volume')


Number of trees in collection = 10
Heaping tree: 1,2,3,4,5,6,7,8,9,10,


R> (hps.pop = montePop(SS.hps, zeroTruncate = TRUE, n = c(10,20,30)))


Population...
  Mean = 10.408513
  Variance = 32.939488
  Standard Deviation = 5.7392933
  Total = 16133.195
  Size (N) = 1550
  Zero-truncated = TRUE
  Sample sizes (n) = 10, 20, 30
  Finite population corrections = 0.9935, 0.9871, 0.9806
  Variance of the mean = 3.2726975, 1.6257231, 1.0767317
  Standard error of the mean = 1.8090598, 1.2750385, 1.0376568
```

In this example we have specified that samples of size $n = (10, 20, 30)$ will eventually be drawn from the population. This prompts the constructor to calculate the finite population correction, population variance of the mean and standard error of the mean associated with these sample sizes as is demonstrated in the summary of the object. This option is not one that would probably be used on its own just to create a "montePop" object, as it would limit the eventual use of the contents to these sample sizes. Where it becomes very useful is in the "monte" object construction, where the sample sizes are an intrinsic component of a given Monte Carlo experiment.

One very important point to keep in mind when using the zero-truncated population and subsequent MC sampling routine is the following. The sampling surface method takes the mean of the surface estimates *including the background cells*, which have zero value, to compute the estimate. When we truncate the zeros in the background, we now will have a population mean that will be larger—perhaps substantially so, depending upon the inclusion zone coverage—than the unbiased estimate we get from running the `sampSurf` method above. For example, the horizontal point sampling population we just created has the following statistics (see also Figure 1)...

```
R> summary(SS.hps)


Object of class: sampSurf
------------------------------------------------------------
sampling surface object
------------------------------------------------------------

Inclusion zone objects: horizontalPointIZ
Measurement units = metric
Number of trees = 10
True tree volume = 4.4827615 cubic meters
True tree basal area = 0.59545644 square meters
True tree surface area = 66.713437 square meters
True tree biomass = NA
True tree carbon = NA

Estimate attribute: volume
Surface statistics...
  mean = 4.4814431
  bias = -0.0013183558
  bias percent = -0.029409458
  sum = 16133.195
  var = 40.746275
  st. dev. = 6.3832809
  cv % = 142.43807
  surface max = 26.413795
  total # grid cells = 3600
  grid cell resolution (x & y) = 0.5 meters
  # of background cells (zero) = 2050
  # of inclusion zone cells = 1550
```

Note that the standing tree volume on the tract is 4.483 m^3, and the unbiased estimate given by the mean over all grid cells of the sampling surface is 4.481. However, we see in the "montePop" object that the population mean is 10.41. The difference is obvious, and the source of the difference is, of course, due to zero-truncation. Finally, note that the size of the population in the "montePop" object is $N = 1550$, which is the number of cells covered by inclusion zones in the above summary. In contrast, the entire sampling surface grid is composed of 3600 grid cells.

3.2 Plotting the object

Currently only histograms are supported for "montePop" objects. The command is...

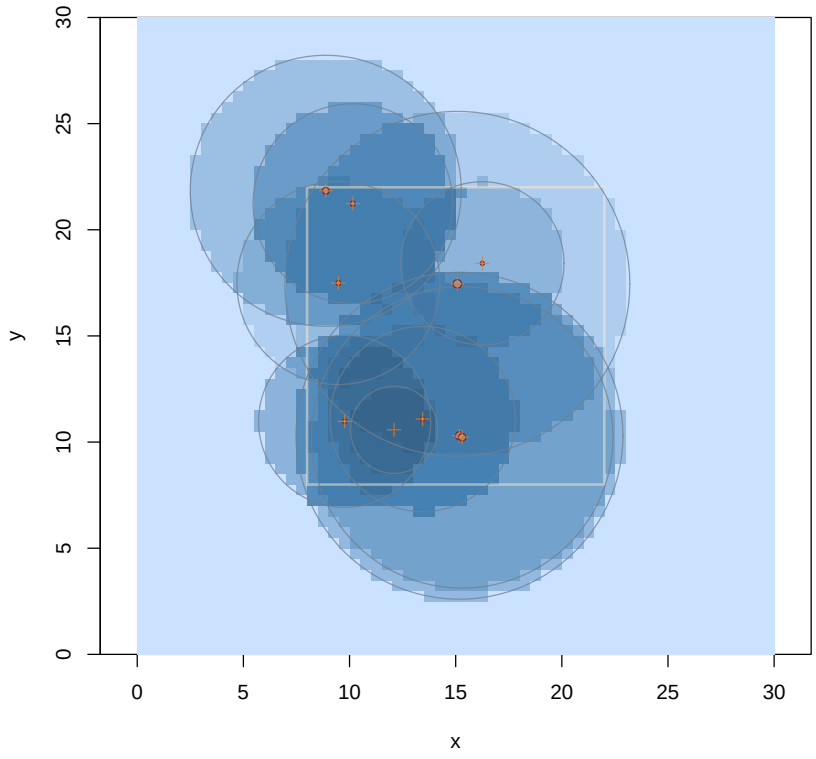

Figure 1: The horizontal point sampling "sampSurf" representation.

