A User's Guide to the ssExtra Package

J. H. Gove[a]

[a] *USDA Forest Service, Northern Research Station, 271 Mast Road, Durham, NH 03824 USA*

(603) 868-7667; e-mail: jgove@fs.fed.us

Wednesday 27th May, 2020

9:29am

Contents

1 Introduction

The R *sampSurf* package (Gove, 2012b) allows the simulation of forest sampling methods by building "sampling surfaces" that show the density surface of an attribute such as volume or basal area, over a tract. The trees that populate the tract may be synthetically generated, or they can be imported into the system from actual mapped plots or stands. The same is true for downed coarse woody debris populations. There are approximately three dozen different sampling meth-

ods, including different protocols within certain methods, that can be simulated in *sampSurf*. The package itself has grown to be quite large and is relatively mature in the sense that there is more than enough present in the way of class structures and methods such that the addition of more functionality at this point has been deemed by the author to make it somewhat unwieldy in size. This is not to say that additional functionality will be completely excluded in the future, only that it is a fairly complete package and additions should be done with some thought to necessity and the possibility of creating a system that may be intimidating to potential users (including the author).

Another option is to leave *sampSurf* in its relatively stable phase, changing it only for corrections, minor extensions, or simplification of use, and to place new functionality that is not necessary to the core *sampSurf* module (package) itself in separate packages that are dependent on the *sampSurf* class and methods structure. This is the approach that was taken in the additional R *ssWavelets* package, that facilitates the application of elementary wavelet filtering to sampling surfaces (Gove, 2017b). Anyone desiring to apply wavelets in a manner similar to what was done in Gove (2017a), can simply use this add-on package; those who want just the base *sampSurf* functionality can ignore this and use the standard classes and methods defined in *sampSurf*.

The current package, *ssExtra*, or *sampSurf* extras has been initiated to handle new additions that don't easily fall within the purview of the extant *sampSurf* class and methods structure. It also is envisioned that it can accommodate miscellaneous functions that may be written for illustration. The main additions to this package at present involve classes and methods that will facilitate double sampling strategies in *sampSurf*, including a Monte Carlo (MC) extension for repeated sampling. A few miscellaneous methods are also included; for example, methods that demonstrate how one might create a synthetic tract and population of trees to be used with *sampSurf*.

The *ssExtra* package is made available either using `library(ssExtra)` or by...

```
R> require(ssExtra, quietly = TRUE, warn.conflicts = FALSE)
```

where the last argument will suppress the message about masking other symbols (to keep the above uncluttered) and may be omitted in general. To view all *potential* conflicts, use `conflicts(detail = TRUE)`. This will automatically load the `sampSurf` package as well if it is not already loaded. Not all of the classes and methods are described in detail in this document. For more information please use R's help system on this package; i.e.,

```
R> package?ssExtra
```

This package also contains code for the new point-based Delta Method (PBDM) variance estimator, which will be discussed in more detail in § 8. The new method is described in detail by Lynch et al. (2020).

2 Creating a Tract and Tree Population

There a several routines included in *ssExtra* that might be of some guidance in creating a synthetic tree population on a tract of a given size. One can easily create a tree population, plot the trees relative to each other and look at all their attributes without creating a "tract" to hold them. However, *sampSurf* requires that the population be coupled with an enclosing area, composed of a raster surface, to hold the tree population. Please keep in mind that, in general, anything that can be done with a tree population can also be done with a population of down coarse woody debris; however, only tree populations are discussed here. In addition, if actual mapped stand data are available, *sampSurf* is fully capable of using such data. Alternatively, any combination of the two is also acceptable; for example, Gove et al. (2020) used dendrometry measurements from eastern white pine trees in different stands and matched these to synthetic spatial locations to collect them into one semi-synthetic stand, as it were.

2.1 Initializing a buffered tract

A buffered tract is one where there is an internal plot that holds the tree population, and an external buffer that is normally wide enough to allow all inclusions zones for trees near the border of the internal plot to lie completely within the full tract plus buffer region. For this reason, it sometimes takes a few calculations or a little trial and error to get the buffer region just right for a given population of trees and sampling method.

A *sampSurf* "Tract" object has extents (normally beginning at zero) and a raster cell size. A one meter square raster cell size is often convenient and provides enough coverage to produce useful results from the simulations. The *sampSurf* package provides methods for the creation of different types of tracts (Gove, 2013b), and the process is simple enough, but the following method combines just a couple steps to make the process a little simpler in general for creating "bufferedTract" objects. Note that a "bufferedTract" also requires a buffer size in addition to the extents and the raster size.

```
R> args(initTract)


function (extents = c(x = 150, y = 150), cellSize = 1, bufferWidth = 14,
    units = "metric", ...)
NULL


R> buffTr = initTract(c(x = 178, y = 178), bufferWidth = 18,
+                     description = 'Northern Harwoods buffered tract')
R> buffTr
```

```
------------------------------------------------------------
Northern Harwoods buffered tract
------------------------------------------------------------
Measurement units = metric
Area in square meters = 31684 (3.1684 hectares)

class      : bufferedTract
dimensions : 178, 178, 31684  (nrow, ncol, ncell)
resolution : 1, 1  (x, y)
extent     : 0, 178, 0, 178  (xmin, xmax, ymin, ymax)
crs        : NA
source     : memory
names      : surf
values     : 0, 0  (min, max)

Buffer width =  18


R> (plotArea = (178 - 2*18)^2/10000)   #internal area in ha


[1] 2.0164
```

The full tract area including the buffer is 31,684 m^2, or 3.1684 ha. The internal portion of the tract that will be populated by trees is therefore: $^{(178-2*18)^2}/_{10000} = 2.0164$ ha. Since *sampSurf* places a sample point at the center of each raster cell on the tract, there are also 31,684 total sample points on the tract.

2.2 Creating the tree population

As noted above, a tree population and tract to place it on with total sampling intensity defined as the number of raster cells (sample points) in the tract, are two of the necessary components for generating a sampling surface. Now we concentrate on the creating a synthetic tree population to eventually be coupled with the buffered tract created in § 2.1.

The methods described are for a mixed northern hardwood stand (or plot) with a given basal area and three-parameter Weibull diameter distribution. The specification of these parameters is enough to determine the stand diameter distribution and retrieve the number of trees and quadratic mean stand diameter through the relationship

$$B = \bar{D}_q^2 N \kappa$$

where B be the stand basal area, N is the number of trees, $\bar{D}_q$ is the quadratic mean stand diameter,

and κ converts from diameter-squared to basal area. Now, if we let (ξ, β, γ) be the three-parameter Weibull location (effectively the minimum DBH), scale and shape parameters, then it can easily be shown (Gove and Patil, 1998) that $\bar{D}_q^2 \equiv \mu_2'$, the second raw moment of a distribution in general. Applying these to the three-parameter Weibull we have...

$$\mu_2' = \beta^2 \Gamma(2/\gamma + 1) + 2\beta\xi\Gamma(1/\gamma + 1) + \xi^2$$

which gives us $\bar{D}_q = \sqrt{\mu_2'}$ in terms of only the Weibull parameters; hence, solving for the number of trees

$$N = \frac{B}{\kappa \bar{D}_q^2}$$

The above process is a simple method to generate a stand with N trees according to the desired basal area and Weibull diameter distribution model. An example follows in which we will replace (ξ, β, γ) by (a, b, c) to make programming a little simpler. In the following the main quantities are input in "English" units (since the author still thinks in such terms) and converted to metric...

```
R> sfa2smh = 1/4.356                    #ft^2/acre to m^2/ha
R> kappa = pi/(4*10000)                 #ba conversion metric
R> (B = 100*sfa2smh)                    #basal area in m^2/ha


[1] 22.956841


R> a = 4*2.54                           #location parameter in cm
R> b = 8*2.54                           #scale parameter in cm
R> c = 2                                #shape parameter
R> mu2 = b^2*gamma(2/c+1) + 2*b*a*gamma(1/c+1) + a^2     #qmsd^2
R> (Dbarq = sqrt(mu2))                                   #qmsd


[1] 29.699381


R> (N = B/(kappa*mu2))                                   #number of trees/ha


[1] 331.38089
```

2.2.1 `drawTreePop` for northern hardwoods

The *ssExtra* package has a method called **`drawTreePop`** that is setup to do the above calculations for a northern hardwoods tract. We can duplicate the above with the following...

```
R> args(drawTreePop)


function (tract = buffTr, solidTypes = c(1.5, 3), topDiams = c(0,
    0), B = 80, hgt.sd = 6, a = 4, b = 8, c = 2, inhibitDist = 3,
    showPlot = TRUE, startSeed = 144, runQuiet = FALSE, ...)
NULL


R> tp = drawTreePop(buffTr, B = 100, hgt.sd = 8, a = 4, b = 8, c = 2,
+                   startSeed = 355, main = NA)


Input specs...
  Number of trees/acre = 134
  Quadratic msd = 11.69267 in
  Basal area/acre = 100
  Weibull shape = 2
  Weibull scale = 8 in
  Weibull shift = 4 in
Output specs...
  units = metric
  Number of trees/ha = 331
  Quadratic msd = 29.699381 cm
  Basal area/ha = 22.956841
  Weibull scale = 20.32 cm
  Weibull shift = 10.16 cm

  Totals for the tract...
  --Tract area (inside the buffer) = 2.0164
  --Total N for above area = 667.4284
  --Height perturbations with sd = 2.4384 meters added.

  --Total Basal area sampled = 48.370658


R> head(tp)


  species    height       dbh topDiam solidType          x         y  units
```

```
1  NHdwds 17.859360 17.664423       0       2.0 149.552999 89.918926 metric
2  NHdwds 23.455572 52.055242       0       1.6  26.391240 38.404951 metric
3  NHdwds 17.497384 22.907154       0       2.6  26.720776 53.124171 metric
4  NHdwds 25.957319 36.436657       0       2.5 134.459547 66.727826 metric
5  NHdwds 16.614532 19.546530       0       2.6  80.744292 29.933990 metric
6  NHdwds 20.146507 28.151105       0       1.5  95.704550 79.859052 metric
```

The first command shows the default arguments for the function if called with only a "buffered-Tract" object. The second line runs the function with the parameters used in Gove et al. (2020). Compare the results of the output to those in the previous code snippet where we calculated the main stand quantities for illustration. Note that the final part of the output titled "Totals for the tract" presents a summary based on the full tract. In summary, the following steps (not necessarily in this order) are performed in `drawTreePop` to generate these results...

1. Convert the input arguments to metric if necessary (they are converted to metric of the "bufferedTract" object is in metric) and calculate the missing stand parameters as was done above.

2. Expand the per hectare quantities up to the total tract size based on the internal area of 2.0164 ha.

3. Draw a random diameter from the three-parameter Weibull function for each of the $N = 667$ trees.

4. Calculate a height for each tree using the "all species" height equation given in Fast and Ducey (2011). Add a height perturbation, ϵ, to these mean heights using random draws from a normal distribution with standard deviation equal to the $\sigma =$ `hgt.sd` argument passed in the call; i.e., $\epsilon \sim \mathcal{N}(0, \sigma^2)$.

5. Determine the spatial coordinates for each tree using the `SSI` function in the **R** `spatial` package with an inhibition distance as given by the `inhibitDist` argument. In this case the default distance was used, and be careful to note that his is always specified in the same units as the "bufferedTract" object.

6. Lastly, plot the diameter distribution from the realized population as a histogram and superimpose the Weibull model on this. Note that the argument `main = NA` is passed to the `hist` function through the `...` argument, suppressing the default title. The result is found in Figure 1.

The third line in the above code simply shows the first few records of the data frame returned from `drawTreePop` holding the synthetic tree population.[1]

[1] For more information on the `solidType` and `topDiam` columns—and indeed on how trees are represented within *sampSurf*—please see Gove (2011).

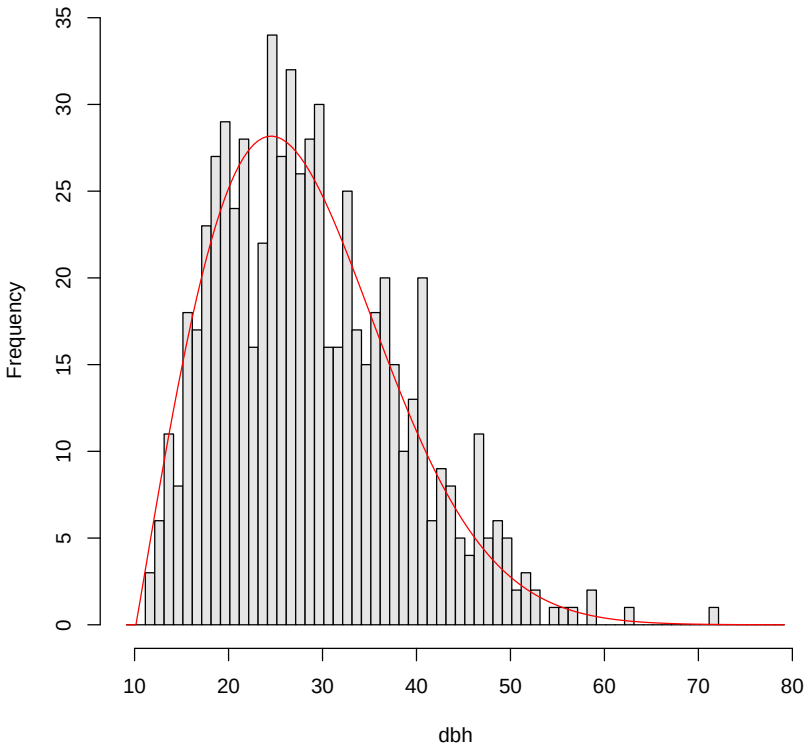


Figure 1: Histogram showing the synthetic diameter distribution of $N = 667$ trees with the theoretical three-parameter Weibull distribution (red line).

2.3 The Full Process for Northern Hardwoods

The above routines are designed to be used individually to create the different components required for a tract and 'synthetic' tree population, specifically for northern hardwoods. These routines have been combined into a simple method called `makePop` that automate the above steps. Specifically, it will...

1. Create a "bufferedTract" object with `initTract` (§ 2.1).
2. Create a 'synthetic' northern hardwoods tree population within the internal extents of the tract with `drawTreePop` (§ 2.2.1).
3. Take the data frame output from `drawTreePop` and create a "standingTrees" object (see, e.g., Gove, 2011).

The following will help solidify the idea...

```
R> args(makePop)


function (extents = c(x = 150, y = 150), cellSize = 1, bufferWidth = 14,
    units = "metric", description = "Tract and northern hardwoods population",
    ...)
NULL


R> nh.pop = makePop(c(x = 178, y = 178), bufferWidth = 18, B = 100,
+                   hgt.sd = 8, startSeed = 355)


Creating tract...
Creating tree population...
Input specs...
  Number of trees/acre = 134
  Quadratic msd = 11.69267 in
  Basal area/acre = 100
  Weibull shape = 2
  Weibull scale = 8 in
  Weibull shift = 4 in
Output specs...
  units = metric
  Number of trees/ha = 331
  Quadratic msd = 29.699381 cm
  Basal area/ha = 22.956841
  Weibull scale = 20.32 cm
  Weibull shift = 10.16 cm

  Totals for the tract...
  --Tract area (inside the buffer) = 2.0164
  --Total N for above area = 667.4284
  --Height perturbations with sd = 2.4384 meters added.
  --Total Basal area sampled = 48.370658

Creating standingTrees object...
Adding to data frame...


R> str(nh.pop, 1)


List of 3
```

```
 $ btr   :Formal class 'bufferedTract' [package "sampSurf"] with 17 slots
 $ strees:Formal class 'standingTrees' [package "sampSurf"] with 5 slots
 $ trees :'data.frame': 667 obs. of  12 variables:
```

The first point to notice is that one can and should pass arguments for any of the above three functions that are called in `makePop` through the call to this routine. In the above call the tract size arguments are sent to `initTract`, while the basal area (`B`) and standard deviation of height (`hgt.sd`) are passed on to `drawTreePop`. The third line of the code above shows a summary of the results that are returned from the routine in a `list` structure[2].

The second point to notice in the above code is that `makePop` adds more information to the data frame of trees (returned in `nh.pop$trees`). This is easily seen by comparing the first few records of the corresponding data frames...

```
R> head(tp)


  species    height       dbh topDiam solidType          x         y  units
1  NHdwds 17.859360 17.664423       0       2.0 149.552999 89.918926 metric
2  NHdwds 23.455572 52.055242       0       1.6  26.391240 38.404951 metric
3  NHdwds 17.497384 22.907154       0       2.6  26.720776 53.124171 metric
4  NHdwds 25.957319 36.436657       0       2.5 134.459547 66.727826 metric
5  NHdwds 16.614532 19.546530       0       2.6  80.744292 29.933990 metric
6  NHdwds 20.146507 28.151105       0       1.5  95.704550 79.859052 metric


R> head(nh.pop$trees)


       species    height       dbh topDiam solidType          x         y
tree.1  NHdwds 17.859360 17.664423       0       2.0 149.552999 89.918926
tree.2  NHdwds 23.455572 52.055242       0       1.6  26.391240 38.404951
tree.3  NHdwds 17.497384 22.907154       0       2.6  26.720776 53.124171
tree.4  NHdwds 25.957319 36.436657       0       2.5 134.459547 66.727826
tree.5  NHdwds 16.614532 19.546530       0       2.6  80.744292 29.933990
tree.6  NHdwds 20.146507 28.151105       0       1.5  95.704550 79.859052
        units            id        vol          ba       vbar
tree.1 metric tree:6c2pgz15 0.17117562 0.024506924  6.9847861
tree.2 metric tree:1s574yod 1.65813126 0.212823128  7.7911234
tree.3 metric tree:e28t3mn5 0.32208958 0.041212804  7.8152794
tree.4 metric tree:fi9xg236 1.13547213 0.104271812 10.8895405
```

[2]Note that there is much more to these objects, which can be seen by passing, e.g., `2` rather than `1` to the `str` method.

```
tree.5 metric tree:1m4k0s6v 0.22424118 0.030007460  7.4728477
tree.6 metric tree:861isrg3 0.41273156 0.062241605  6.6311202


R> identical(tp, nh.pop$trees)


[1] FALSE
```

The information through the `units` column is all identical, but four other columns have been added to the data frame, in addition to a tree number in the row names. The interested reader may want to verify the VBARs in the last column (e.g., (1)). The check for exact duplication of the two data frames obviously fails because of the differing dimensions and `rownames`.

3 Big BAF sampling

3.1 A Quick Review of Big BAF Sampling

A simple review/overview with notation used in this document follows.[3] Let $\mathcal{F}_{\mathrm{c}}$ and $\mathcal{F}_{\mathrm{v}}$ ($\mathrm{m}^2\,\mathrm{ha}^{-1}$) be the basal area factors for the selection of count and volume trees, respectively. Thus, the $\mathrm{BAF}_{\mathrm{c}}$ sample of trees are selected with the $\mathcal{F}_{\mathrm{c}}$ gauge, while the $\mathrm{BAF}_{\mathrm{v}}$ sample uses the $\mathcal{F}_{\mathrm{v}}$ gauge to select the trees for volume. Notably, $\mathcal{F}_{\mathrm{c}} < \mathcal{F}_{\mathrm{v}}$, and possibly $\mathcal{F}_{\mathrm{c}} \ll \mathcal{F}_{\mathrm{v}}$.

The double sampling big BAF estimator begins by forming the volume to basal area ratios for each tree in the $\mathrm{BAF}_{\mathrm{v}}$ sample of n points (e.g., Kershaw et al., 2016, p. 377); i.e., for the ith tree on a given sample point we have...

$$\mathbb{V}_i = \frac{v_i}{b_i} \tag{1}$$

and, averaging over all trees on the $\mathrm{BAF}_{\mathrm{v}}$ sample gives

$$\bar{\mathbb{V}} = \frac{1}{m_{\mathrm{v}}} \sum_{s=1}^{n} \sum_{i=1}^{m_{\mathrm{v}s}} \mathbb{V}_i \tag{2}$$

where v is some estimate of tree volume and b is tree basal area. Here m_{v_s} is the number of $\mathrm{BAF}_{\mathrm{v}}$ trees measured for volume on the sth point; it follows that the total number of trees sampled for volume is therefore $m_{\mathrm{v}} = \sum_{s=1}^{n} m_{\mathrm{v}s}$.

[3] This is the same notation used in Gove et al. (2020).

The full set of counts with BAF_c on all n points provides an estimate of the mean basal area, $\hat{B}_\text{c}$. The estimator for average basal area per hectare based on the count sample is

$$\begin{aligned}\hat{B}_\text{c} &= \frac{\mathcal{F}_\text{c}}{n} m_\text{c} \\ &= \bar{m}_\text{c} \mathcal{F}_\text{c} \end{aligned} \tag{3}$$

where $m_\text{c} = \sum_{s=1}^{n} m_{\text{c}_s}$ is the number of BAF_c "in" trees on all n sample points, and m_{c_s} is the number of BAF_c trees tallied on the sth point. Similarly, the estimator for the total is simply $\hat{B}_\text{c} = \bar{m}_\text{c} A \mathcal{F}_\text{c}$, where A is the area of the tract in hectares. Note that $\hat{B}_\text{c}$ can refer to either the total or per unit area estimate according to the context. Because the sampling surfaces are in terms of totals, that will be the main use throughout.

The product of the mean VBAR and basal area gives an estimate of the volume under big BAF as...

$$\hat{V}_\mathcal{B} = \bar{\mathbb{V}} \times \hat{B}_\text{c} \tag{4}$$

It is important to realize that in (2) the total number of VBAR trees *must* include replicates for any tree sampled on more than one point. The double sum ensures that the summation over the s sample points will include multiple counts on the point-wise tallies. An alternate but equivalent way to calculate $\bar{\mathbb{V}}$ is given in G&V (8.34a),[4] which is simply the ratio of the expanded total volumes to total basal areas from the big BAF sample. This form makes it clear that we must include trees sampled more than once on different points.

3.1.1 Estimated variances

The variances corresponding to (2) and (3) are (e.g., Gregoire, 2009, p. 3, (12) & (13)) as follows...

$$\widehat{\text{var}}\left(\bar{\mathbb{V}}\right) = \frac{1}{m_\text{v}(m_\text{v} - 1)} \sum_{s=1}^{n} \sum_{i=1}^{m_{\text{v}s}} \left(\mathbb{V}_i - \bar{\mathbb{V}}\right)^2 \tag{5}$$

and

$$\begin{aligned}\widehat{\text{var}}\left(\hat{B}_\text{c}\right) &= \frac{1}{n(n-1)} \sum_{s=1}^{n} \left(\mathcal{F}_\text{c} m_{\text{c}_s} - \hat{B}_\text{c}\right)^2 \\ &= \frac{\mathcal{F}_c^2}{n(n-1)} \sum_{s=1}^{n} \left(m_{\text{c}_s} - \bar{m}_\text{c}\right)^2 \end{aligned} \tag{6}$$

Kershaw et al. (2016, p. 380) provide equivalent estimators. The standard error estimates corresponding to (5) and (6) are given as $\widehat{\text{se}}\left(\bar{\mathbb{V}}\right) = \sqrt{\widehat{\text{var}}\left(\bar{\mathbb{V}}\right)}$ and $\widehat{\text{se}}\left(\hat{B}_\text{c}\right) = \sqrt{\widehat{\text{var}}\left(\hat{B}_\text{c}\right)}$, respectively.

[4] Throughout the rest of this document, G&V should understood as Gregoire and Valentine (2008). And equations of the form (8.34a) refer to their equation of that number.

3.1.2 Ratio variance estimator

In addition to the normal sample variance estimator given in (5), G&V (8.42) & (8.43) provide an alternative ratio variance estimator for $\bar{\mathbb{V}}$. To be consistent with this reference, the estimators below are defined for the total quantities on a tract of area A. In addition, we will need to differentiate quantities such as basal area that can be computed under both BAFs. As previously noted, in both the above and in what follows the context will distinguish whether the per unit area or total estimates are used, and the same notation is adopted for each; thus, the estimator for the total basal area from the BAF_c inventory is simply $\hat{B}_\text{c} = \bar{m}_\text{c} A \mathcal{F}_\text{c}$. Likewise, the total basal area for the big BAF sample is similarly defined as $\hat{B}_\text{v} = \bar{m}_\text{v} A \mathcal{F}_\text{v}$. The variance for the total basal area over the tract scales (6) by A^2. In *sampSurf* all cell-wise (point-wise) and summary statistics are calculated based on totals rather than per unit area values for objects of class "sampSurf", thus the above arises by default.

The VBAR variances (5) are not scaled as they are based on the number of sampled trees. However, there is an alternate variance estimator for VBAR that is based on the ratio of total volume to basal area. In general, for either the count or big BAF sample, let...

$$\hat{V}_s = \mathcal{F} A \sum_{i=1}^{m_s} \mathbb{V}_i$$

and

$$\hat{V} = \frac{1}{n} \sum_{s=1}^{n} \hat{V}_s$$

G&V (8.33) note that multiplying by $^{B}/_{B}$, total volume can be written using this ratio-based quantity as...

$$\hat{V} = \hat{B} \left(\frac{\hat{V}}{\hat{B}} \right)$$

This is a general (i.e., outside double sampling) motivation for using the product variances to be discussed in the following sections.

However, if we use the big BAF double sampling estimates for volume ($\hat{V}_\text{v}$) and basal area ($\hat{B}_\text{v}$) in the ratio portion of $\hat{V}$, we can write this as...

$$\hat{V} = \hat{B}_\text{c} \left(\frac{\hat{V}_\text{v}}{\hat{B}_\text{v}} \right) \tag{7}$$

$$= \hat{B}_\text{c} \times \bar{\mathbb{V}}_\text{R} \tag{8}$$

where

$$\bar{\mathbb{V}}_{\mathrm{R}} = \frac{\hat{V}_{\mathrm{v}}}{\hat{B}_{\mathrm{v}}} \tag{9}$$

The ratio quantity, $\bar{\mathbb{V}}_{\mathrm{R}}$, comes from the big BAF sample and it is scaled by the basal area estimate from the count sample. G&V (8.34) note that the ratio quantity, $\bar{\mathbb{V}}_{\mathrm{R}}$, is equivalent to $\bar{\mathbb{V}}$ in (2)[5]. Thus, they posit that one could alternatively use the following variance estimator for the ratio $\bar{\mathbb{V}}_{\mathrm{R}} = \hat{V}_{\mathrm{v}}/\hat{B}_{\mathrm{v}}$ for the variance of the VBAR trees in (5).

The alternative ratio variance estimator for $\bar{\mathbb{V}} \equiv \bar{\mathbb{V}}_{\mathrm{R}}$ given by G&V (8.42) is...

$$\begin{aligned} \widehat{\mathrm{var}}_{\mathrm{R}}\left(\bar{\mathbb{V}}\right) &= \widehat{\mathrm{var}}\left(\frac{\hat{V}_{\mathrm{v}}}{\hat{B}_{\mathrm{v}}}\right) \\ &= \frac{1}{\hat{B}_{\mathrm{v}}^2}\frac{s_r^2}{n} \end{aligned} \tag{10}$$

where

$$s_r^2 = \frac{1}{n-1}\sum_{s=1}^{n}\left(\hat{V}_{\mathrm{v}_s} - \bar{\mathbb{V}}\hat{B}_{\mathrm{v}_s}\right)^2$$

Of course, it is the variance of volume, $\hat{V}$, that is of ultimate interest. As can be seen in (4) and (8), the big BAF estimate of volume comes from a product of means. The following section looks at some ways to estimate this variance.

4 Variance Estimation

As noted above, there are several possible variance estimators that can be compared for big BAF. These are discussed in more detail below and in § 8.

4.1 Non-parametric estimators

The two most likely candidates for use in this category include the jackknife and bootstrap estimators. Both bootstrapping (with $\mathrm{BC_a}$ intervals) and jackknifing are included in the `ssExtra` package

[5] We will use the latter in most cases, but keep in mind that $\bar{\mathbb{V}}_{\mathrm{R}} \equiv \bar{\mathbb{V}}$ so that it can be calculated either by (2) or from the ratio (9).

for big BAF, within the MC context (§ 6). The R `bcaboot` package (Efron and Narasimhan, 2018) is used for the calculation of bootstrap and jackknife sample statistics; but refer to § A in the Appendix for a some observations on the R packages available for these methods.

4.2 Exact Variance of a Product

As noted above, we can see from (4) that the big BAF estimator is the product of two random variables (estimators really). Goodman (1960) presented the formulas for the variance of the product of two random variables as well as those for estimators. It is the latter that we are interested in, hence Goodman's formula (6) applies here...

$$\operatorname{Var}(\bar{x}\bar{y}) = \operatorname{E}[x]^2 \operatorname{Var}(\bar{y}) + \operatorname{E}[y]^2 \operatorname{Var}(\bar{x}) + \operatorname{Var}(\bar{x}) \operatorname{Var}(\bar{y}) \tag{11}$$

Goodman also presents an "unbiased estimate" [estimator] for (11), which is suggested in Gregoire and Valentine (2008, p. 258, (8.36)) as a "design unbiased estimator" for the variance of a product. This is given by Goodman's equation (9) as...

$$\widehat{\operatorname{var}}_{\mathrm{G}}(\bar{x}\bar{y}) = \bar{x}^2 \widehat{\operatorname{var}}(\bar{y}) + \bar{y}^2 \widehat{\operatorname{var}}(\bar{x}) - \widehat{\operatorname{var}}(\bar{x}) \widehat{\operatorname{var}}(\bar{y}) \tag{12}$$

where, e.g., $\operatorname{var}(\bar{x})$ is the usual variance of the mean.

Let the random variables $\bar{x} \equiv \bar{\mathbb{V}}$ and where $\bar{y} \equiv \hat{B}_{\mathrm{c}}$. Then for big BAF sampling we have from (12)...

$$\widehat{\operatorname{var}}_{\mathrm{G}}\left(\bar{\mathbb{V}}\hat{B}_{\mathrm{c}}\right) = \bar{\mathbb{V}}^2 \widehat{\operatorname{var}}\left(\hat{B}_{\mathrm{c}}\right) + \hat{B}_{\mathrm{c}}^2 \widehat{\operatorname{var}}\left(\bar{\mathbb{V}}\right) - \widehat{\operatorname{var}}\left(\bar{\mathbb{V}}\right) \widehat{\operatorname{var}}\left(\hat{B}_{\mathrm{c}}\right) \tag{13}$$

where the appropriate variances are given in (5) and (6). Note that (13) is equally applicable when $\hat{B}_{\mathrm{c}}$ is the expanded total basal area, or when it represents basal area per unit area. In all cases, $\bar{\mathbb{V}}$ is a tree-wise statistic and so it is never scaled by area. As noted above, *sampSurf* represents the sampling surface and all statistics as totals rather than per unit area values.

4.3 The Delta Method approximation to the big BAF sampling variance

The Delta Method is a method based on Taylor Series that gives an approximation to the variance, in this case the variance of a product. Gove et al. (2020) give a historical summary of this method and demonstrate its connection to Bruce's method (§ 4.3.1). They note that in the historical development that the Delta Method seems to have been well-known at the time. For example, Goodman (1960, p. 708) refers to "the usual formula" for the variance of a product as an "approximation" and cites "Yates (1953, p. 198)" as one source for this formula. An alternate source that covered the Delta Method and was commonly available at the time was Cramér (1946, p. 353).[6] For more information see Gove et al. (2020) and the accompanying supplementary material.

[6] Note that Cramér provides a proof of the Delta Method for the mean and variance, but his notation can be difficult.

The volume estimator given in (4) is the product estimator $\hat{V}_{\mathcal{B}} = \bar{\mathbb{V}} \times \hat{B}_{\text{c}}$. The Delta Method approximation for this product variance is given as (Gove et al., 2020)...

$$\widehat{\text{var}}_{\delta}\left(\hat{V}_{\mathcal{B}}\right) = \widehat{\text{var}}\left(\bar{\mathbb{V}}\right) \hat{B}_{\text{c}}^2 + \widehat{\text{var}}\left(\hat{B}_{\text{c}}\right) \bar{\mathbb{V}}^2 \tag{14}$$

where the two estimated variance terms for the mean VBAR and basal area are computed using the formulas presented in (5) and (6), respectively. When the assumption of independence is not tenable, including the covariance term in (14) will give a first-order approximation to the variance for big BAF as...

$$\widehat{\text{var}}_{\delta}\left(\hat{V}_{\mathcal{B}}\right) = \widehat{\text{var}}\left(\bar{\mathbb{V}}\right) \hat{B}_{\text{c}}^2 + \widehat{\text{var}}\left(\hat{B}_{\text{c}}\right) \bar{\mathbb{V}}^2 + 2\,\widehat{\text{cov}}\left(\bar{\mathbb{V}}, \hat{B}_{\text{c}}\right) \cdot \bar{\mathbb{V}} \cdot \hat{B}_{\text{c}} \tag{15}$$

4.3.1 Bruce's method

It is straightforward to show that Bruce's method is simply the Delta method approximation. It was evidently first used in forest sampling by Bell and Alexander (1957) and it is uncertain whether these authors derived it or acquired it from another source, such as those listed above. Later, Bruce (1961, p. 26) used this and evidently popularized it—hence the name. Again, Bruce gave no hint as to the source.

The Delta Method variance estimator written in terms of $\bar{\mathbb{V}}$ and B is, from (14)...

$$\widehat{\text{var}}_{\delta}\left(\bar{\mathbb{V}}\hat{B}_{\text{c}}\right) = \widehat{\text{var}}\left(\bar{\mathbb{V}}\right) \hat{B}_{\text{c}}^2 + \widehat{\text{var}}\left(\hat{B}_{\text{c}}\right) \bar{\mathbb{V}}^2 \tag{16}$$

and dividing both sides by $\hat{B}_{\text{c}}^2\bar{\mathbb{V}}^2$ gives...

$$\frac{\widehat{\text{var}}_{\delta}\left(\bar{\mathbb{V}}\hat{B}_{\text{c}}\right)}{\hat{B}_{\text{c}}^2\bar{\mathbb{V}}^2} = \frac{\widehat{\text{var}}\left(\bar{\mathbb{V}}\right) \hat{B}_{\text{c}}^2}{\hat{B}_{\text{c}}^2\bar{\mathbb{V}}^2} + \frac{\widehat{\text{var}}\left(\hat{B}_{\text{c}}\right) \bar{\mathbb{V}}^2}{\hat{B}_{\text{c}}^2\bar{\mathbb{V}}^2}$$

taking the square root of both sides and converting to percent with $\widehat{\text{var}}_{\delta}\left(\bar{\mathbb{V}}\hat{B}_{\text{c}}\right) \equiv \widehat{\text{var}}_{\delta}\left(\hat{V}_{\mathcal{B}}\right)$ gives

$$\widehat{\text{se}\%}\left(\hat{V}_{\mathcal{B}}\right) = \sqrt{\widehat{\text{se}\%}\left(\bar{\mathbb{V}}\right)^2 + \widehat{\text{se}\%}\left(\hat{B}_{\text{c}}\right)^2} \tag{17}$$

Recall that in the above that the $\widehat{\text{var}}(\cdot)$ terms are the variance of the mean so that (17) is written in terms of the standard error.

4.3.2 Deconstructing the Delta Method

The form of the Delta Method given in (16) is composed of two 'weighted' variance terms. We think (correctly so) of both basal area and VBAR as random variables. However, consider the implication if one of the two are treated as known. For example, for each variance term, treat the mean multiplier as a constant. Then we have the following interpretation.

1. The first term in (16) can be written using (5) as...

$$\begin{aligned}\hat{B}_{\text{c}}^2 \widehat{\text{var}}\big(\bar{\mathbb{V}}\big) &= \frac{\hat{B}_{\text{c}}^2}{m_{\text{v}}(m_{\text{v}}-1)} \sum_{s=1}^{n} \sum_{i=1}^{m_{\text{v}s}} \big(\mathbb{V}_i - \bar{\mathbb{V}}\big)^2 \\ &= \frac{1}{m_{\text{v}}(m_{\text{v}}-1)} \sum_{s=1}^{n} \sum_{i=1}^{m_{\text{v}s}} \Big(\hat{B}_{\text{c}}\mathbb{V}_i - \hat{B}_{\text{c}}\bar{\mathbb{V}}\Big)^2 \end{aligned} \tag{18}$$

   Note that any product of a VBAR with basal area yields an estimate of volume. Thus, both terms inside the summation are in terms of volume, yielding a variance in volume. In fact, the 'mean' volume in the variance term is our estimate of big BAF volume given by $\hat{V}_{\mathcal{B}}$ in (4).

2. Similarly, the second term in (16) can be written using (6) as...

$$\begin{aligned}\bar{\mathbb{V}}^2 \widehat{\text{var}}\Big(\hat{B}_{\text{c}}\Big) &= \frac{\bar{\mathbb{V}}^2}{n(n-1)} \sum_{s=1}^{n} \Big(\mathcal{F}_{\text{c}} m_{\text{c}_s} - \hat{B}_{\text{c}}\Big)^2 \\ &= \frac{1}{n(n-1)} \sum_{s=1}^{n} \Big(\bar{\mathbb{V}}\mathcal{F}_{\text{c}} m_{\text{c}_s} - \bar{\mathbb{V}}\hat{B}_{\text{c}}\Big)^2 \end{aligned} \tag{19}$$

   Again, we have the terms within the summation written as a product of basal area and VBAR, yielding a variance in volume. Similarly, the 'mean' estimate of volume in the variance term is also $\hat{V}_{\mathcal{B}}$ here as well.