```
R> hist(hps.pop)
```

4 The "monteSample" Class

This is a general class for holding information on the MC samples drawn from a "montePop" population object. It is a virtual class; therefore, you must use one of the subclasses that have been tailored to normal theory or bootstrap resampling described below—or create your own subclass for a new application.

```
R> showClass('monteSample')
```

```
Virtual Class "monteSample" [package "sampSurf"]
```

```
Slots:

Name:   mcSamples          n      alpha    replace    ranSeed        fpc
Class:    numeric    numeric    numeric    logical    numeric    numeric

Name:       means       vars     stDevs   varMeans     stErrs   lowerCIs
Class: data.frame data.frame data.frame data.frame data.frame data.frame

Name:    upperCIs     caught  caughtPct      stats
Class: data.frame data.frame    numeric data.frame

Known Subclasses: "monteNTSample", "monteBSSample"
```

- *mcSamples*: A scalar numeric specifying the number of Monte Carlo samples drawn from the population.
- n: A numeric vector listing the different sample sizes recorded in the object that have been drawn from a "montePop" population object. So, if we have drawn samples of size $n = 10, 20, 30$, then this would hold `c(10,20,30)`. The associated names should always be of the form `c('n.10','n.20','n.30')`.
- *alpha*: The two-tailed alpha level for which confidence intervals have been calculated. I.e., for the 95% confidence level ($\alpha = 0.05$) `alpha = 0.05`.
- *replace*: `TRUE` if the samples have been drawn from the population with replacement, `FALSE` otherwise.
- *ranSeed*: The random number seed as a numeric vector. Please see the **R** documentation on `.Random.seed` for information on the format of this slot. Note that it is *not* a simple scalar integer "seed", but a vector of integers containing the state of the random number generator at the beginning of the simulations.
- *fpc*: The finite population correction factors for each sample size `n`. The correction is: $f_c = (N - n)/N$.
- *means*: A data frame with `mcSamples` rows, and one column for each of the sample sizes in the `n` slot of the object. What is stored here depends on the subclass object type, so please see the definitions below for these slots.

  Note: The next six slots have the same dimensions as the `means` slot.

- *vars*: Contains the individual sample variances for each sample for both "monteNTSample" and "monteBSSample" subclasses: $s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(y_i - \bar{y})^2$.
- *stDevs*: Contains the individual sample standard deviations for each sample for both "monteNTSample" and "monteBSSample" subclasses: $s = \sqrt{s^2}$.

- *varMeans*: Contains the individual sample variance of the mean for each sample for both "monteNTSample" and "monteBSSample" subclasses: $s^2_{\bar{y}} = \frac{s^2}{n} \times f_c$.
- *stErrs*: Contains the individual standard errors for each sample for both "monteNTSample" and "monteBSSample" subclasses: $s_{\bar{y}} = \sqrt{s^2_{\bar{y}}}$.
- *lowerCIs*: Contains the individual lower limit for the confidence intervals. This is defined differently for the "monteNTSample" and "monteBSSample" subclasses.
- *upperCIs*: Contains the individual upper limit for the confidence intervals. This is defined differently for the "monteNTSample" and "monteBSSample" subclasses.