The fact that the two Delta Method variance terms are both variances of volume is unremarkable, as it is clear that the Delta Method is a weighted sum of variances as noted above and clearly shown in (14). It has also been pointed out (and is clear from (17) in terms of the standard errors from Bruce's method) that (18) is based on the number of VBAR trees selected by the BAF_{v} subsample of points, and is therefore a variance in estimated volume of the VBAR trees only. The second term, (19) is somewhat less apparent from Bruce's formula, where it is interpreted (quite correctly) as the (squared) standard error (as a percent of the mean) of basal area over all n sample points. Writing this in the variance form demonstrates that the term is also identifiable as a estimated variance in volume over all sample points. However, it does contrast with the usual identification of (17) being composed of the sum of VBAR and basal area relative variances of the mean. The algebra in going from (16) to (17) is enough to disguise the original interpretation of Bruce's Method as being

one of the sum of volume variances. Both interpretations are correct, relative to their associated formulæ.

Beginning with Bruce (1961, p. 26) a number of authors (e.g., Bell et al., 1983; Desmarais, 2002; Marshall et al., 2004; Kershaw et al., 2016, p. 380; Yang et al., 2017) have made the point that the two variance terms can be quite different in magnitude. Consequently, reducing the component of greatest magnitude will have the largest affect on shrinking the overall Delta Method variance; of course this was before the actual big BAF method was proposed, but the idea is the same. All authors agree that under big BAF sampling any increased sampling effort should be concentrated on reducing the second component by adding more points, rather than more VBAR trees, which parallels the recommendations of Bruce. The simulations in § 6.4 will corroborate this recommendation.

4.4 Horvitz-Thompson (H-T) variance

As noted above, (Gregoire, 2009, p. 3) and G&V (p. 259) suggests that his equation (14) might be preferred as a variance estimator because it is design-unbiased. The estimator is the usual variance of the mean volume...

$$\widehat{\text{var}}\left(\hat{V}\right) = \frac{1}{n(n-1)} \sum_{s=1}^{n} \left(\hat{V}_s - \hat{\bar{V}}\right)^2 \tag{20}$$

where $\hat{V}_s$ is the volume estimate on the sth sample point, and n is the total number of points as defined above in the inventory. Because n will include a number of zero-volume points under the big BAF design, it is anticipated that this variance will be high. As noted below, the simulations will have volume calculated on a larger proportion of the n sample points under the BAF_c sample, so this will enable a comparison using (20) on both the count and BAF_v samples.

5 Double Sampling in sampSurf

The *sampSurf* package was created to simulate simple sampling methods used in forestry that are applied individually to a population of trees or down logs. As such, it was never envisioned to be extended to multi-stage or multi-phase sampling schemes. Such schemes can differ extensively in their design and execution, all but negating a general double sampling class (in an object-oriented sense) from being developed. The big BAF method is one example of a double sampling method that is somewhat unique in the sense that both primary and secondary samples are conducted on the same sample units, where in most familiar double sampling designs there are two different sample sizes with a larger primary and smaller secondary sample being taken. In addition, these designs rely on a two different but related variables (attributes) in their application. However, in *sampSurf*, different sampling surfaces are required for each different attribute if multiple attributes are to be simulated. How these may then be combined or used is application specific.

The fact that *sampSurf* was really designed to look at the efficiency of different methods over the entire population makes double sampling a different sort of creature. It is in the drawing of samples from the overall population that these designs find their application, not at the full population level (i.e., the entire sampling surface). Therefore, their applicability comes in sampling from the full population of sample points (grid cell centers) based on the protocols of the method in question. This then puts the application of double sampling designs largely in the realm of Monte Carlo (Monte Carlo) sampling from the population.

The *ssExtra* package does, however, have a new "virtual" double sampling class from which specific protocols can be established. A virtual class is in a sense a base class[7] in the S4 objected oriented system within R; and it is one from which no objects can be directly created. Its use is in the establishment of a common minimum degree of functionality that will be shared by all subsequent subclass definitions. Please see Chambers (2008, Chapter 9) or Gentleman (2008, Chapter ??) for more information on S4 classes and object-oriented programming in R in general.

In the following, the virtual class for double sampling ("ssDoubleSampling") and its child class definition for big BAF sampling ("ssBigBAF") are described in detail. Subsequently, an example is presented that demonstrates how to work with the different object components—known as "slots" in S4 class definition parlance.

5.1 The R "ssDoubleSampling" Class

The virtual class used to extend *sampSurf* for double sampling applications is the "ssDoubleSampling" class. Because of the flexibility of double sampling designs, it is a simple class with only one slot[8] and currently no methods. The class looks like this...

```
R> showClass('ssDoubleSampling')


Virtual Class "ssDoubleSampling" [package "ssExtra"]

Slots:

Name:  description
Class:   character

Known Subclasses: "ssBigBAF"
```

[7] This is playing a little loose with the terminology. It is 'base' in the sense of direct descendants with single inheritance, but is only one of possible several 'base' classes in the sense of multiple inheritance—i.e., with multiple parents.

[8] Please note that this may change in the future.

The above shows us that the only slot common to this virtual class and all subclasses is a character description for the created object—it is a pretty minimal class at this point. In addition, we see that there is currently only one subclass: "ssBigBAF".

5.2 "ssBigBAF:" Big BAF Sampling Class

As noted earlier, the *sampSurf* package is structured in such a way as to accommodate sampling methods that are based on a single sampling surface containing one attribute, which has inclusion zones appropriate for the method of sampling being studied. A simple example would be inclusion zones of fixed area for fixed area circular plot sampling. Big BAF sampling is somewhat different in that its context as a form of double sampling requires that we have not one sampling surface, but three; i.e., one surface for the basal area count sample, and then both basal area and volume surfaces (to create the VBARs) for the volume sample. However, since this is simulation and extra surfaces cost nothing but a little extra computer time to generate, a fourth surface is also of interest and part of this class, that of the count volume surface. Having this last surface allows one to determine what the volume estimate would be for a given BAF_c on the full sample of points from the surface; this is particularly useful in the Monte Carlo experiments.

A subclass of "ssDoubleSampling" is available to create objects that can be used in applying the big BAF sampling method in a Monte Carlo context. The class structure is described as follows...

```
R> showClass('ssBigBAF')


Class "ssBigBAF" [package "ssExtra"]

Slots:

Name:       ss.bb.vol       ss.bb.ba       ss.ct.ba      ss.ct.vol
Class:       sampSurf       sampSurf       sampSurf       sampSurf

Name:          csl.bb         csl.ct    description
Class: ssCellStemList ssCellStemList      character

Extends: "ssDoubleSampling"
```

5.2.1 Class slots

- `ss.bb.vol`: An object of class "sampSurf" for the big BAF volume surface.
- `ss.bb.ba`: An object of class "sampSurf" for the big BAF basal area surface.

- `ss.ct.ba`: An object of class "sampSurf" for the count basal area surface.
- `ss.ct.vol`: An object of class "sampSurf" for the count volume surface.
- `csl.bb`: An object of class "ssCellStemList" for the two big BAF surfaces.
- `csl.ct`: An object of class "ssCellStemList" for the two count surfaces.
- `description`: A `character` description of the object if desired (a default is given for the class in the constructor method). Note that his slot is inherited from the "ssDoubleSampling" parent class.

The above is a seemingly simple class structure, containing only seven slots. However, because four full *sampSurf* objects must be created for the constructor (§ 5.3), it can take some time on large populations. In addition, the class contains slots for another new class: "ssCellStemList". These objects also require time to build in the creation of the required `list` objects for the class.

5.2.2 A note on the "ssCellStemList" class

This class, while necessary for the implementation of big BAF sampling via Monte Carlo experiments, is not essential to fully understand in order to use the system. A short explanation is that the Monte Carlo subsampling will select samples of size n from the four populations. In the process of calculating the required statistics and summary information, it is necessary to have a list of the trees that are sampled in each cell (i.e., at each sample point) comprising a given Monte Carlo sample. This class builds the `list` objects for every cell in each sampling surface pair[9] to provide this information. It also builds a `list` that is its complement, identifying which cells (sample points) on the sampling surface lie within each tree's inclusion zone. For more information on the structure of these objects and their use in general please type `class?ssCellStemList` at the command prompt and follow the links there to the constructor method if desired.

5.3 An example

As noted above, in order to create an object of class "ssBigBAF" one should use the object constructor of the same name: `ssBigBAF`. The constructor takes as input the first four slots described in § 5.2.1. It follows that one must first create the four sampling surfaces to fill the four *sampSurf* object slots (§ 5.2.1). Thus, the job of the "ssBigBAF" object constructor is simply to do the required work to complete the two remaining "ssCellStemList" slots and build the object. The constructor is detailed in the help file, which can be access using `methods?ssBigBAF` at the command prompt. In summary, simply create the four *sampSurf* objects and pass them to the constructor.

[9] Because the inclusion zones are determined only by the tree size and BAF, and both of these are constant for the two count and two big BAF surfaces, only two objects are required—one for count and one for volume.

Note that the four objects passed must be conformable with each other; that is, they must have the same tract dimensions, units of measure, tree population, and the two count surfaces must share the same BAF_c, while the big BAF surfaces likewise must share the same BAF_v. These constraints are rigorously enforced through a validity check of the newly created object.

The above is simple enough and assumes that one already has a tree population, common "buffered-Tract" to place it on, and specification (e.g., the angle gauges) for the inclusion zones associated with each of the two BAFs (BAF_c and BAF_v) used. In order to provide more help to essentially automate the procedure, one last function has been included that will combine all the steps through "makePop" (§ 2.3), then create the four sampling surfaces from this information, and finally run the constructor, `ssBigBAF`, on these results to create a return object of class "ssBigBAF". An example follows using the results for a smaller, more manageable tract size than the prior examples.

```
R> args(createBBNH)


function (extents = c(x = 178, y = 178), cellSize = 1, bufferWidth = 18,
    units = "metric", baf.ct = 4, baf.bb = 10, startSeed = 355,
    ...)
NULL


R> ssBB = createBBNH(c(x=100, y=100), bufferWidth = 14, B = 60, baf.bb = 20)


Creating tract...
Creating tree population...
Input specs...
  Number of trees/acre = 80
  Quadratic msd = 11.69267 in
  Basal area/acre = 60
  Weibull shape = 2
  Weibull scale = 8 in
  Weibull shift = 4 in
Output specs...
  units = metric
  Number of trees/ha = 198
  Quadratic msd = 29.699381 cm
  Basal area/ha = 13.774105
  Weibull scale = 20.32 cm
  Weibull shift = 10.16 cm

  Totals for the tract...
  --Tract area (inside the buffer) = 0.5184
  --Total N for above area = 102.6432
```

```
  --Height perturbations with sd = 1.8288 meters added.
  --Total Basal area sampled = 7.0688455

Creating standingTrees object...
Adding to data frame...

Creating standingtreeIZs...
Creating sampling surfaces...
---Big BAF volume...
Number of trees in collection = 102
Heaping tree: 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30

---Big BAF basal area...
Number of trees in collection = 102
Heaping tree: 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30

---Count volume...
Number of trees in collection = 102
Heaping tree: 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30

---Count basal area...
Number of trees in collection = 102
Heaping tree: 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30

Creating ssBigBAF object...
Creating the big BAF surface cell-stem IZ list...
Creating the count surface cell-stem IZ list...

***>Be careful to heed any warning about partial inclusion zones lying outside the tract!
    This warning means you need to increase the buffer size for this population!


R> class(ssBB)


[1] "ssBigBAF"
attr(,"package")
[1] "ssExtra"
```

The final line of output verifies that we have created an object of the correct class. Please remember to keep in mind that those arguments that are being passed through the `...` argument to `drawTreePop` should be in "English" units (i.e., basal area per acre, `B`), while all other units in the above are metric.

The northern hardwoods tract used in Gove et al. (2020) is larger than the example used here, but it has the same tree population as is found in § 2.2.1. The actual command that can be used to create the "ssBigBAF" object used in the simulations for a BAF pair $(\mathcal{F}_c, \mathcal{F}_v) = (4, 20)$ $m^2\,ha^{-1}$is given as...

```
R> ssBB.all = createBBNH(B = 100, hgt.sd = 8, baf.bb = 20, startSeed = 355,
+                        runQuiet = TRUE)
```

Please note in the above that some of the argument default values are taken, not only in `createBBNH`, but also in the routines that it calls as discussed in the previous sections.

A summary of the "ssBigBAF" object shows the various main components in the object slots...

```
R> ssBB


Object of class: ssBigBAF
------------------------------------------------------------
Big BAF sampling surface object
------------------------------------------------------------

Sampling surface slot contents/estimates...
  ss.bb.vol: Big BAF volume surface, horizontalPointIZ, 20 BAF (metric)
  ss.bb.ba: Big BAF basalArea surface, horizontalPointIZ, 20 BAF (metric)
  ss.ct.ba: Count BAF basalArea surface, horizontalPointIZ, 4 BAF (metric)
  ss.ct.vol: Count BAF volume surface, horizontalPointIZ, 4 BAF (metric)
Number of stems = 102
Mean BA surface-based sampling ratio count:bb = 4.9830843
True baf-based sampling ratio count:bb = 5

Please use "summary" on individual components (slots) for more details.


R> ntrees = length(ssBB@ss.bb.vol@izContainer@iZones)
R> baf.c = ssBB@ss.ct.vol@izContainer@iZones[[1]]@angleGauge@baf
R> baf.v = ssBB@ss.bb.vol@izContainer@iZones[[1]]@angleGauge@baf
R> baf.ratio = baf.v/baf.c
R> n.bb = ssBB@ss.bb.ba@surfStats$mean/baf.v
R> n.ct = ssBB@ss.ct.ba@surfStats$mean/baf.c
R> (ss.ratio = n.ct/n.bb)


[1] 4.9830843
```

From the summary above we see that there are 102 trees in the population. Also, a BAF_c of $\mathcal{F}_\text{c} = 4\ \text{m}^2\,\text{ha}^{-1}$ was used on the count surfaces, while a BAF_v of $\mathcal{F}_\text{v} = 20\ \text{m}^2\,\text{ha}^{-1}$ was used on the big BAF surfaces. Finally, the theoretical ratio of count to VBAR trees is given by the ratio of the BAFs; thus, in this case we have an expected ratio of count to volume trees of $20/4 = 5$. However, an *estimate* of the ratio given the sampling surface tract size, resolution (i.e., grid size) and the juxtaposition of the stems in the current population is based on the total number of trees sampled over all points on the count surface to those in the volume surface; this estimate turns out to be 4.983. This is an unbiased estimate and will converge as the cell size decreases (number of points increases), just like other statistics.

The code in the above snippet demonstrates how one can extract different quantities out of an "ssBigBAF" object and use them for the basis of other calculations if desired. Note that all of the above quantities derive from one or more of the individual sampling surfaces. Of course we already know what the value of the two BAFs were, because we created the objects, but the above simply shows how to extract these from the object. There is much more to the individual *sampSurf* objects, which is explained in detail in Gove (2012b). Furthermore, recall that the individual tree inclusion zones are objects within a larger "container" object of class "izContainer" (Gove, 2013a). These zones will be exactly the same for both counts surfaces, since the same BAF_c is used; likewise, they will be exactly the same for both big BAF surfaces due to the same BAF_v having been used. Note that if one were to use the population totals in the above estimated ratio, it would essentially be exact; *viz.*,

```
R> nt.bb = ssBB@ss.bb.ba@surfStats$popTotal/baf.v
R> nt.ct = ssBB@ss.ct.ba@surfStats$popTotal/baf.c
R> nt.ct/nt.bb


[1] 5
```

It will be left as an exercise for the reader to reason out why this is so.

One final comment that may be obvious from the above little analysis. Since each of the four sampling surface slots hold a valid object of class "sampSurf", any methods that can be applied to such objects can be applied to the objects in these four slots. Examples would be using the `plot` and `hist` methods on these individual objects; while summaries of the individual sampling surfaces can be generated by, e.g., `summary(ssBB@ss.ct.vol)` for the count volume surface. In addition, there is a `plot` method that operates on the entire "ssBigBAF" object as shown in the following...

```
R> plot(ssBB, whichSS = c('ss.bb.ba','ss.bb.vol','ss.ct.ba','ss.ct.vol'),
+       useImage = FALSE)
```

Note in Figure 2 that the titles are the slot names from the object by default. The top row shows the BAF_v surfaces for basal area and volume, and the second row the corresponding BAF_c surfaces.

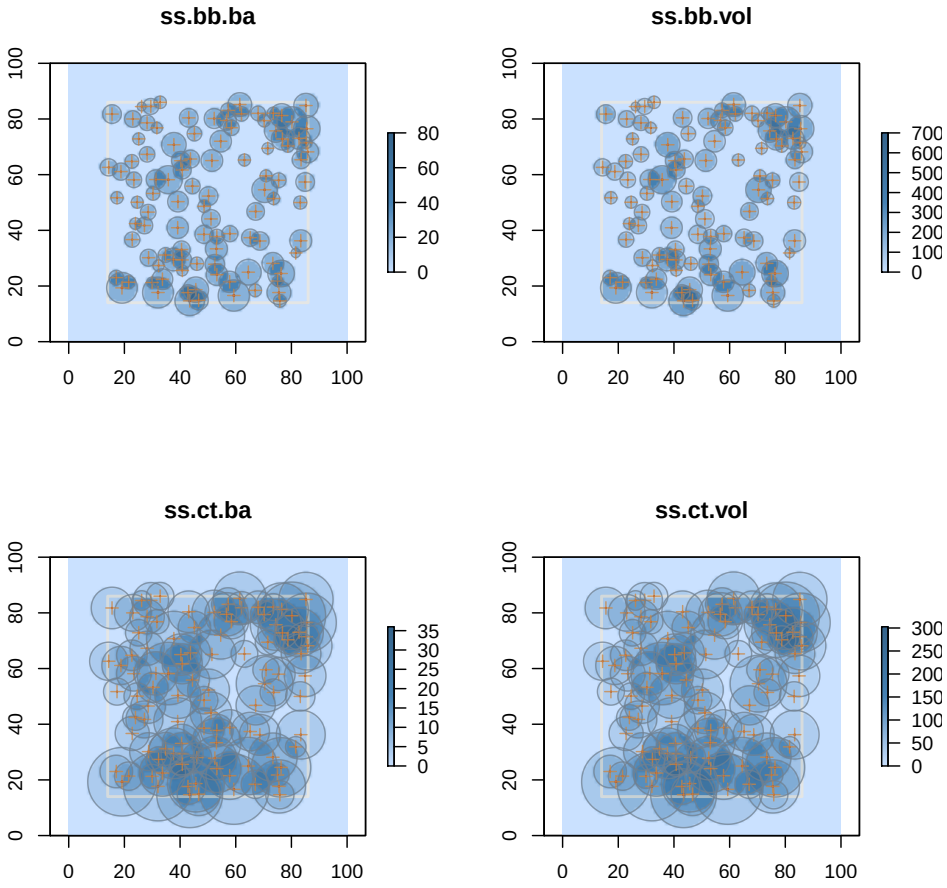


Figure 2: The four sampling surfaces in an example "ssBigBAF" object. The top row displays the basal area and volume surfaces (left to right) for the larger BAF_v. The second row displays the same information but for the smaller BAF_c.