- *caught*: Contains a flag where `TRUE` means the confidence interval caught the population mean and `FALSE` means it failed to catch the population mean. Taking column sums, therefore (since `TRUE == 1` and `FALSE == 0`) will give the number of intervals that caught the population mean for each sample size. This is used to calculate the next slot below.
- *caughtPct*: The percentage of times the confidence intervals caught the population mean as calculated from the data frame in the `caught` slot of the object.
- *stats*: A summary data frame with rows as the *average* of each column (i.e., over all MC samples) from the information in the data frames above (`means`, `vars`, `stDevs`, `varMeans`, `stErrs`, `lowerCIs`, and `upperCIs`). The interpretation of some of the rows depends on the subclass object as has been mentioned above.

4.1 The "monteNTSample" class

This class holds information for classic normal theory confidence intervals under simple random sampling. It adds only one slot to the "monteSample" superclass, *viz.*, `t.values`. Some of the other slot definitions that depend on the type of intervals are also defined below for this class.

- *t.values*: The $t^{1-\alpha/2}_{n-1}$ Student's t values for each sample size `n` with two-tailed α-level `alpha`.
- *means*: The data frame contains the individual means for all `mcSamples` by `length(n)` samples drawn from the population. Taking column means gives the overall mean for each of the sample sizes. The sample mean is: $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$.
- *lowerCIs*: This is the usual normal theory lower limit for each sample: $\bar{y} - t^{1-\alpha/2}_{n-1} s_{\bar{y}}$, where t is Student's t-value and $s_{\bar{y}}$ is the standard error of the mean for the sample.
- *upperCIs*: This is the usual normal theory upper limit for each sample: $\bar{y} + t^{1-\alpha/2}_{n-1} s_{\bar{y}}$, where t is Student's t-value and $s_{\bar{y}}$ is the standard error of the mean for the sample.

The `stats` slot averages over all Monte Carlo samples and therefore has the usual interpretation for each row in the data frame.

4.1.1 Object creation

The constructor for objects of this class has the same name as the class. Below we create an object from a sampling surface for several sample sizes, n, and a small number of MC samples for illustration.

Please note that while we can create objects in the following manner, it is preferable to use the method `monte` to do so as discussed later. The reason for this is that objects of class "monte" store everything needed to reconstruct the samples, including the population they came from, which is not present in this class of MC results.

```
R> hps.nts = monteNTSample(hps.pop, n = c(10,20,30), mcSamples=100)
R> hps.nts
```

```
Number of Monte Carlo samples = 100
Sample sizes: n = 10, 20, 30
Sample summary statistics (mean values)...
              n.10       n.20       n.30
mean    10.4378798 10.3603330 10.5292852
var     32.7134852 31.5283643 32.1753484
stDev    5.5537928  5.5130718  5.6252926
VarMean  3.2502430  1.5560773  1.0517533
stErr    1.7505890  1.2247812  1.0170456
lowerCI  6.4777725  7.7968364  8.4491934
upperCI 14.3979871 12.9238295 12.6093771