The difference in the number of cells (sample points) covered by inclusion zones is quite apparent between the two BAF's from the figure. It is worth pointing out that the `whichSS` argument allows one to pick the sampling surfaces that are to be displayed by slot name; from one to all four may be chosen with the default being all four displayed in the order in which the slots are stored in the object.[10] This argument name must be specified as the second argument in the `plot` generic function is the "`y`" argument (the first is the "`x`" argument, which corresponds to the "ssBigBAF" object), and in this method developed for objects of class "ssBigBAF" the second argument (while still there in the call to `plot`) is always missing (`NA`). This may take a little pondering.[11] The argument `useImage` derives from the `plot` method for a "Tract" object (see

[10] This is why I have explicitly specified them so that they align both horizontally (by BAF) and vertically (by attribute).

[11] See the argument list in the `plot` generic `?graphics::plot` for the key to this if need be. All methods based on this generic share the first two arguments, others can also be defined, like `whichSS`; e.g., see `methods?ssExtra::plot`.

`methods?sampSurf::plot` and find the "Tract" method); thus we again see how arguments in ... are passed through until resolved in the line of function calls.[12]

A summary of the big BAF volume surface gives[13]...

```
R> zz = sampSurf::summary(ssBB@ss.bb.vol)


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

Inclusion zone objects: horizontalPointIZ
Measurement units = metric
Number of trees = 102
True tree volume = 59.845463 cubic meters
True tree basal area = 7.0688455 square meters
True tree surface area = 1030.6119 square meters
True tree biomass = NA
True tree carbon = NA

Estimate attribute: volume
Surface statistics...
  mean = 60.061252
  bias = 0.21578921
  bias percent = 0.36057739
  sum = 600612.52
  var = 11649.502
  st. dev. = 107.93286
  cv % = 179.70464
  surface max = 700.09283
  total # grid cells = 10000
  grid cell resolution (x & y) = 1 meters
  # of background cells (zero) = 7152
  # of inclusion zone cells = 2848


R> tractArea = area(ssBB@ss.bb.vol@tract)/10000       #in ha
R> ncells = ssBB@ss.bb.vol@surfStats[['nc']]          #same for both
R> ncz.ct = freq(ssBB@ss.ct.vol@tract, 0, digits=15)  #BAFc background cells
```

[12] There you will see that the `plot` method for a "Tract" object is actually using the `plot` method for a "raster" class object, which is the parent class of "Tract"—an on it goes.

[13] The "`sampSurf::`" package qualifier is required in order for this command to run within the `knitr` environment (which creates this document in R); it is not required in general at the command line.

```
R> ncIZ.ct = ncells - ncz.ct                                    #BAFc covered by IZones
```

Hence each tract[14] has an area of 1 ha covered by 10,000 grid cells (i.e., total sample points), 2,848 of which are covered by inclusion zones using BAF_v. Running the same command on the other three objects will show that the same trees with exact same volume, &c., are used in each one. However, the number of cells covered by inclusion zones will be different for the count surfaces, where 6,511 cells are within the zones due to the smaller BAF_c used. The high CV% = 179.7 is evidently due to the small tract and low basal area, yielding large number of zero (background) cells.

The population basal area may seem low for a tract of 1 ha. Remember, however, that there are no trees in the boundary area. The area of the internal "plot" can be derived for this "bufferedTract" object using the same methods as in § 2.1;[15] i.e.,...

```
R> (bb = bbox(ssBB@ss.bb.vol@tract))                            #Tract bounding box

  min max
x   0 100
y   0 100

R> (bw = ssBB@ss.bb.vol@tract@bufferRect['x',1] - bb['x',1]) #buffer width

[1] 14

R> (plotArea.mc = (bb['x',2] - 2*bw)^2)                         #internal area in m^2

[1] 5184

R> (popBA = ssBB@ss.bb.ba@surfStats[['popTotal']])              #total BA of trees

[1] 7.0688455

R> (ba.pha = 10000*popBA/plotArea.mc)                           #ba/ha

[1] 13.63589
```

[14] Recall, one can get a summary of the "Tract" object by typing `ssBB@ss.bb.vol` at the R prompt.

[15] Note that the calculations err on the side of caution; for example, subtracting off the lower bound of the "bufferedTract" bounding box, which is zero—but need not be in general—to get the buffer width.

The internal tract area inside the buffer can be easily calculated as in the above; *viz.*, $(100 - 2 * 14)^2/10000 = 0.52$ ha. Thus, the total basal area on the tract interior, when expanded to a ha^{-1} basis is therefore 13.6 $m^2\,ha^{-1}$; this translates to 59.4 ft^2ac^{-1}, which is very close to the basal area specified in the creation of the `ssBB` "ssBigBAF" object using `createBBNH` in § 5.3.

6 Monte Carlo Sampling

6.1 When is n sufficiently large?

The well-worn title of this section derives from Jim Barrett (e.g. Barrett and Goldsmith, 1976), and perhaps continuing to use it is a nice tribute to Jim.[16] The `sampSurf` package has a set of classes and methods that allow sampling from the population of grid cells comprising the surface. These are derived from the "monte" class and `monte` generic function defined within the package, and include methods for objects of different `S4` classes. Ideally it would have been nice to build on the available class structure and generics for the new double sampling applications (§ 5), but no clear way availed itself to directly inherit from the extant `monte` methods. In addition, a completely new class structure (§ 6) was required in the more complex double sampling context. However, it was possible to create a new method of the `sampSurf::monte` generic for objects of class "ssBigBAF" that allows drawing individual replicate (Monte Carlo) samples from each of the four individual surfaces for a given sample size (e.g., $n = 20, 40, 100, \ldots$). The statistics from the replicate samples such as confidence interval capture rates are available for each sample size. The capture rates facilitate approximating an answer to the sample size question ("when is n sufficiently large?") for the given population and sampling design parameters (i.e., BAFs).

The technical reason why the *sampSurf* "monte" class structure could not be inherited from directly may not be clear at this point. Suffice it to say, without going deeply into the code, that the "monte" class was designed to sample and retain results from a *single* sampling surface. Now, as noted in § 5.2, there are four sampling surfaces to both sample from, and retain results from. Objects of class "monte" use data frames[17] in the slot structure to store the results of a run. However, the multiple sampling surfaces in the double sampling context require `lists` of data frames for the results. In addition, multiple variances were desired to be compared in the double sampling context, whereas only one variance (bootstrapping notwithstanding) is of concern in the single sampling surface runs. Suffice it to say that one can compare the class structure presented below (§ 6.3) with that of the "monte" class structure in *sampSurf* (see, e.g., Gove, 2012a) if one wants to study the differences in more depth—the details are really immaterial to the use of these methods.

[16] For those unfamiliar with it, Jim also wrote a set of `Basic` programs for his book (Barrett and Nutt, 1979), one of which was the program `monte` for answering the question posed in the section title.

[17] Through the subclasses of the "MonteSample" class.

6.2 The R "monteDoubleSampling" Class

The "monteDoubleSampling" class is another virtual class that is used to form the basis for subclasses that will more specifically address different double sampling methods in the MC application of such methods. It is much like the virtual "ssDoubleSampling" discussed in § 5.1 in this sense. The simple class structure is defined as...

```
R> showClass('monteDoubleSampling')


Virtual Class "monteDoubleSampling" [package "ssExtra"]

Slots:

Name:  description    estimate
Class:   character   character

Known Subclasses: "monteBigBAF"
```

6.2.1 Class slots

- `description`: A `character` description of the object.
- `estimate`: A `character` identifier for characterizing the attribute that the sampling surface(s) represent in terms of the double sampling estimator(s). When a subclass object has been instantiated, this slot *must* be a legal attribute from a "sampSurf" object.

Again, inasmuch as this is a virtual class, these are simply placeholders for any subclass objects such as the following.

6.3 "monteBigBAF:" Big BAF Sampling Class

The "monteBigBAF" class builds on the "monteDoubleSampling" class by adding the structure necessary to enable studying the big BAF sampling method through MC experiments. The class structure is...

```
R> showClass('monteBigBAF')


Class "monteBigBAF" [package "ssExtra"]
```

```
Slots:

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

Name:     t.values        boot      numBSS       means        vars      stDevs
Class:     numeric     logical     numeric        list        list        list

Name:     varMeans      stErrs    lowerCIs    upperCIs      caught  otherVarms
Class:        list        list        list        list        list        list

Name:      n.tvbar       corrs        covs      gm.all      sm.all  mc.samples
Class:        list        list        list        list        list        list

Name:  description    estimate
Class:   character   character

Extends: "monteDoubleSampling"
```

6.3.1 Class slots

The number of slots in the "monteBigBAF" class is rather large as can be seen from the output above. The following definitions for each slot are also long and perhaps tedious. They are presented for completeness in documentation, and not for casual reading (as a perusal will quickly verify).

- `mcSamples`: The (scalar) number of MC replications drawn for each sample size, `n`.
- `n`: A vector of samples sizes, n, to be drawn from the population surfaces comprising an "ssBigBAF" object—replicated `mcSamples` times.
- `fpc`: The finite population correction factors for each sample size `n`. The correction is: $f_c = (N - n)/N$. These are calculated and stored, but are not applied at this point because our interest is mainly in the product variances, and it is not at all clear how these relate.
- `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` (the default—see the method arguments).
- `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.

- `t.values`: The $t_{n-1}^{1-\alpha/2}$ Student's t values for each sample size `n` with two-tailed α-level `alpha`.
- `boot`: `TRUE`: include jackknife and bootstrap estimates; `FALSE`: do not include these.
- `numBSS`: The number of bootstrap samples to be drawn from each MC replicate and sample size if `boot = TRUE`.
- `means`: A `list` of data frames each containing the individual means for all `mcSamples` by `length(n)` samples drawn from the population.[18] The following data frames (each a different estimated attribute) are contained in this `list` with component names...

  `vol.bb`: The big BAF sample means for volume; i.e., the volume estimated using the big BAF estimator $\hat{V}_{\mathcal{B}}$ in (4) on the sample points drawn out of the BAF_v surface.

  `ba.bb`: The big BAF sample means for basal area estimated from the big BAF sample points.

  `ba.count`: The count sample means for basal area estimate on the BAF_c count sample points.

  `vol.count`: The count means for volume from the sample points drawn out of the count BAF_c surface. Note that this is the usual volume estimate over all points.

  `tvbar`: The mean tree VBARs for "in" trees on big BAF points, where the mean tree VBAR, $\bar{\mathbb{V}}$, is calculated by (2) from the BAF_v sample points.

  If `boot == TRUE` then the following two components are appended to the above...

  `boot`: The bootstrap mean estimates of the volume $\hat{V}_{\mathcal{B}}$ from (4).

  `jack`: The jackknife mean estimates of the volume $\hat{V}_{\mathcal{B}}$ from (4).

  Note: The next ten slots contain `lists` whose data frames have the same dimensions as the `means` slot data frames above. Note, however, that only some of these have the same component names in the `list`.

- `vars`: A `list` of data frames each containing the individual variances for all `mcSamples` by `length(n)` samples drawn from the population where: $s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(y_i - \bar{y})^2$. Note that the `list` component names are the same as for the `means` slot (just substitute "variance" for "mean" in the slot definitions).
- `stDevs`: A `list` of data frames each containing the individual standard deviations for all `mcSamples` by `length(n)` samples drawn from the population where: $s = \sqrt{s^2}$. Note that the `list` component names are the same as for the `means` slot (just substitute "standard deviation for "mean" in the slot definitions).

[18] In other words the dimensions of each data frame are `mcSamples` rows × `length(n)` columns, where `length(n)` is the number of different sample sizes, n.

- `varMeans`: A `list` of data frames each containing the individual variances of the mean for all `mcSamples` by `length(n)` samples drawn from the population. The following data frames (each a different estimated variance for volume) are contained in this `list` with component names...

  `delta`: The Delta Method product variance estimates (14).

  `goodm`: Goodman's exact product variance estimates (13).

  `delta.r`: The Delta Method product variance estimates given in (16), but using the ratio variance estimator for VBARs given in (10) rather than (5).

  `goodm.r`: Goodman's exact product variance estimates given in (13), but using the ratio variance estimator for VBARs given in (10) rather than (5).

  `vol.bb`: The big BAF sample variances of the mean for volume using the H-T variance for volume, (20), from the BAF_v samples.

  `vol.ct`: The count sample variances of the mean for volume using the H-T variance for volume, (20), from the BAF_c samples.

  If `boot == TRUE` then the following two components are appended to the above...

  `boot`: The bootstrap variance estimates of the volume $\hat{V}_{\mathcal{B}}$ from (4). Recall that the so-called BC_a intervals are calculated exclusively.

  `jack`: The jackknife variance estimates of the volume $\hat{V}_{\mathcal{B}}$ from (4).

  `pbDelta`: The PBDM variance estimates of the volume $\hat{V}_{\mathcal{B}}$ (see § 8 for details).

- `stErrs`: A `list` of data frames each containing the individual standard errors of the mean for all `mcSamples` by `length(n)` samples drawn from the population. Note that the `list` component names are the same as for the `varMeans` slot (just substitute "standard error" for "variance" in the slot definitions).

- `lowerCIs`: A `list` of data frames each containing the individual lower limit confidence interval endpoints for all `mcSamples` by `length(n)` samples drawn from the population. Note that the `list` component names are the same as for the `varMeans` slot (just substitute "lower limit confidence interval endpont" for "variance" in the slot definitions).

- `upperCIs`: A `list` of data frames each containing the individual upper limit confidence interval endpoints for all `mcSamples` by `length(n)` samples drawn from the population. Note that the `list` component names are the same as for the `varMeans` slot (just substitute "upper limit confidence interval endpont" for "variance" in the slot definitions).

- `caught`: A `list` of data frames each containing the individual confidence interval percent catch statistics for all `mcSamples` by `length(n)` samples drawn from the population. Note that the `list` component names are the same as for the `varMeans` slot (just substitute "confidence interval catch statistic" for "variance" in the slot definitions). Please also note that this `list` differs from the preceding `varMeans` through `upperCIs`, in that each data frame in the `caught` list is composed entirely of entries of class "logical," with values `TRUE` if the confidence interval catches the population mean and `FALSE` otherwise.

- `otherVarms`: A `list` of data frames each containing other individual variances of the mean for all `mcSamples` by `length(n)` samples drawn from the population. The following data frames (each a different estimated attribute variance) are contained in this `list` with component names...

  `ba.bb`: The big BAF sample variances of the mean estimates for basal area using BAF_v.

  `ba.ct`: The count sample variance of the mean estimates for basal area using BAF_c.

  `tvbar`: The variance estimates for BAF_v sample tree VBARs, $\widehat{\text{var}}\left(\bar{\mathbb{V}}\right)$, given in (5).

  `tvbar.r`: The ratio variance estimates for BAF_v sample tree VBARs, $\widehat{\text{var}}_\text{R}\left(\bar{\mathbb{V}}\right)$, given in (10).

  `dm.tvbar`: The volume component of the Delta Method variance estimates (16) as given by (18). This uses the tree VBAR variances given in (5).

  `dm.tvbar.r`: The same as `dm.tvbar` above, except using the ratio estimator for the variance of tree VBARs given in (10) rather than (5).

  `dm.ba`: The basal area component of the Delta Method variance estimates (16) as given by (19).

- `n.tvbar`: A `list` of data frames each containing the number of VBAR trees sampled on each individual big BAF sample replicate for all `mcSamples` by `length(n)` samples drawn from the population. The following data frames are contained in this `list` with component names...

  `count`: The number of BAF_c count sample trees, m_c.

  `bb`: The number of BAF_v count sample trees, m_v.

- `corrs`: A `list` of data frames each containing individual sets of correlations computed on different combinations of measurements for all `mcSamples` by `length(n)` samples drawn from the population. The definitions for each of the first set of components listed below are found in § 7. The following data frames (each a different estimated attribute correlation between VBAR and basal area) are contained in this `list` with component names...

  `tvbar.ba`: Based on individual tree VBARs and basal area of the measured trees in the full sample (over all points). These are tree-based, with no expansion. (See item 1 in § 7.)

  `Tvbar.ba`: Point-wise correlation based on aggregate tree-based VBARs and basal area at each sample point. (See item 2 in § 7.)

  `pvbar.ba`: Point-wise correlation on aggregate tree VBARs from the big BAF sample with basal area counts from the BAF_c count sample. (See item 3 in § 7.)

  `mvbar.ba`: Point-wise correlation on tree VBARs from the big BAF sample with mean basal area counts from the BAF_c count sample. (See item 4 in § 7.)

  `Pvbar.ba`: Point-wise correlation on aggregate tree VBARs from the big BAF sample with basal area counts also from the big BAF sample. (See item 5 in § 7.)

  `Mvbar.ba`: Similar to `mvbar.ba`, these are the point-wise correlations on mean tree VBARs and basal area tally (not unexpanded tree basal area), with both from the big BAF sample. (See item 6 in § 7.)

Please see § 8 for details on the following...

`BcVv.cor`: Point-wise correlations for count basal area (`Bc`) vs. big BAF volume (`Vv`) as given in (26).

`BcBv.cor`: Point-wise correlations for count basal area (`Bc`) vs. big BAF basal area (`Bv`) as given in (28).

`VvBv.cor`: Point-wise correlations for big BAF volume (`Vv`) vs. big BAF basal area (`Bv`) as given in (27).

- `covs`: A `list` of data frames each containing individual sets of paired PBDM covariances. The covariances corresponding to the approximate component correlations calculated above in `corrs` (i.e., all but the last three PBDM quantities) are not contained in this `list` object as they are readily calculable from the correlations and respective standard errors. Rather, this `list` contains the following covariances required for the PBDM for all `mcSamples` by `length(n)` samples drawn from the population[19]...

  `BcVv.cov`: Point-wise covariances for count basal area (`Bc`) vs. big BAF volume (`Vv`) as given in (23).

  `BcBv.cov`: Point-wise covariances for count basal area (`Bc`) vs. big BAF basal area (`Bv`) as given in (25).

  `VvBv.cov`: Point-wise covariances for big BAF volume (`Vv`) vs. big BAF basal area (`Bv`) as given in (24).

- `gm.all`: This is a `list` of "grand" means by sample size. Each component is a `matrix` object containing means for rows that are the defined in the `list` objects for the slots above from `means` through `corrs`, while the columns of each matrix are sample size.[20]

- `sm.all`: This is a `list` of two data frames. The data frames contains the sampling variance and standard errors of the means (from the `means` slot as rows), over the sample size, `n`, as columns. These are the "normal theory" estimates of these quantities calculated over all Monte Carlo samples.[21] Note that if `boot = TRUE`, the normal theory estimates for the bootstrap means will also be included here (irrespective of their overall usefulness).

- `mc.samples`: This is a `list` of data frames. Each component of the list corresponds to a sample size from the vector of slots `n`. Thus, the list has length `length(n)`. The rows in each data frame hold the cell (point) numbers that were selected in each of the n individual sample points, sorted in ascending order for a given MC replicate sample. The MC replicates are given by the columns. Thus, the data frames for each `list` component are defined by the sample size for that component, n, for the rows. However, every data frame has `mcSamples` columns of MC replicates. The dimensions of the data frame are therefore `n` rows by `mcSamples` columns, where `n` is different for each `list` component.

[19] For a complete explanation of these quantities, please see § 8.

[20] These means are calculated using the non-exported "hidden" function `ssExtra:::.grandMeans`; see the code for documentation.

[21] These estimates are calculated using the non-exported "hidden" function `ssExtra:::.samplingVarMeans`; see the code for documentation.

6.4 An Example

Having now gone through all the component classes that will allow Monte Carlo subsampling from a sampling surface, it is time to present an example of its use. Here we employ the "ssBigBAF" object created in § 5.3 using the `monte` function to construct an object of class "monteBigBAF"...

```
R> ssBB.mc = monte(ssBB, n = c(25, 50, 75, 100), mcSamples = 250, boot= TRUE,
+                  startSeed = 355)


n=25, 50, 75, 100,
Population of 102 trees...
Population size N = 10000 (number of cells)
Sample size = 25 50 75 100
Sampling fraction n/N = 0.0025 0.005 0.0075 0.01
Samples drawn with replacement
Basal Area...
  Population tree ba = 7.0688455
  mean ss.bb.ba = 7.094
  mean ss.ct.ba ba = 7.07
Volume...
  Population tree volume = 59.845463
  mean ss.bb.vol volume = 60.061252
    %diff pop vol = 0.36057739
  Population mean Vbar = 8.4660873
Correlation (tree vbar & ba)...
  Population on individual trees = 0.59577056
```

The report generated just gives an overview of the populations and requested sampling parameters. We can look at more detail with...

```
R> ssExtra::summary(ssBB.mc)


Object of class: monteBigBAF
------------------------------------------------------------
Big BAF Sampling Monte Carlo Simulation
------------------------------------------------------------
Estimate attribute = volume (except ba.* means)
Sample sizes (n) = 25, 50, 75, 100
Number of Monte Carlo samples = 250
Number of bootstrap samples = 100
```

```
Monte Carlo results...
--Means...
             n.25       n.50       n.75      n.100
vol.bb 60.0651217 59.9834148 60.9472139 60.4027582
ba.bb   7.0112000  7.3376000  7.2405333  7.0496000
ba.ct   7.1340800  7.0944000  7.2200533  7.1329600
vol.ct 60.4033034 59.9112339 61.0125652 60.3359557
tvbar   8.4169643  8.4525121  8.4375671  8.4669344
boot   59.7679374 59.6834729 60.6439443 60.1098792
jack   59.7691330 59.6838686 60.6441077 60.1099539

--Variances of the means...
             n.25       n.50       n.75      n.100
delta   162.50130  81.675133  55.608191  41.997622
goodm   161.90866  81.541172  55.546795  41.960880
delta.r 159.59640  80.891610  55.132101  41.650216
goodm.r 159.13028  80.774427  55.077271  41.617157
vol.bb  446.29334 242.163198 155.981507 115.086404
vol.ct  153.67079  77.420696  52.639024  39.504480
boot    168.99254  80.903716  55.950035  42.220877
jack    166.17213  83.719233  56.926515  42.818144
pbDelta 163.29970  83.292522  56.559350  42.515344

--Standard Errors of the means...
             n.25       n.50       n.75      n.100
delta   12.586523  8.9779703  7.4292139  6.4641942
goodm   12.563476  8.9705674  7.4251015  6.4613596
delta.r 12.469958  8.9338766  7.3973249  6.4373898
goodm.r 12.451711  8.9273805  7.3936361  6.4348285
vol.bb  20.446532 15.3445901 12.3766536 10.6561486
vol.ct  12.246899  8.7446230  7.2301118  6.2702629
boot    12.771463  8.9163342  7.4318077  6.4605169
jack    12.717015  9.0872848  7.5150261  6.5247073
pbDelta 12.602325  9.0658155  7.4921120  6.5023710

--Percent catch...
        n.25 n.50 n.75 n.100
delta   94.0 94.4 95.2  95.6
goodm   94.0 94.4 95.2  95.6
delta.r 93.6 94.0 94.8  95.2
goodm.r 93.6 94.0 94.8  95.2
vol.bb  98.0 99.6 99.6  99.2
vol.ct  95.2 94.0 95.6  94.4
```

```
boot    92.8 94.4 92.4  93.6
jack    94.0 94.0 95.6  95.2
pbDelta 94.4 94.8 94.8  95.2
```

Many other quantities are calculated as described in § 6.3.1 but not listed in the summary above. For example, the following shows the means of a number of other variances that are computed...

```
R> ssBB.mc@gm.all$gm.otherVarms


                  n.25        n.50         n.75        n.100
ba.bb         5.97482667  3.23260082  2.092388709  1.532124444
ba.ct         2.08384853  1.04939729  0.717113350  0.536951402
tvbar         0.29804781  0.13021747  0.086415844  0.068913252
tvbar.r       0.23140070  0.11345622  0.077168942  0.062048862
dm.tvbar     14.31699200  6.42194814  4.470819716  3.475360870
dm.tvbar.r   11.41209881  5.63842548  3.994729628  3.127954329
dm.ba       148.18430328 75.25318458 51.137371611 38.522261435
```

For instance, we can see from the above that the Delta Method variance component (18) (either `dm.tvbar` or `dm.tvbar.r`) is dwarfed by component (19) (in `dm.ba`), which is in accord with what others have pointed out and reported. Also note that the Delta Method variance components, (18) and (19), perfectly partition the full variance, as noted in § 4.3.2...

```
R> ssBB.mc@gm.all$gm.otherVarms['dm.tvbar',] +
+        ssBB.mc@gm.all$gm.otherVarms['dm.ba',]


      n.25       n.50       n.75      n.100
162.501295  81.675133  55.608191  41.997622


R> ssBB.mc@gm.all$gm.varMeans['delta',]


      n.25       n.50       n.75      n.100
162.501295  81.675133  55.608191  41.997622
```

The same can be shown using the ratio estimates for the variance and will be left as a simple exercise.

One convenient display that is automatic with the class structure is a histogram of the sample

means by sample size[22]...

```
R> hist(ssBB.mc, attribute = 'vol.bb', xlab = 'volume')
```

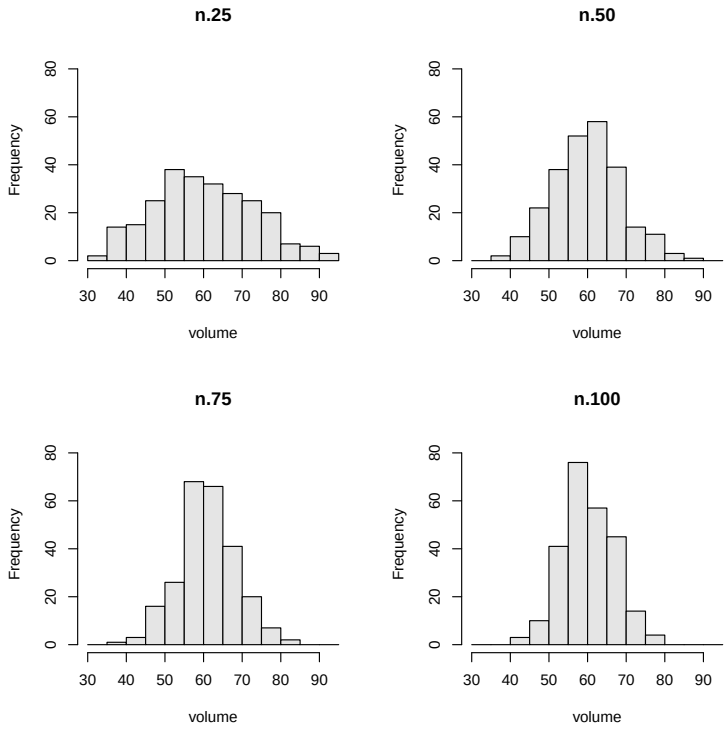


Figure 3: The distribution of volume sample means under Big BAF from repeated sampling of the surfaces shown in Figure 2 by sample size, n.

The result is displayed in Figure 3.

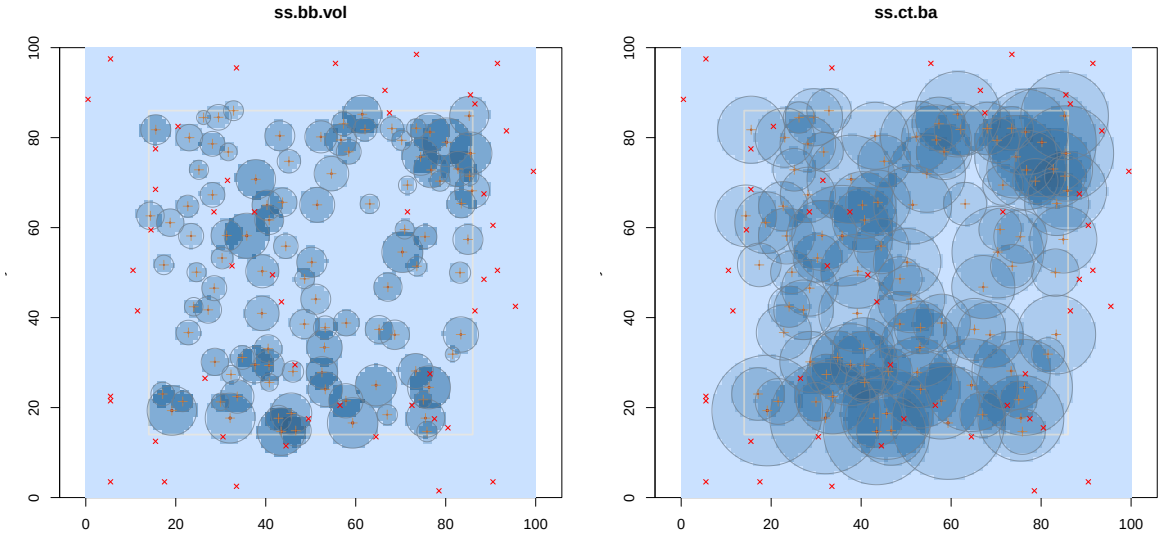
Finally, we can look at any of the individual Monte Carlo replicate draws at a selected sample size for each of the surfaces in Figure 2. Below we choose to look at the two surfaces that contribute to the Big BAF estimate: BAF_v volume and BAF_c basal area. The following command displays the sample points used in the first Monte Carlo sample draw under a sample size of $n = 50$. The results are displayed in Figure 4...

```
R> slotNames(ssBB)[1:4]


[1] "ss.bb.vol" "ss.bb.ba"  "ss.ct.ba"  "ss.ct.vol"
```

[22] We can also look at basal area and VBAR displays.

```
R> plot(ssBB, whichSS = slotNames(ssBB)[c(1,3)],
+       sampleCells = ssBB.mc@mc.samples$n.50$mc.1, point.col = 'red')
```



(a) volume (b) basal area

Figure 4: A re-display of the BAF_v volume (a) and BAF_c basal area (b) surfaces from Figure 2 showing the first Monte Carlo replicate sample of size $n = 50$ ('x').

The above selection of the first MC sample as specified in the `sampleCells` argument will be explained more thoroughly in § 6.4.1; but see also the definition of the "monteBigBAF" class `mc.samples` slot in § 6.3.1.

6.4.1 Viewing individual Monte Carlo simulations

It may be of interest to look a little more deeply at individual Monte Carlo simulation experiments. Some of the code in the previous section makes reference to the `ssBB.mc@mc.samples` slot. As noted in § 6.3.1, this is a `list`[23] object with each element being a data frame of MC sample draws for a given sample size, n. In our case, there are $M = 250$ draws for each sample size. One can look at any of the MC sample draws for a desired sample size, n; these are arranged as columns in a data frame. The data in each column lists the cell (sample point) numbers in the sampling surface that were selected for the sample. Figure 4, for example, displays the sample points for the first MC sample of size $n = 50$, and the code shows how this is extracted from the object. This hierarchy is illustrated in more detail in the following...

[23] The list structure was necessary because of the differing sample sizes that are normally drawn in the simulations.

```
R> ssBB.mc@mcSamples                          #number of MC samples


[1] 250


R> ssBB.mc@n                                  #sample sizes


 n.25  n.50  n.75 n.100
   25    50    75   100


R> str(ssBB.mc@mc.samples, 1)


List of 4
 $ n.25 :'data.frame': 25 obs. of  250 variables:
 $ n.50 :'data.frame': 50 obs. of  250 variables:
 $ n.75 :'data.frame': 75 obs. of  250 variables:
 $ n.100:'data.frame': 100 obs. of  250 variables:


R> head(names(ssBB.mc@mc.samples$n.25))       #a few column names


[1] "mc.1" "mc.2" "mc.3" "mc.4" "mc.5" "mc.6"


R> head(ssBB.mc@mc.samples$n.25[,1:6])        #basic data frame structure


  mc.1 mc.2 mc.3 mc.4 mc.5 mc.6
1   45  106  405  254  283 1712
2  212  491  585 1087 1308 1870
3  418 1333 1810 1347 1547 2105
4  933 1339 2116 1786 1841 2129
5  934 1476 2147 2057 3140 2332
6 1652 2372 2247 2474 3690 2874


R> ssBB.mc@mc.samples$n.25$mc.1               #sampled cell numbers


 [1]   45  212  418  933  934 1652 1661 1797 2176 2186 2767 3196 3235 3497 3511
[16] 3927 4001 4217 4477 6487 7060 8007 9286 9656 9880
```

Note in the above that the individual MC samples are access using the appropriate column name within each sample size-based data frame. Thus, `mc.1` is the first MC sample, and so on, such that the last line above extracts the first MC replicate sample for a sample size of $n = 25$ points (cells), displaying each of the cell numbers in the sample..

6.4.2 Viewing individual Monte Carlo simulation summaries

Having gained some knowledge of the "monteBigBAF" object structure from § 6.3.1 & 6.4.1, we can duplicate some of the computations on individual MC sample draws exactly as computed in the `monte` constructor. There are two main functions that are called in each MC replication within the `monte` constructor to calculate various point- and tree-based summaries as part of the simulations proper. These can be called using the results of the `monte` run to look more closely at *some* of the results. For more information on these methods, see their associated help pages in package `ssExtra`.

Point-wise statistics: The first function, `monteStatsBB`, calculates point-based quantities that are properly *expanded totals* for the tract size and associated basal area factors. In the following we continue to look at the first MC sample draw for $n = 25$...

```
R> msBB.mc1 = monteStatsBB(ssBB.mc@mc.samples$n.25$mc.1, ssBB)
R> str(msBB.mc1, 1)


List of 3
 $ samples :'data.frame': 25 obs. of  7 variables:
 $ stats.ct:'data.frame': 3 obs. of  3 variables:
 $ stats.bb:'data.frame': 3 obs. of  3 variables:


R> (msBB.mc1$samples)


    sdx ba.cts    vol.cts idx.cts ba.bbs   vol.bbs idx.bbs
1    45      0   0.000000   FALSE      0   0.00000   FALSE
2   212      0   0.000000   FALSE      0   0.00000   FALSE
3   418      0   0.000000   FALSE      0   0.00000   FALSE
4   933      4  23.586357    TRUE      0   0.00000   FALSE
5   934      4  23.586357    TRUE      0   0.00000   FALSE
6  1652     12  96.689847    TRUE     20 171.08144    TRUE
7  1661     16 139.261101    TRUE     40 383.93771    TRUE
8  1797      0   0.000000   FALSE      0   0.00000   FALSE
9  2176     24 206.483185    TRUE     60 536.72698    TRUE
```

```
10 2186     20 173.365701    TRUE     40 373.90090    TRUE
11 2767      4  39.352453    TRUE      0   0.00000   FALSE
12 3196      0   0.000000   FALSE      0   0.00000   FALSE
13 3235     12 116.369968    TRUE      0   0.00000   FALSE
14 3497      0   0.000000   FALSE      0   0.00000   FALSE
15 3511      4  30.969008    TRUE      0   0.00000   FALSE
16 3927     12  97.132133    TRUE      0   0.00000   FALSE
17 4001      0   0.000000   FALSE      0   0.00000   FALSE
18 4217      8  55.710877    TRUE      0   0.00000   FALSE
19 4477      8  80.290583    TRUE      0   0.00000   FALSE
20 6487      4  30.610090    TRUE     20 153.05045    TRUE
21 7060     20 193.408308    TRUE      0   0.00000   FALSE
22 8007      0   0.000000   FALSE      0   0.00000   FALSE
23 9286      0   0.000000   FALSE      0   0.00000   FALSE
24 9656      0   0.000000   FALSE      0   0.00000   FALSE
25 9880      0   0.000000   FALSE      0   0.00000   FALSE


R> with(msBB.mc1$samples, vol.bbs[idx.bbs])


[1] 171.08144 383.93771 536.72698 373.90090 153.05045


R> with(msBB.mc1$samples, msBB.mc1$samples[idx.bbs,])


    sdx ba.cts    vol.cts idx.cts ba.bbs   vol.bbs idx.bbs
6  1652     12  96.689847    TRUE     20 171.08144    TRUE
7  1661     16 139.261101    TRUE     40 383.93771    TRUE
9  2176     24 206.483185    TRUE     60 536.72698    TRUE
10 2186     20 173.365701    TRUE     40 373.90090    TRUE
20 6487      4  30.610090    TRUE     20 153.05045    TRUE


R> with(msBB.mc1$samples, ba.cts[idx.cts])


 [1]  4  4 12 16 24 20  4 12  4 12  8  8  4 20
```

The function call extracts the individual sample point and overall summary statistics for both basal area and volume on the count and big BAF samples; the returned `list` component (date frame) names and dimensions are as shown in the second line. The information by the individual sample points is illustrated in the next line. The columns in this data frame list the count basal area and

volume (`ba.cts` & `vol.cts`) with `idx.cts` an indicator of those points (cells) with at least one sampled tree. Moreover, `ba.bbs`, `vol.bbs` and `idx.bbs` have similar definitions for the big BAF sample. The `sdx` column is simply the cell number from the sampling surface corresponding to these quantities.[24] In the above data frame for the full set of sampled cells (points) we can see that no trees were selected (`idx.bbs = FALSE`) on 20 of the BAF_v points, while 11 BAF_c points were no tally (`idx.cts = FALSE`). Furthermore, applying the `idx.bbs` index picks out all of the 5 points from the data frame that had sample trees selected using the $\mathcal{F}_\text{v}$ gauge as shown in the penultimate line. Similarly, there were 14 points with trees selected using the $\mathcal{F}_\text{c}$ gauge as shown in the last line (or simply by counting in the entire data frame).