Percentage of confidence intervals (95%) that caught the population mean...
n.10 n.20 n.30
  91   93   96
```

4.2 The "monteBSSample" class

This subclass of "monteSample" handles bootstrap confidence intervals in the MC setting. At present, only bootstrap "BCa" confidence intervals are calculated. The general idea is as follows. First, draw a sample from the population of interest, just like in the normal theory case. Then run R bootstrap resample replicates and calculate the mean and confidence interval endpoints from the distribution of bootstrap means of the replicates. This is repeated for each MC sample and for each sample size. The bootstrap intervals provide a nonparametric alternative to the normal theory intervals and may be more valid when the distribution of sample means is non-Gaussian. Thus, the bootstrap is really a nested—or second-stage—set of Monte Carlo iterations for each first-stage MC iteration and sample size.

The class adds two new slots to the "monteSample" superclass as shown below. In addition, a few of the other slot definitions that depend on the type of intervals are also defined below for this class.

- *degenerate*: It may happen that, especially for small n, some of the samples drawn from the population can be degenerate (all the same value). When this happens, all of the bootstrap resamples will also be degenerate, and confidence interval estimation is impossible since it is based on the distribution of the bootstrap sample means. This slot is a numeric vector with the number of degenerate samples for each sample size in the `n` slot of the object.
- R: The number of bootstrap sample replications.
- *means*: The data frame contains the overall bootstrap sample means for each of the `mcSamples` by `length(n)` samples drawn from the population. The overall bootstrap sample mean is defined here as the mean of the R individual (second-stage) bootstrap sample means for each case. Taking column means gives the overall mean for each of the sample sizes.
- *lowerCIs*: This is the lower "BCa" confidence interval endpoint for the $1 - \alpha/2$ confidence level. It is calculated from the distribution of bootstrap sample means that is created in bootstrap sampling for each MC sample and sample size, n.
- *upperCIs*: This is the upper "BCa" confidence interval endpoint for the $1 - \alpha/2$ confidence level. It is calculated from the distribution of bootstrap sample means that is created in bootstrap sampling for each MC sample and sample size, n.

The `stats` slot again averages over all Monte Carlo samples in each column of the data frames as defined above. Note, however, that only the `means`, `lowerCIs` and `upperCIs` have a meaning that differs from those of class "monteNTSample". Therefore, the other rows in `stats` contain the usual Monte Carlo averages, not Monte Carlo averages based on bootstrapping results.

4.2.1 Object creation

As with the normal theory subclass, objects can be generated with a constructor of the same name. It is, however, preferable to use the `monte` constructor in general.

```
R> hps.bss = monteBSSample(hps.pop, n = c(10,20,30), mcSamples=100, R=50)
R> hps.bss
```

```
Number of bootstrap samples = 50
Number of Monte Carlo samples = 100
Sample sizes: n = 10, 20, 30
```

```
Sample summary statistics (mean values)...
              n.10       n.20       n.30
mean    10.2799493 10.4287504 10.4488183
var     31.6262372 32.2541703 33.6917917
stDev    5.4416925  5.5943347  5.7552639
varMean  3.1422197  1.5918994  1.1013231
stErr    1.7152543  1.2428345  1.0405443
lowerCI  7.1002891  8.1677508  8.4761784
upperCI 13.7873644 13.0531182 12.5869331

Percentage of confidence intervals (95%) that caught the population mean...
n.10 n.20 n.30
  91   95   94
```

The example uses a much smaller number of bootstrap iterations than is normally recommended in practice, just for illustration here.

4.2.2 Plotting the object

Currently only histograms are supported for "monteSample" subclass objects. The command is, e.g.,...

```
R> hist(hps.bss)
```

The histograms will be illustrated in the next section using objects of class "monte".

5 The "monte" Class

We have described all of the component classes that go into a possible set of Monte Carlo samples from a population of interest. The "monte" class combines the above class structures into slots in its structure. The constructor for class "monte" constructs the individual objects for the classes discussed above and then constructs the "monte" object.

The class structure is shown as follows...

```
R> showClass('monte')
```

```
Class "monte" [package "sampSurf"]

Slots:

Name:                  pop            estimate           NTsamples
Class:            montePop           character monteNTSampleOrNULL

Name:            BSsamples         description
Class: monteBSSampleOrNULL           character
```

By now, most of what follows should be self-explanatory.

- *pop*: An object of class "montePop".
- *estimate*: In the case of "sampSurf" objects, this is the attribute for which the surface has been estimated.
- *NTsamples*: An object of class "monteNTSample", or `NULL` if non-existent.
- *BSsamples*: An object of class "monteBSSample", or `NULL` if non-existent.
- *description*: Some descriptive text about the object.

One thing to note is that the object can contain either normal theory or bootstrap information or both. This will be illustrated below in the constructor method.

5.1 Object creation

Currently there are three methods for constructing objects of class "monte". The signature argument in each case wants a "population" specification from which to draw the repeated samples. The signature argument can be either a "numeric" vector, a "montePop" population object, or a "sampSurf" object. In all cases, the method for the "montePop" object is ultimately called to do the work, the other two are just wrappers to generate "montePop" objects from the population that was specified in the signature argument. Here we demonstrate the method that takes a signature argument of class "sampSurf". Information on other arguments shown below and available in general in the `monte` generic is found in the help pages (methods?monte).

```
R> mo = monte(SS.hps, zeroTruncate = TRUE, n = c(10, 20, 30, 50), mcSamples=200,
+             R=120)
R> mo
```

```
Estimate attribute = volume

Population...
  Mean = 10.408513
  Variance = 32.939488
  Standard Deviation = 5.7392933
  Total = 16133.195
  Size (N) = 1550
  Zero-truncated = TRUE
  Sample sizes (n) = 10, 20, 30, 50
  Finite population corrections = 0.9935, 0.9871, 0.9806, 0.9677
  Variance of the mean = 3.2726975, 1.6257231, 1.0767317, 0.63753848
  Standard error of the mean = 1.8090598, 1.2750385, 1.0376568, 0.79846007

Normal theory results...
Number of Monte Carlo samples = 200
Sample sizes: n = 10, 20, 30, 50
Sample summary statistics (mean values)...
              n.10       n.20       n.30        n.50
mean    10.4373706 10.3893004 10.4627020 10.33724911
var     34.7620274 32.9272407 31.9264719 32.92123281
stDev    5.7228927  5.6561952  5.6023021  5.70380109
VarMean  3.4537756  1.6251187  1.0436180  0.63718515
stErr    1.8038903  1.2565774  1.0128890  0.79352233
lowerCI  6.3566873  7.7592536  8.3911115  8.74260629
upperCI 14.5180539 13.0193472 12.5342925 11.93189193

Percentage of confidence intervals (95%) that caught the population mean...
n.10 n.20 n.30 n.50
93.0 94.5 96.0 98.0

Bootstrap results...
Number of bootstrap samples = 120
Number of Monte Carlo samples = 200
Sample sizes: n = 10, 20, 30, 50
Sample summary statistics (mean values)...
              n.10       n.20       n.30        n.50
mean    10.3085479 10.4660787 10.4585813 10.44576810
var     33.3654434 33.4517862 33.4042704 32.88356662
stDev    5.5988492  5.6899283  5.7336186  5.70302579
varMean  3.3150182  1.6510075  1.0919245  0.63645613
stErr    1.7647910  1.2640716  1.0366308  0.79341447
lowerCI  7.2593559  8.2000600  8.5061707  8.93895808
upperCI 14.1511558 13.2485723 12.7238695 12.19924689
```

```
Percentage of confidence intervals (95%) that caught the population mean...
n.10 n.20 n.30 n.50
91.5 91.0 95.0 92.0
```

The output from the above summary is fairly lengthy, as it includes the summaries from the individual objects and both normal theory and bootstrap intervals were calculated (`type = 'both'` by default). Note that the sample size fields in the "montePop" object are present because `n` is an argument to `monte`. Please note that only a few bootstrap ($R = 120$) and Monte Carlo samples (200) have been used here for illustration, more are often recommended in practice.

5.2 Plotting the object

We can plot histograms of the different results from a "monte" object. Since it is possible to create histograms of several of the component slots, you must specify which one you want to display. The options are `type = c('normal', 'bootstrap', 'population')`...

```
R> hist(mo, type='boot')
```

In Figure 2 all sample sizes, n, that were requested are displayed. In general, the formal argument (`n`) to the function accepts a subset or all (the default) of the sample sizes present in the "monte" object. This will work for both the normal theory and bootstrap histograms, and is, of course, not applicable to the population histogram.

The histograms can also be plotted independently by simply referring to the individual slot objects in the "monte" object. For example, the above histogram could also have been created using...