Notice that no cell-based VBAR information is shown because it is not a particularly useful point-wise quantity under big BAF sampling (as the VBAR statistics are calculated tree-wise).[25] In addition, while we can easily calculate point-wise VBAR quantities from the above point-wise volume and basal area point totals, they are not the correct summaries to use since they are not equivalent to the tree-wise VBARs, unless only one tree per point is sampled. Recall that all quantities returned from `monteStatsBB` are expanded totals as summarized at each grid cell (sample point); thus, such point-wise VBARs are ratios of accumulated total volume and basal area over the trees sampled on each point. To get the correct quantities for (2) we need to dig back into the actual trees sampled at each point to recover the tree-wise VBARs that are required for final volume estimation and variance calculation. Remember, however, that we can alternatively estimate $\bar{\mathbb{V}}$ in (2) from the ratio of total volume to total basal area via $\bar{\mathbb{V}}_\text{R}$ in (9) as shown in G&V, (8.34a) and mentioned above in § 3.1.2; *viz.*,...

```
R> msBB.mc1$stats.bb


             ba          vol        vbar
mean   7.200000    64.747899 8.992763759
var  262.666667 21798.830673 1.279464964
varm  10.506667   871.953227 0.051178599


R> ( vbar.rat = with(msBB.mc1$stats.bb['mean',], vol/ba) )


[1] 8.9927638
```

The equivalence of (9) as calculated above to (2) will be verified below. The `'varm'` (3rd line) for the `'vbar'` is the variance of the VBAR based on this ratio as calculated using (10), as in G&V (8.42), and will be illustrated below. Note that the `msBB.mc1$stats.ct` data frame gives an equivalent summary as shown above, but for the count points.

[24] In case one is wondering about the rather odd name for this column, it stands for "sample point (or cell) index.
[25] There is no VBAR sampling surface, though it can be computed.

One salient point must be emphasized concerning the mean volume returned in the `stats.bb` data frame is that, as demonstrated, it is the mean volume from the points on the individual Monte Carlo draw illustrated above. Let's do a little comparison with what is calculated for the big BAF estimate from the same Monte Carlo sample used above...

```
R> (tvbar25.1 = ssBB.mc@means$tvbar$n.25[1])          #BAFv vbar ratio

[1] 8.9927638

R> (ba.ct25.1 = ssBB.mc@means$ba.ct$n.25[1])          #BAFc basal area estimate

[1] 6.08

R> vol.bb = ba.ct25.1 * tvbar25.1                     #big BAF volume estimate
R> vol.bb25.1 = ssBB.mc@means$vol.bb$n.25[1]          #big BAF volume estimate
R> vol25.1 =  msBB.mc1$stats.bb['mean', 'vol']        #point-wise mean estimate
R> c(vol.bb, vol.bb25.1, vol25.1)

[1] 54.676004 54.676004 64.747899

R> c(ssBB.mc@varMeans$vol.bb$n.25[1], msBB.mc1$stats.bb['varm', 'vol'])

[1] 871.95323 871.95323
```

Note that the estimated volume for this Monte Carlo draw as calculated directly from the sampling surface is 64.7 and is exactly what would normally be computed from the mean of all the point volumes in a typical (non-big BAF) cruise. The variance and variance of the mean in this sample are also the traditional estimates based on the expanded point volumes. On the other hand, the volume estimate given in the `ssBB.mc@means$vol.bb` entry for this Monte Carlo sample is the big BAF estimate, which is shown to be 54.7. It should also be noted that the variance of the mean in `ssBB.mc@varMeans$vol.bb` is, as just noted, the traditional variance of the mean from the point estimates.[26] This, of course, should be logical, if not obvious, since the variance of the mean for the big BAF estimate in `ssBB.mc@means$vol.bb` is given by the Delta Method, Goodman's exact variance, and the PBDM, the latter of which will be discussed in § 8.

[26] That is, it is not the variance of the mean of point-wise big BAF estimates.

Tree-wise statistics: The second function, `monteTreeStatsBB`, provides tree-wise VBAR information by first calculating *unexpanded* tree-wise statistics, and then summarizes these (still unexpanded) by sample point. The following code uses only the tree-wise records on the big BAF sample as noted by the third argument (`stype`) in the function call below (results for the count sample can be calculated in like manner by passing `'count'` to this argument instead of `'bigBAF'`)...

```
R> mtsBB.mc1 = monteTreeStatsBB(ssBB.mc@mc.samples$n.25$mc.1, ssBB, 'bigBAF')
R> str(mtsBB.mc1, 1)                                  #return list components


List of 6
 $ df.trees :'data.frame': 29 obs. of  7 variables:
 $ list.df  :List of 25
 $ df.cells :'data.frame': 25 obs. of  6 variables:
 $ treeStats:'data.frame': 4 obs. of  2 variables:
 $ cellStats:'data.frame': 4 obs. of  2 variables:
 $ n.v      : int 9


R> with(mtsBB.mc1, df.trees[df.trees$idx,])           #trees on tally points


   rep.cdx cellNum tree.id            id       vbar          ba  idx
6        6    1652 tree.73 tree:1pnx54g9  8.5540720 0.079783051 TRUE
7        7    1661 tree.59 tree:qu281wd6  8.9516398 0.042281931 TRUE
8        7    1661 tree.69 tree:u29h7tb0 10.2452457 0.113082749 TRUE
10       9    2176  tree.4 tree:107sfp6i  8.3919467 0.183361796 TRUE
11       9    2176 tree.41 tree:23y0hg1m  9.8381132 0.105588615 TRUE
12       9    2176 tree.44 tree:xyk80p79  8.6062891 0.114125628 TRUE
13      10    2186  tree.4 tree:107sfp6i  8.3919467 0.183361796 TRUE
14      10    2186 tree.12 tree:9a68w1ek 10.3030981 0.141080225 TRUE
24      20    6487  tree.9 tree:4r6go9l3  7.6525225 0.096553775 TRUE


R> with(mtsBB.mc1$df.trees, cor(vbar, ba))            #includes zero points


[1] 0.90584024


R> with(mtsBB.mc1$df.trees, cor(vbar[idx], ba[idx]))  #no zero points


[1] -0.04351607
```

```
R> mtsBB.mc1$treeStats


             vbar             ba
mean  8.992763759  0.11769106275
var   0.857751999  0.00211538760
varm  0.095305778  0.00023504307
corr -0.043516070 -0.04351607045


R> c(sum(mtsBB.mc1$df.trees$idx), mtsBB.mc1$n.v)


[1] 9 9
```

The second line lists the names and class types (with dimensions) of the `list` components returned from the function call. The third line shows the results in a data frame for those trees sampled over all points with at least one tallied tree (`idx = TRUE`). Note that there can be multiple records with the same cell (sample point) number, `cellNum` (this column corresponds to the `sdx` variable in the previous `monteStatsBB` example), due to more than one tree being sampled on these points. The `rep.cdx` variable is a little more complicated, suffice it to say that it keeps track of the same cell that might have been chosen more than once if sampling with replacement. In this way, one can keep track of the trees on the point separately for each visit to the cell. The `tree.id` is the unique tree identifier used in summary `list` or data frame objects, whereas the `id` is the spatial tree identifier that is nested deep within a "standingTree" object. The `idx` column again picks out only those records that have sample trees (and thus excludes all background zero-count cells). The following two lines show the correlation between individual tree $\mathbb{V}$ and B from the sample when all zero-count points are included, and next when only those points with sample trees are included. The latter is the correct measure when one wants simply to look at the correlation in the individual tree $\mathbb{V}$ and B for trees sampled using $\mathcal{F}_{\mathrm{v}}$; this is echoed by the results in the penultimate line—all of these statistics are tree-wise.[27] The last line simply shows the number of trees tallied on all points under $\mathcal{F}_{\mathrm{v}}$ arrived at in two different but equivalent ways.

To validate the estimate of $\bar{\mathbb{V}}_{\mathrm{R}}$ calculated above from the ratio of total volume to total basal area (9), we can look at the mean of the individual tree VBARs, for all trees tallied, $\bar{\mathbb{V}}$, which is exactly what is calculated in (2)...

```
R> mtsBB.mc1$treeStats                              #as above


             vbar             ba
```

[27] Thus, these do *not* include zero-count points since the sample size here is based on tallied trees, not number of sample points.

```
mean  8.992763759  0.11769106275
var   0.857751999  0.00211538760
varm  0.095305778  0.00023504307
corr -0.043516070 -0.04351607045


R> c(vbar.rat, mtsBB.mc1$treeStats$vbar[1])        #compare the two


[1] 8.9927638 8.9927638
```

And the results show the equivalence in practice, as demonstrated algebraically in G&V, (8.34a). A salient point that has been previously noted and bears repeating is that if we want an estimate of the variance of the mean VBAR, $\bar{\mathbb{V}}$ in (2), it is given by (5) based on the individual tree VBARs. And while the ratio of the totals will conveniently give us the same estimated value, it will not provide the required information for calculating a variance of this quantity in the usual manner. However, as noted in § 3.1.2, there is an alternative ratio variance estimator, (10), that can be used for this purpose as given in G&V, (8.42) (using (8.43)). One of these variance estimates (normal theory from the tree vbars, or ratio), is of course required to estimate the overall big BAF variance using either (13) or (16). The following code shows the calculation of the ratio standard error estimate for the current sample of points and compares it with the normal theory estimate from (5)...

```
R> ( vol.bbs = msBB.mc1$samples$vol.bbs )                #point-wise volume


 [1]   0.00000   0.00000   0.00000   0.00000   0.00000 171.08144 383.93771
 [8]   0.00000 536.72698 373.90090   0.00000   0.00000   0.00000   0.00000
[15]   0.00000   0.00000   0.00000   0.00000   0.00000 153.05045   0.00000
[22]   0.00000   0.00000   0.00000   0.00000


R> ( ba.bbs = msBB.mc1$samples$ba.bbs )                  #point-wise ba


 [1]  0  0  0  0  0 20 40  0 60 40  0  0  0  0  0  0  0  0  0 20  0  0  0  0  0


R> ( n = length(ba.bbs) )                                #no. of sample points


[1] 25


R> ( mean.ba = msBB.mc1$stats.bb['mean', 'ba'] )
```

```
[1] 7.2


R> ( r.se = sqrt( sum( (vol.bbs - vbar.rat*ba.bbs)^2 )/( n*(n - 1)*mean.ba^2 ) ) )


[1] 0.22622687


R> ( nt.se = sqrt(mtsBB.mc1$treeStats['varm', 'vbar']) )


[1] 0.30871634
```

The ratio standard error estimate for this particular sample differs by about -26.7% from the normal sample estimate.

The above was made into a little function that facilitates the comparison of these results from any Monte Carlo sample at any sample size; first to verify the above results...

```
R> rv.1 = ratioVariance(ssBB, ssBB.mc, n = 25, mcSample = 1)


Sample size n = 25
Monte Carlo replicate = 1
Ratio variance of the mean = 0.051178599 calculated here
Ratio variance of the mean = 0.051178599 from monteStatsBB
Normal theory varm = 0.095305778


R> rv.1$df


           ratio  normTheory
varm 0.051178599 0.095305778
se   0.226226874 0.308716339


R> rv.1$means


  mean.ba mean.vbar mvbar.rat
7.2000000 8.9927638 8.9927638
```

```
R> msBB.mc1$stats.bb['mean','ba']  #compare to mean.ba above


[1] 7.2
```

are equivalent to those calculated "by hand" in the previous code snippet. The penultimate line of code prints the point-wise mean for basal area (`mean.ba`), $\hat{B}_c$ from (3), which is verified in the last line. Additionally, the tree-wise mean VBAR, $\bar{\mathbb{V}}$ (`mean.vbar`), from (2), and the ratio estimate for the mean VBAR, $\bar{\mathbb{V}}_R$ (`mvbar.rat`), from (9) are presented; the two estimates of the mean VBAR are again shown to be equal to 8.993 for this particular sample.

As a second example, the following presents an illustration of the use of `ratioVariance` with a different sample size, n, and MC draw...

```
R> rv.248 = ratioVariance(ssBB, ssBB.mc, n = 100, mcSample = 248)


Sample size n = 100
Monte Carlo replicate = 248
Ratio variance of the mean = 0.065342157 calculated here
Ratio variance of the mean = 0.065342157 from monteStatsBB
Normal theory varm = 0.081825512


R> rv.248$df


           ratio  normTheory
varm 0.065342157 0.081825512
se   0.255621120 0.286051591


R> rv.248$means


  mean.ba mean.vbar mvbar.rat
7.4000000 8.7574192 8.7574192
```

where we see the variance and standard errors of the means for both the ratio and normal theory sample estimates, along with the means, for Monte Carlo sample replicate #248 with a sample size of $n = 100$. It is also possible with this routine to calculate a ratio estimate from the count sample and compare it with the results from the big BAF sample above; *viz.*,

```
R> crv.248 = ratioVariance(ssBB, ssBB.mc, n = 100, mcSample = 248,
+                          stype = 'count')    #note: for the count sample


Sample size n = 100
Monte Carlo replicate = 248
Ratio variance of the mean = 0.010506318 calculated here
Ratio variance of the mean = 0.010506318 from monteStatsBB
Normal theory varm = 0.012267189


R> crv.248$df


           ratio  normTheory
varm 0.010506318 0.012267189
se   0.102500331 0.110757344


R> crv.248$means


  mean.ba mean.vbar mvbar.rat
 7.400000  8.540938  8.540938
```

In each of the above examples of `ratioVariance` there are two versions of the ratio variance estimate that are printed in the set of results. The first is calculated in this routine based on results from the chosen simulation relicate (i.e., "calculated here"). The second is the estimate as calculated within `monte` from `monteStatsBB`. The `ratioVariance` function was the original test prototype written before the ratio estimates discussed in § 3.1.2 were added to the `monte` system. These in-line calculations have been retained for comparison and are redundant. The code is similar to that used above (in the code chunk that calculates `r.se`) but also demonstrates in more detail how to extract different components from the simulation replications.

The above return object from `monteTreeStatsBB` also summarizes the tree-wise information from the `df.trees` data frame above into a cell-wise (point-wise) data frame `df.cells`. This is illustrated as...

```
R> names(mtsBB.mc1)


[1] "df.trees"  "list.df"   "df.cells"  "treeStats" "cellStats" "n.v"


R> head(mtsBB.mc1$df.cells, 10)
```

```
   rep.cdx cellNum      vbar          ba no.trees   idx
1        1      45  0.000000 0.000000000        0 FALSE
2        2     212  0.000000 0.000000000        0 FALSE
3        3     418  0.000000 0.000000000        0 FALSE
4        4     933  0.000000 0.000000000        0 FALSE
5        5     934  0.000000 0.000000000        0 FALSE
6        6    1652  8.554072 0.079783051        1  TRUE
7        7    1661 19.196886 0.155364680        2  TRUE
8        8    1797  0.000000 0.000000000        0 FALSE
9        9    2176 26.836349 0.403076039        3  TRUE
10      10    2186 18.695045 0.324442020        2  TRUE


R> with(mtsBB.mc1$df.cells, vbar[idx])


[1]  8.5540720 19.1968855 26.8363491 18.6950448  7.6525225


R> with(mtsBB.mc1$df.cells, mtsBB.mc1$df.cells[idx,])


   rep.cdx cellNum       vbar          ba no.trees  idx
6        6    1652  8.5540720 0.079783051        1 TRUE
7        7    1661 19.1968855 0.155364680        2 TRUE
9        9    2176 26.8363491 0.403076039        3 TRUE
10      10    2186 18.6950448 0.324442020        2 TRUE
20      20    6487  7.6525225 0.096553775        1 TRUE


R> mtsBB.mc1$cellStats


            vbar            ba
mean  3.23739495 0.04236878259
var  54.49707668 0.01094505102
varm  2.17988307 0.00043780204
corr  0.96340552 0.96340552308


R> with(mtsBB.mc1$df.cells, cor(vbar, ba))          #include zero-count points


[1] 0.96340552
```

```
R> with(mtsBB.mc1$df.cells, cor(vbar[idx], ba[idx])) #exclude zero-count points


[1] 0.8907168
```

It is important to keep in mind that these are direct accumulations of the tree-level data per cell (sample point); thus, they are again a multi-tree summary, and are *not* expanded to the tract total or per-unit area statistics.[28] In addition to the variables in the `df.trees` data frame, the `df.cells` data frame also includes the `no.trees` column showing how many trees were selected on each point; in this case under big BAF sampling. Recall that the same information for the count sample can be extracted from a separate run of `monteTreeStatsBB` by specifying "`count`" for the third argument in the function call. Note that all of the cell statistics in the `cellStats` data frame include zero-tally points in contrast to the tree-level summaries described previously—the zero-tally excluded version in the above example is presented simply for illustration.

The following are two more ways to compute the correlation, both are based on using the big BAF expanded sample basal area against the unexpanded tree tally VBAR results. The correlations are again based on the big BAF sample of points.[29] The first set of correlations is over all points in the sample, including zero count points; the second set excludes zero-count points...

```
R> #include zero-count points...
R> msBB.mc1$samples$sdx                          #expanded cell numbers


 [1]   45  212  418  933  934 1652 1661 1797 2176 2186 2767 3196 3235 3497 3511
[16] 3927 4001 4217 4477 6487 7060 8007 9286 9656 9880


R> mtsBB.mc1$df.cells$cellNum                    #tally cell numbers


 [1]   45  212  418  933  934 1652 1661 1797 2176 2186 2767 3196 3235 3497 3511
[16] 3927 4001 4217 4477 6487 7060 8007 9286 9656 9880


R> cor(msBB.mc1$samples$ba.bbs, mtsBB.mc1$df.cells$vbar)


[1] 0.99854259


R> #exclude zero-count points
R> with(msBB.mc1$samples, sdx[idx.bbs])          #expanded cell numbers
```

[28] Notably, the basal area used in here is *individual tree* basal area, and not the BAF count.

[29] This is a single replicate using the calculations in item 5, § 7, where zero points are included.

```
[1] 1652 1661 2176 2186 6487


R> with(mtsBB.mc1$df.cells, cellNum[idx])        #tally cell numbers


[1] 1652 1661 2176 2186 6487


R> cor(with(msBB.mc1$samples, ba.bbs[idx.bbs]),
+      with(mtsBB.mc1$df.cells, vbar[idx]))


[1] 0.99416711
```

In each case above, the cell (point) numbers agree between the two sets of data—expanded and tally (unexpanded) as is shown in the two lines preceding each correlation. The cell numbers are shown simply to demonstrate that the same ordered sets of sample points are being used in each case for the correlations.

The following demonstrates that the same results for correlation can be arrived at as in the above, but based solely on the expanded plot-level results from `monteStatsBB` alone...

```
R> with(msBB.mc1$samples, cor(vol.bbs, ba.bbs))             #include zero counts


[1] 0.99854259


R> with(msBB.mc1$samples, cor(vol.bbs[idx.bbs], ba.bbs[idx.bbs])) #exclude zeros


[1] 0.99416711
```

Notice here that we are computing the correlation of expanded volume (not VBAR) with expanded basal area. But expanded volume is a constant (i.e., $\mathcal{F}_v A/n$) multiplied by the sum of the VBARs over all sampled trees (i.e., the summations in (2)); thus, it is exactly the same as was used in the previous code segment in correlating the raw tree VBARs against expanded basal area.[30] More results on correlations from the entire set of Monte Carlo samples are found in § 7.