```
R> hist(mo@BSsamples)
```

6 Summary

The classes detailed here are intended to be used as described above. More details on the different components of "monte" and its associated classes can be found in the R help files for `sampSurf`.[3] Two helpful sources of information on the S4 paradigm for object-oriented programming within R are Chambers (2008) and Gentleman (2008). A brief introduction is also found in Gove (2012a) and the appendix of Gove (2011a).

[3] Try, for example, `class?monte` or `methods?montePop`.

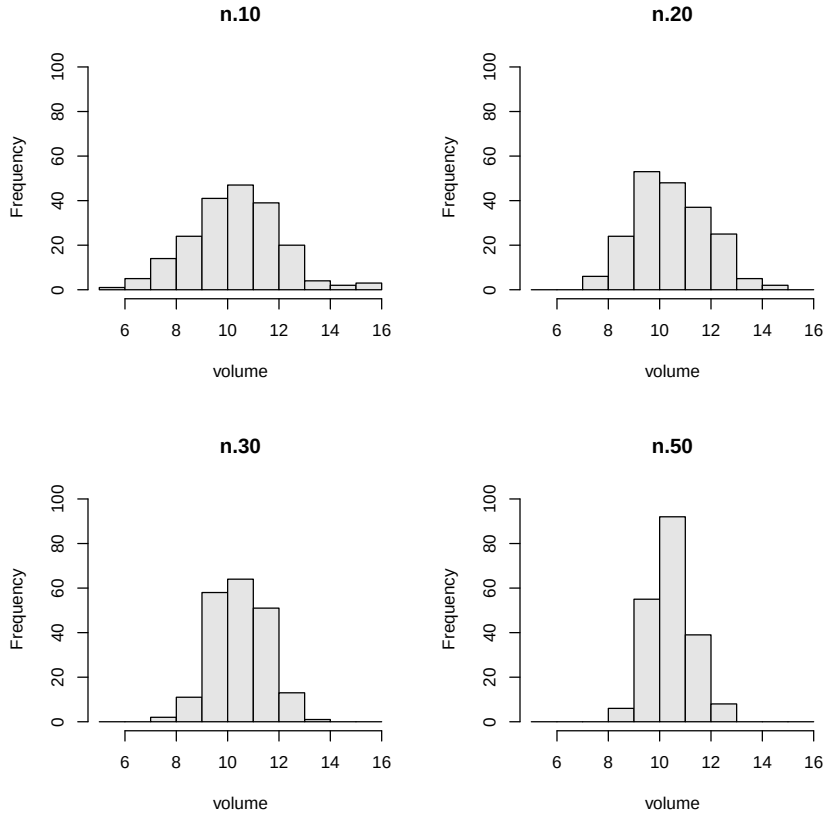

Figure 2: Histogram of a "monte" object showing the bootstrap results for each sample size.

References

J. P. Barrett and L. Goldsmith. When is n sufficiently large? *The American Statistician*, 30:67–70, 1976. 1

J. P. Barrett and M. E. Nutt. *Survey sampling in the environmental sciences: A computer approach*. COMPress, Inc., 1979. 2

J. M. Chambers. *Software for Data Analysis: Programing with R*. Springer, 2008. 15

R. Gentleman. *R Programming for Bioinformatics*. Chapman & Hall/CRC, 2008. 15

J. H. Gove. *The* ***sampSurf*** *Package Overview*, 2011a. URL http://CRAN.R-project.org/package=sampSurf. `sampSurf` package vignette. 15

J. H. Gove. *The "sampSurf" Class*, 2011b. URL http://CRAN.R-project.org/package=sampSurf. `sampSurf` package vignette. 1, 2

J. H. Gove. `sampSurf`: Sampling surface simulation for areal sampling designs in R. *Journal of Statistical Software*, 2012a. (In Preparation). 2, 15

J. H. Gove. ***sampSurf:*** *Sampling Surface Simulation for Areal Sampling Methods*, 2012b. URL http://CRAN.R-project.org/package=sampSurf. R package version 0.6-4. 2

S. van Buuren. *Flexible imputation of missing data*. Champan and Hall/CRC, 2012. 2