[30] Scale factors matter in covariance, but they cancel in correlation.

7 Correlations Between VBAR and Basal area

Both the Delta Method and the exact variance assume independence of the two random variables (or estimates), in this case VBAR and basal area. As noted in Gove et al. (2020, see their appendix for details), both the Delta Method and Goodman's exact variance take an alternative form incorporating extra covariance terms in the case where the two variables are not independent. Therefore it is of some interest to know whether one can assume independence so that the correct form of the variance in each case can be used. If we judge independence via the correlation, $\rho(x, y)$, then this can be determined in the simulations.

We have a bit of a conundrum because mean VBAR, $\bar{\mathbb{V}}$, and it's variance are calculated on an individual tree basis (i.e., over all sample trees (2) & (5)), but mean basal area, $\hat{B}_{\mathrm{c}}$, and its variance are calculated on a point-wise basis (e.g., (6)). Thus, unlike many sampling applications where everything is point-based for summation, here we have disparate sample units and sizes based on the number of VBAR trees and the number of basal area points. Why the conundrum? Well, because while we could use the standard deviations associated with (5) and (6) in the denominator of the correlation coefficient, the covariance in the numerator must be based on the same sample support. This suggestion, even if it were to work, is not serious, only meant to illustrate the problem of the disagreement in what forms the sample units between the two quantities and form the basis for the estimation of both Delta Method and Goodman's exact variances. In addition, the two extended formulæ for the variance in this case also contain covariance terms that must be evaluated, and so the same caveat applies to the actual calculation to both of these extended versions of the variance. This seems to be a unique application.

Having noted the above caveat, there are several potential ways to calculate the correlations based on the data available from the simulations, though these are not exhaustive. These are cross referenced below to the `list` items in the `corrs` slot in § 6.3.1. They are as follows...

1. *Big BAF individual tree-wise:* Consider tree VBAR and BA as the random variables. From (1) this would be the individual tree $\mathbb{V}_i$ and b_i values for $i = 1, \ldots, m_{\mathrm{v}}$. This is not as odd as it may seem since it forms the basis for (2), but it does not capture point-wise sampling variability. The correlation would thus be $\hat{\rho}(\mathbb{V}_i, b_i)$ over all trees (m_{v}) in the big BAF measured sample. (See: `corrs$tvbar.ba`.)

2. *Big BAF aggregate individual tree point-wise:* This method considers the individual point estimates themselves as random variables over the sample point draw for each Monte Carlo sample. This uses the same unexpanded individual tree information from the big BAF sample

as item 1, but is based on the point-wise aggregate values of VBAR and basal area; thus,...

$$\mathbb{V}_{\mathrm{v}_s} = \sum_{i=1}^{m_{\mathrm{v}_s}} \mathbb{V}_i$$

$$b_{\mathrm{v}_s} = \sum_{i=1}^{m_{\mathrm{v}_s}} b_i$$

and recall that the subscripts v_s denotes the big BAF sample on the sth point so that, e.g., m_{v_s} is the number of trees on the sth point for the big BAF sample. Note in the above that both quantities will be zero on all count points where no big BAF trees have been measured. The correlation estimate will be $\hat{\rho}(\mathbb{V}_{\mathrm{v}_s}, b_{\mathrm{v}_s})$ for $s = 1, \ldots, n$. (See: `corrs$Tvbar.ba`.)

3. *Count-Big BAF point-wise totals:* This method again considers the individual point estimates as the random variables as in item 2. Here the VBARs are the individual tree aggregates, $\mathbb{V}_{\mathrm{v}_s}$, from the big BAF sample and the basal area is from the expanded count sample. The point-wise *total* basal area estimates are thus...

$$B_{\mathrm{c}_s} = \mathcal{F}_{\mathrm{c}} m_{\mathrm{c}_s}$$

where m_{c_s} is the number of trees on the sth point for the count sample. Thus, the correlation estimate will be $\hat{\rho}(\mathbb{V}_{\mathrm{v}_s}, B_{\mathrm{c}_s})$ for $s = 1, \ldots, n$.[31] (See: `corrs$pvbar.ba`.)

4. *Count-Big BAF point-wise means:* Similar to the last point, we can look at the point-wise per-tree means. These are given as...

$$\bar{\mathbb{V}}_{\mathrm{v}_s} = \frac{1}{m_{\mathrm{v}_s}} \sum_{i=1}^{m_{\mathrm{v}_s}} \mathbb{V}_i$$

$$\bar{B}_{\mathrm{c}_s} = \mathcal{F}_{\mathrm{c}}$$

Note, of course, that $\bar{B}_{\mathrm{c}_s}$ is constant at each point or zero. This will diminish the correlation somewhat. It follows that the correlation estimate will be $\hat{\rho}\big(\bar{\mathbb{V}}_{\mathrm{v}_s}, \bar{B}_{\mathrm{c}_s}\big)$ for $s = 1, \ldots, n$. And again, we will have $\bar{\mathbb{V}}_{\mathrm{v}_s} = 0$ on non-big BAF count points. (See: `corrs$mvbar.ba`.)

5. *Big BAF point-wise totals:* This method is similar to item 3 above in that it considers the individual point estimates themselves as random variables over the sample point draw for each Monte Carlo sample. However, the correlation here is between VBAR and the big BAF basal area tally from $\mathcal{F}_{\mathrm{v}}$. The point-wise total estimates are...

$$\mathbb{V}_{\mathrm{v}_s} = \sum_{i=1}^{m_{\mathrm{v}_s}} \mathbb{V}_i$$

$$B_{\mathrm{v}_s} = \mathcal{F}_{\mathrm{v}} m_{\mathrm{v}_s}$$

Thus, the correlation estimate will be $\hat{\rho}(\mathbb{V}_{\mathrm{v}_s}, B_{\mathrm{v}_s})$ for $s = 1, \ldots, n$. (See: `corrs$Pvbar.ba`.)

[31] Recall that expanded totals are the default in *sampSurf*, so the basal area is expanded by A both here and in items 4, 5 and 6.

6. *Big BAF point-wise means:* Similar to item 4, we can look at the point-wise per-tree means for the big BAF tally as in item 5. These are given as...

$$\bar{\mathbb{V}}_{\mathrm{v}s} = \frac{1}{m_{\mathrm{v}s}} \sum_{i=1}^{m_{\mathrm{v}s}} \mathbb{V}_i$$

$$\bar{B}_{\mathrm{v}s} = \mathcal{F}_{\mathrm{v}}$$

   Note again, of course, that $\bar{B}_{\mathrm{v}s}$ is constant at each point or zero. It follows that the correlation estimate will be $\hat{\rho}\big(\bar{\mathbb{V}}_{\mathrm{v}s}, \bar{B}_{\mathrm{v}s}\big)$ for $s = 1, \ldots, n$. (See: `corrs$Mvbar.ba`.)

7. *Monte Carlo replication-based:* The final method would be to consider each MC replication at each sample size as an estimate of $\bar{\mathbb{V}}_m$ and $\bar{B}_m$ where $m = 1, \ldots, M$, the number of MC sample replicates, and determine the correlation for each sample size from these means. This is a correlation between VBAR and basal area estimates among the repeated samples drawn from the sampling surface and is applied to each sample size. These are not calculated or stored by `monte` as they are not really useful for trying to deduce the correlation of interest, they may be easily calculated as in the example that follows.

Note in particular that there is a possibility of generating a warning in the simulations for any of items 3–6 (but especially the mean versions) in computing the correlation for a given MC draw because the standard deviation could be zero[32]. This would be the case, for example, if exactly one tree were sampled on each point for either the big BAF or count sample so that the basal areas are all equal to the BAF, inducing zero variance over the points. This is *very* infrequent, but can happen for small n and should not be a concern as the number of MC draws is large.

The first six correlations above are calculated automatically in the results of the MC simulations using `monte`. Thus, from the results above we have the following 'grand' averages over all simulations for each quantity...

```
R> ssBB.mc@gm.all$gm.corrs


               n.25       n.50       n.75      n.100
tvbar.ba 0.48601556 0.52165768 0.52616361 0.53366921
Tvbar.ba 0.93271434 0.93371281 0.92962876 0.92846714
pvbar.ba 0.56941734 0.60167837 0.59539842 0.59064744
mvbar.ba 0.44052903 0.45477442 0.44944859 0.45267165
Pvbar.ba 0.98771462 0.98754870 0.98717385 0.98674540
Mvbar.ba 0.98376867 0.98250036 0.98172696 0.98097457
BcVv.cor 0.56941734 0.60167837 0.59539842 0.59064744
BcBv.cor 0.57388086 0.60453169 0.59979847 0.59698414
VvBv.cor 0.98771462 0.98754870 0.98717385 0.98674540
```

[32] `In cor(m.vb.bb, m.ba.ct) : the standard deviation is zero.`

```
R> with(ssBB.mc@means, mapply(cor, tvbar, ba.ct))


      n.25       n.50       n.75      n.100
0.02376513 0.05529851 0.11813149 0.04711983
```

In the above output, the following recap what each row summarizes...

1. `tvbar.ba` displays the tree-wise correlations, $\hat{\rho}(\mathbb{V}_i, b_i)$, corresponding to item 1 in the above list.
2. `Tvbar.ba` are the point-wise tree-based correlations $\hat{\rho}(\mathbb{V}_{\mathrm{v}_s}, b_{\mathrm{v}_s})$ as defined in item 2.
3. `pvbar.ba` gives the point-wise correlation, $\hat{\rho}(\mathbb{V}_{\mathrm{v}_s}, B_{\mathrm{c}_s})$, described in item 3.
4. `mvbar.ba` shows the results for item 4: $\hat{\rho}\big(\bar{\mathbb{V}}_{\mathrm{v}_s}, \bar{B}_{\mathrm{c}_s}\big)$.
5. `Pvbar.ba` gives the big BAF-related correlations in item 5: $\hat{\rho}(\mathbb{V}_{\mathrm{v}_s}, B_{\mathrm{v}_s})$.
6. `Mvbar.ba` gives 6: $\hat{\rho}\big(\bar{\mathbb{V}}_{\mathrm{v}_s}, \bar{B}_{\mathrm{v}_s}\big)$

The results of the second line of code above give the estimated correlations using the method described in item 7. The correlation between VBAR and BA for the population of trees used in the simulations is calculated as $\rho(\mathbb{V}_i, b_i) = 0.5958$. This can be compared against the above results.

7.1 Correlation approximation summary

The interesting thing about all the above potential correlations is that *none* of them are exactly what are needed to employ the extended (non-independent) version of either product variance formula. Again, this is because of the so-called conundrum that was mentioned earlier with regard to the two incommensurate support bases: trees and points. Gove et al. (2020) discuss this in more detail, and an elegant solution to this problem is discussed in § 8.

8 The Point-Based Delta Method

This section presents a new variance estimator that is based on the Delta Method. The estimator provides a clean way to calculate covariances and correlations, has the same efficiency as Bruce's method, and requires only a little extra computational work. A more detailed explanation regarding derivation and use of the formulæ for the variance estimator given in (22) below is provided in Lynch et al. (2020). We have called this section the "point-based Delta Method" because of the

fact that it is based completely on point-wise estimates unlike Bruce's method. This method came about through an insightful observation by Dr. Tom Lynch regarding the Delta Method application discussed in Gove et al. (2020) (and above), and it is hoped that it will come to be referred to as "Lynch's method" in the future of big BAF sampling.

Recall that the big BAF estimator for volume, (4), can be written as (7)...

$$\hat{V} = \hat{B}_{\text{c}} \left(\frac{\hat{V}_{\text{v}}}{\hat{B}_{\text{v}}} \right) \tag{21}$$

In this form it can be seen that the right-hand side is composed of three random variables; *viz.*,

$\hat{B}_{\text{c}}$: the basal area estimate from the count (BAF_{c}) sample points.

$\hat{V}_{\text{v}}$: the volume estimate from the big BAF (BAF_{v}) sample points.

$\hat{B}_{\text{v}}$: the basal area estimate from the big BAF (BAF_{v}) sample points.

As such, the Delta Method can be applied to this equation in terms of these three variables, instead of the normal interpretation of two ($\hat{B}_{\text{c}}$ and $\bar{\mathbb{V}}$) given in the preceding sections, which leads naturally to Bruce's method, as we have seen.

The three-fold variance estimator for (21) is (Lynch et al., 2020)...

$$\begin{aligned} \widehat{\text{var}}_{\text{L}}\left(\hat{V}_{\mathcal{B}}\right) = &\left(\frac{\hat{V}_{\text{v}}}{\hat{B}_{\text{v}}}\right)^2 \widehat{\text{var}}\left(\hat{B}_{\text{c}}\right) + \left(\frac{\hat{B}_{\text{c}}}{\hat{B}_{\text{v}}}\right)^2 \widehat{\text{var}}\left(\hat{V}_{\text{v}}\right) + \left(\frac{\hat{V}_{\text{v}}\hat{B}_{\text{c}}}{\hat{B}_{\text{v}}^2}\right)^2 \widehat{\text{var}}\left(\hat{B}_{\text{v}}\right) + \\ &2\left(\frac{\hat{V}_{\text{v}}}{\hat{B}_{\text{v}}}\right)\left(\frac{\hat{B}_{\text{c}}}{\hat{B}_{\text{v}}}\right) \widehat{\text{cov}}\left(\hat{B}_{\text{c}}, \hat{V}_{\text{v}}\right) - 2\left(\frac{\hat{V}_{\text{v}}\hat{B}_{\text{c}}}{\hat{B}_{\text{v}}^2}\right)\left(\frac{\hat{V}_{\text{v}}}{\hat{B}_{\text{v}}}\right) \widehat{\text{cov}}\left(\hat{B}_{\text{c}}, \hat{B}_{\text{v}}\right) - \\ &2\left(\frac{\hat{V}_{\text{v}}\hat{B}_{\text{c}}}{\hat{B}_{\text{v}}^2}\right)\left(\frac{\hat{B}_{\text{c}}}{\hat{B}_{\text{v}}}\right) \widehat{\text{cov}}\left(\hat{V}_{\text{v}}, \hat{B}_{\text{v}}\right) \end{aligned} \tag{22}$$

where $\widehat{\text{var}}\left(\hat{B}_{\text{c}}\right)$ is given by (6). In addition we have...

$$\widehat{\text{var}}\left(\hat{V}_{\text{v}}\right) = \frac{1}{n(n-1)} \sum_{s=1}^{n} \left(\hat{V}_{\text{v}_s} - \hat{V}_{\text{v}}\right)^2$$

and

$$\widehat{\text{var}}\left(\hat{B}_{\text{v}}\right) = \frac{1}{n(n-1)} \sum_{s=1}^{n} \left(\mathcal{F}_{\text{v}} m_{\text{v}_s} - \hat{B}_{\text{v}}\right)^2$$

The three required covariance terms are computed similarly as...

$$\widehat{\text{cov}}\left(\hat{B}_{\text{c}}, \hat{V}_{\text{v}}\right) = \frac{1}{n(n-1)} \sum_{s=1}^{n} \left(\mathcal{F}_{\text{c}} m_{\text{c}_s} - \hat{B}_{\text{c}}\right) \left(\hat{V}_{\text{v}_s} - \hat{V}_{\text{v}}\right) \tag{23}$$

$$\widehat{\text{cov}}\left(\hat{B}_{\text{v}}, \hat{V}_{\text{v}}\right) = \frac{1}{n(n-1)} \sum_{s=1}^{n} \left(\mathcal{F}_{\text{v}} m_{\text{v}_s} - \hat{B}_{\text{v}}\right) \left(\hat{V}_{\text{v}_s} - \hat{V}_{\text{v}}\right) \tag{24}$$

$$\widehat{\text{cov}}\left(\hat{B}_{\text{v}}, \hat{B}_{\text{c}}\right) = \frac{1}{n(n-1)} \sum_{s=1}^{n} \left(\mathcal{F}_{\text{v}} m_{\text{v}_s} - \hat{B}_{\text{v}}\right) \left(\mathcal{F}_{\text{c}} m_{\text{c}_s} - \hat{B}_{\text{c}}\right) \tag{25}$$

The remaining quantities are as defined in § 3.

The above set of formulæ may at first glance look somewhat formidable. But variances and covariances are rudimentary formulæ that are easily calculable from the data that is already available in a big BAF cruise—and they only need to be programmed once if used in practice. The overall Lynch's method variance, $\widehat{\text{var}}_{\text{L}}\left(\hat{V}_{\mathcal{B}}\right)$, is simply a combination of these in a very straightforward manner.

The briefly noted salient benefit to the above approach is the fact that all of the above quantities are now aggregated on a point-wise basis; this is easily verified by regarding the summations in the above. Thus, given the caveat concerning the approximate correlations calculated in § 7, what we have now is a set of true point-wise covariances (and their associated correlations) that do not require any algebraic trickery nor assumptions to calculate; nor is there any problem with interpretation of any of the above quantities.

The correlations corresponding to the covariances in (23)–(25) are calculated in the usual way and are given as...

$$\hat{\rho}\left(\hat{B}_{\text{c}}, \hat{V}_{\text{v}}\right) = \frac{\widehat{\text{cov}}\left(\hat{B}_{\text{c}}, \hat{V}_{\text{v}}\right)}{\widehat{\text{se}}\left(\hat{B}_{\text{c}}\right) \widehat{\text{se}}\left(\hat{V}_{\text{v}}\right)} \tag{26}$$

$$\hat{\rho}\left(\hat{B}_{\text{v}}, \hat{V}_{\text{v}}\right) = \frac{\widehat{\text{cov}}\left(\hat{B}_{\text{v}}, \hat{V}_{\text{v}}\right)}{\widehat{\text{se}}\left(\hat{B}_{\text{v}}\right) \widehat{\text{se}}\left(\hat{V}_{\text{v}}\right)} \tag{27}$$

$$\hat{\rho}\left(\hat{B}_{\text{v}}, \hat{B}_{\text{c}}\right) = \frac{\widehat{\text{cov}}\left(\hat{B}_{\text{v}}, \hat{B}_{\text{c}}\right)}{\widehat{\text{se}}\left(\hat{B}_{\text{v}}\right) \widehat{\text{se}}\left(\hat{B}_{\text{c}}\right)} \tag{28}$$

In each case the above standard errors are simply the square root of the respective variances $\widehat{\text{var}}\left(\hat{B}_{\text{c}}\right)$, $\widehat{\text{var}}\left(\hat{V}_{\text{v}}\right)$, and $\widehat{\text{var}}\left(\hat{B}_{\text{v}}\right)$.

8.1 An Example

The example in § 6.4 is continued here, where we will provide more explanation on the `pbDelta` standard errors shown there, but deferred to this section. Here we will again use the summary statistics results for the first Monte Carlo sample run with $n = 25$. For the sake of clarity, it is probably best to calculate the PBDM variance estimate $\widehat{\text{var}}_{\text{L}}\left(\hat{V}_{\mathcal{B}}\right)$ from (22) in small pieces to make it simpler to understand.

First, we define the point-based sample mean quantities, $(\hat{B}_{\text{c}}, \hat{B}_{\text{v}}, \hat{V}_{\text{v}})$, from the summary statistics run above; *viz.*,

```
R> Bc = ba.ct25.1                                  #previously defined
R> Bv = ssBB.mc@means$ba.bb$n.25[1]                #BAFb basal area estimate
R> Vv = vol25.1                                    #previously defined
R> c(Bc, Bv, Vv)


[1]  6.080000  7.200000 64.747899
```

Recall that these are simply the usual HPS point-based means as one would normally calculate them from a cruise, with the only difference that we are using two BAFs.

Next, a quick refresher on where some of the variances of the mean can be found in the "monteBigBAF" object. First, the various Delta Method and related variances are found in the `varMeans` slot...

```
R> str(ssBB.mc@varMeans, 1)


List of 9
 $ delta  :'data.frame': 250 obs. of  4 variables:
 $ goodm  :'data.frame': 250 obs. of  4 variables:
 $ delta.r:'data.frame': 250 obs. of  4 variables:
 $ goodm.r:'data.frame': 250 obs. of  4 variables:
 $ vol.bb :'data.frame': 250 obs. of  4 variables:
 $ vol.ct :'data.frame': 250 obs. of  4 variables:
 $ boot   :'data.frame': 250 obs. of  4 variables:
 $ jack   :'data.frame': 250 obs. of  4 variables:
 $ pbDelta:'data.frame': 250 obs. of  4 variables:


R> head(with(ssBB.mc@varMeans, cbind(delta, pbDelta)))  #delta & pt-based Delta
```

```
          n.25      n.50      n.75     n.100      n.25      n.50      n.75
mc.1 185.18903 73.410252 47.191225 58.074447 193.45021 73.652775 48.852289
mc.2 128.23126 97.206021 62.151868 46.581830 149.05531 98.616165 64.127529
mc.3 122.04863 46.106039 65.544704 46.610749 118.53534 50.118537 68.749759
mc.4 132.96637 69.722019 43.763732 42.437295 118.01864 76.463680 48.380001
mc.5 152.82770 94.621684 50.600789 44.074990 158.59765 97.933457 50.441779
mc.6 134.54721 93.109746 53.878623 34.205645 127.50758 95.585927 52.933315
         n.100
mc.1 59.472929
mc.2 52.560179
mc.3 49.108425
mc.4 43.377752
mc.5 44.227548
mc.6 36.346457


R> head(ssBB.mc@varMeans$vol.bb)


          n.25      n.50      n.75      n.100
mc.1 871.95323 188.64813 145.90751 142.261942
mc.2 232.64716 316.38147 156.60849 157.194602
mc.3 221.57492 230.23919 173.75158 154.060023
mc.4 214.37048 312.62602 146.71897 121.151766
mc.5 224.37004 292.98670 231.29779 127.453213
mc.6 375.79408 285.21960 145.99052  73.325311
```

The first line shows the entire `list` of data frames. The second line compares the Delta Method variance estimates (first four columns) with those of Lynch's method (last four columns) for the first few Monte Carlo replicates over all sample sizes. The last line shows the point sample variance of the mean estimates of the big BAF volume estimates for the first few Monte Carlo samples over all sample sizes; please see the **R** code snippet under the "point-wise statistics" in § 6.4.2 for more information on these estimates.

The estimated variances of the means for basal area are also required...

```
R> str(ssBB.mc@otherVarms, 1)


List of 7
 $ ba.bb     :'data.frame': 250 obs. of  4 variables:
 $ ba.ct     :'data.frame': 250 obs. of  4 variables:
 $ tvbar     :'data.frame': 250 obs. of  4 variables:
 $ tvbar.r   :'data.frame': 250 obs. of  4 variables:
```

```
 $ dm.tvbar  :'data.frame': 250 obs. of  4 variables:
 $ dm.tvbar.r:'data.frame': 250 obs. of  4 variables:
 $ dm.ba     :'data.frame': 250 obs. of  4 variables:


R> head(ssBB.mc@otherVarms$ba.bb)


          n.25      n.50      n.75     n.100
mc.1 10.5066667 2.2987755 1.8412012 1.8662626
mc.2  3.6266667 4.2971429 1.9661261 2.0541414
mc.3  2.6666667 3.1869388 2.2332733 2.0117172
mc.4  3.0400000 4.6400000 1.8008408 1.5268687
mc.5  3.0400000 3.8400000 3.1961562 1.6464646
mc.6  5.4400000 3.5134694 1.8969369 0.9579798
```

The first line refreshes us on the fact that the two basal area variances (of the means) are stored in the `otherVarms` slot, while the last line shows the first few Monte Carlo entries for the HPS variance of the mean estimates for big BAF basal area using BAF_{v}.

The following code extracts the variances previously defined as $\widehat{\text{var}}\left(\hat{B}_{\text{c}}\right)$, $\widehat{\text{var}}\left(\hat{B}_{\text{v}}\right)$ and $\widehat{\text{var}}\left(\hat{V}_{\text{v}}\right)$, respectively...

```
R> (Bc.varm = ssBB.mc@otherVarms$ba.ct$n.25[1])  #BAFc variance of the mean ba


[1] 2.2464


R> (Bv.varm = ssBB.mc@otherVarms$ba.bb$n.25[1])  #BAFv variance of the mean ba


[1] 10.506667


R> (Vv.varm = ssBB.mc@varMeans$vol.bb$n.25[1])   #BAFv variance of the mean volume


[1] 871.95323
```

The last estimate for $\widehat{\text{var}}\left(\hat{V}_{\text{v}}\right) = 871.95$ was shown in the code snippet above.

At this point, we have the necessary information to calculate the first part of the variance up to

the covariance terms—this might be termed the 'independent' variance component...

```
R> Vv.Bv = Vv/Bv
R> Bc.Bv = Bc/Bv
R> VvBc.Bv = Vv*Bc/(Bv^2)
R> (pbDelta.ind = Vv.Bv^2 * Bc.varm + Bc.Bv^2 * Vv.varm + VvBc.Bv^2 * Bv.varm)


[1] 1409.3335


R> sqrt(pbDelta.ind)


[1] 37.541091
```

As we can see, this estimate is quite high, because it is not meant to be used alone: the covariances adjust it and are required for the correct variance estimate. The above is simply a convenient intermediate step in the calculation that helps to illustrate the full computation, and is also found in the package **R** code.

The covariances are all calculated according to the formulæ in equations (23)–(25). These are given in the `covs` slot of the "monteBigBAF" object...

```
R> str(ssBB.mc@covs, 1)


List of 3
 $ BcVv.cov:'data.frame': 250 obs. of  4 variables:
 $ BcBv.cov:'data.frame': 250 obs. of  4 variables:
 $ VvBv.cov:'data.frame': 250 obs. of  4 variables:


R> head(ssBB.mc@covs$BcBv.cov)


           n.25       n.50       n.75      n.100
mc.1 3.50933333 0.83591837 0.48470871 0.80913131
mc.2 1.07200000 1.47722449 0.80663063 0.62028283
mc.3 0.40000000 0.91689796 0.94481682 0.76072727
mc.4 1.78666667 1.72473469 0.43089489 0.62254545
mc.5 0.91733333 1.49289796 0.87466667 0.60929293
mc.6 2.24533333 1.32832653 0.65037838 0.34892929
```

The first line above shows the set of data frames, one for each covariance over all Monte Carlo samples and sample sizes. The second line shows the first few Monte Carlo replication results for the covariance $\widehat{\text{cov}}\left(\hat{B}_{\text{v}}, \hat{B}_{\text{c}}\right)$[33].

Finally, we are in a position to add the covariance portion of the equation and calculate the full PBDM variance estimate $\widehat{\text{var}}_{\text{L}}\left(\hat{V}_{\mathcal{B}}\right)$ in (22) for Lynch's method as...

```
R> BcBv.cov = ssBB.mc@covs$BcBv.cov['mc.1', 'n.25']
R> BcVv.cov = ssBB.mc@covs$BcVv.cov['mc.1', 'n.25']
R> VvBv.cov = ssBB.mc@covs$VvBv.cov['mc.1', 'n.25']
R> c(BcBv.cov, BcVv.cov, VvBv.cov)


[1]  3.5093333 32.2099454 95.5752985


R> pbDelta = pbDelta.ind + 2*Vv.Bv*Bc.Bv * BcVv.cov -
+            2*VvBc.Bv*Vv.Bv * BcBv.cov -
+            2*VvBc.Bv*Bc.Bv * VvBv.cov
R> c(pbDelta, ssBB.mc@varMeans$pbDelta['mc.1', 'n.25'])   #final comparison


[1] 193.45021 193.45021
```

The results in the last line above agree with those given in the Monte Carlo simulation results. Again, while the computations for (22) may appear initially to be tedious and difficult, viewed as a set of just a few steps, they are quite simple to implement in practice. Additionally, the calculations require nothing beyond simple statistics that, with the exception of the covariances, would normally be performed in a general cruise, augmented by the dual BAF nature of big BAF sampling.

It was noted above that the covariances can be used to calculate the associated correlations. The following shows these correlations, including the ones discussed in § 7...

```
R> str(ssBB.mc@corrs, 1)


List of 9
 $ tvbar.ba:'data.frame': 250 obs. of  4 variables:
 $ Tvbar.ba:'data.frame': 250 obs. of  4 variables:
 $ pvbar.ba:'data.frame': 250 obs. of  4 variables:
 $ mvbar.ba:'data.frame': 250 obs. of  4 variables:
```

[33] The order does not matter, this could just as easily be written as $\widehat{\text{cov}}\left(\hat{B}_{\text{c}}, \hat{B}_{\text{v}}\right)$ to match the name in the code.

```
 $ Pvbar.ba:'data.frame': 250 obs. of  4 variables:
 $ Mvbar.ba:'data.frame': 250 obs. of  4 variables:
 $ BcVv.cor:'data.frame': 250 obs. of  4 variables:
 $ BcBv.cor:'data.frame': 250 obs. of  4 variables:
 $ VvBv.cor:'data.frame': 250 obs. of  4 variables:
```

The last three data frames in the above list are associated with the desired correlations. In the following, those correlations for the first Monte Carlo sample with a sample size of $n = 25$ will be extracted...

```
R> BcBv.cor = ssBB.mc@corrs$BcBv.cor['mc.1', 'n.25']
R> BcVv.cor = ssBB.mc@corrs$BcVv.cor['mc.1', 'n.25']
R> VvBv.cor = ssBB.mc@corrs$VvBv.cor['mc.1', 'n.25']
R> c(BcBv.cor, BcVv.cor, VvBv.cor)


[1] 0.72235153 0.72777955 0.99854259
```

Finally, we calculate the first correlation above for $\widehat{\text{cov}}\left(\hat{B}_\text{c}, \hat{B}_\text{v}\right)$ to illustrate how these were determined in the R code...

```
R> BcBv.corr = BcBv.cov/(sqrt(Bc.varm)*sqrt(Bv.varm))
R> c(BcBv.corr, BcBv.cor)


[1] 0.72235153 0.72235153
```

The correlation between the two basal area estimates is 0.7224.

In summary Lynch's method provides a straightforward method for estimating the variance of a big BAF inventory. It also provides a clearly interpretable set of covariances and correlations built on a foundation of point-wise samples (with *no* aggregation approximations required) that are useful for the interpretation of cruise results beyond the estimation of the PBDM variance itself.

Appendix

A Bootstrap BC_a Comparison

This section was motivated based on the new R package `bcaboot` by Efron and Narasimhan (2018) that provides bootstrap BC_a interval estimates. The *sampSurf* package uses the `boot` package

(Canty and Ripley, 2020), and will continue to do so at this point, unless there is good reason to change it. However, since it was decided to use the former package here as it looked quite simple, and the authors certainly should be trusted, there is a question of how well the two might concur in the construction of BC_a intervals on a set of test data.

The following simple little function is included with the *ssExtra* package. It will draw n samples from a standard normal distribution, calculate the normal theory intervals, and calculate the bootstrap BC_a intervals from each of the two packages based on B bootstrap replications. Here is an example run...

```
R> args(testBoot)


function (n = 1000, B = 1000, alpha = 0.05, mbm = FALSE, parallel = "no",
    ncpus = 3L, times = 10L, unit = "s", runQuiet = FALSE, startSeed = 245,
    ...)
NULL


R> tb = testBoot()


Original sample normal theory intervals...
  Mean = -0.040537028 for n = 1000
  0.95 CIs = -0.10192412 0.02085006
bcajack...
  mean = -0.040537028
  0.95 bca cis =  -0.10200987 0.015298104
boot...
  mean = -0.040537028
  0.95 bca cis =  -0.098467178 0.027124759
```

Right away there is shown to be some discord between the two algorithms. The random number seed is reset to `startSeed` prior to invoking each of the two routines that calculate the BC_a intervals. Thus, the disagreement should not be due to any random number stream problems unless there is some randomization going on inside one or both of the package routines aside from drawing the bootstrap samples that I am not aware of. If that is the cause for the differences, however, then we should see them essentially converge (i.e., the randomness should be swamped by the sample size) as n and B get large. Note that `boot` requires a large bootstrap sample size, B, for BC_a intervals; too small a value will give an error[34], and it is quite replicable. For more information see this post at `stackoverflow`.

The results of a more extensive set of simulations are shown in Table 1, wich is created with the

[34] Error in bca.ci(boot.out, conf, index[1L], L = L, t = t.o, t0 = t0.o, : estimated adjustment 'a' is `NA`.

following[35]...

```
R> n = c(100, 200, 500, 1000)        #sample size
R> B = c(1000, 2000, 4000, 5000)     #number of bootstrap samples
R> tablePath = 'tables'
R> tableName = 'bootTable.tex'
R> cb = compareBoot(n = n, B = B, tablePath = tablePath, tableName = tableName,
+                   ssshhh = TRUE)
```

n	B	Norm.lo	Norm.hi	bcajack.lo	bcajack.hi	boot.lo	boot.hi
100	1000	-0.198	0.221	-0.201	0.205	-0.191	0.204
100	2000	-0.198	0.221	-0.200	0.207	-0.205	0.217
100	4000	-0.198	0.221	-0.191	0.212	-0.186	0.225
100	5000	-0.198	0.221	-0.192	0.208	-0.195	0.216
200	1000	-0.177	0.105	-0.172	0.108	-0.181	0.099
200	2000	-0.177	0.105	-0.173	0.110	-0.178	0.099
200	4000	-0.177	0.105	-0.172	0.107	-0.169	0.114
200	5000	-0.177	0.105	-0.171	0.106	-0.176	0.107
500	1000	-0.156	0.025	-0.165	0.016	-0.159	0.022
500	2000	-0.156	0.025	-0.159	0.019	-0.153	0.026
500	4000	-0.156	0.025	-0.158	0.022	-0.157	0.021
500	5000	-0.156	0.025	-0.158	0.025	-0.158	0.020
1000	1000	-0.102	0.021	-0.102	0.015	-0.098	0.027
1000	2000	-0.102	0.021	-0.103	0.018	-0.102	0.023
1000	4000	-0.102	0.021	-0.103	0.020	-0.101	0.022
1000	5000	-0.102	0.021	-0.103	0.020	-0.101	0.021

Table 1: Bootstrap Confidence Interval Comparison

For the smaller sample sizes the results do not look so good. And drawing large numbers of bootstrap samples from a small sample probably is not the best idea. As the sample size increases, the two are more concordant for the highest values of B. They also are tending to converge to the normal theory intervals at these values of B as well. Fortunately the differences are not terrible, and hopefully repeated sampling in a Monte Carlo setting will average out any discrepancies so that the percent catch statistics will not be too different. That will not be explored further here.

References

J. P. Barrett and L. Goldsmith. When is n sufficiently large? *The American Statistician*, 30:67–70, 1976. 29

[35] Please see the help `?compareBoot` for more information on this routine.

J. P. Barrett and M. E. Nutt. *Survey sampling in the environmental sciences: A computer approach*. COMPress, Inc., 1979. 29

J. F. Bell and L. B. Alexander. Application of the variable plot method of sampling forest stands. Research Note 30, Oregon State Board of Forestry, 1957. 16

J. F. Bell, K. Iles, and D. D. Marshall. Balancing the ratio of tree count-only sample points and VBAR measurements in variable plot sampling. In J. F. Bell and T. Atterbury, editors, *Renewable Resouce Inventories for Monitoring Changes and Trends*, pages 699–702, Corvallis, Oregon, December 1983. College of Forestry, OSU. 18

D. Bruce. Prism cruising in the western United States with volume tables for use therewith. Technical report, Mason, Bruce & Girard Consulting Foresters, Portland, Oregon, 1961. 16, 18

A. Canty and B. Ripley. boot: Bootstrap functions. R package, 2020. URL `http://lib.stat.cmu.edu/R/CRAN/web/packages/boot/index.html`. (Version 1.3-25). 67

J. M. Chambers. *Software for Data Analysis: Programing with R*. Springer-Verlag, 2008. 19

H. Cramér. *Mathematical methods of statistics*. Princeton University Press, Princeton, NJ, 1946. 15

K. M. Desmarais. Using BigBAF sampling in a New England mixedwood forest. Technical report, John Bell Associates, 2002. URL `http://www.john-bell-associates.com/guest/guest58b.htm`. 18

B. Efron and B. Narasimhan. The automatic construction of bootstrap confidence intervals. Technical Report 2018-07, Department of Statistics Stanford University, Stanford, California 94305-4065, 2018. 15, 66

A. J. Fast and M. J. Ducey. Height-diameter equations for select New Hampshire tree species. *Northern Journal of Applied Forestry*, 28(3):157–160, 2011. 7

R. Gentleman. *R Programming for Bioinformatics*. Chapman & Hall/CRC, Boca Raton, 2008. 19

L. A. Goodman. On the exact variance of products. *Journal of the American Statistical Association*, 55(292):708–713, 1960. 15

L. A. Goodman. The variance of the product of k random variables. *Journal of the American Statistical Association*, 57(297):54–60, 1962. 55

J. H. Gove. *The "Stem" Class*. `sampSurf` package vignette: `http://sampsurf.r-forge.r-project.org/index.html`, 2011. URL `https://CRAN.R-project.org/package=sampSurf`. 7, 8

J. H. Gove. *"monte": When is n sufficiently large?* `sampSurf` package vignette: `http://sampsurf.r-forge.r-project.org/index.html`, 2012a. URL `https://CRAN.R-project.org/package=sampSurf`. 29

J. H. Gove. sampSurf: Sampling surface simulation. *https://r-forge.r-project.org/projects/sampsurf/*, 2012b. 1, 25

J. H. Gove. *The "InclusionZone" Class.* `sampSurf` package vignette: `http://cran.r-project.org/web/packages/sampSurf/index.html`, 2013a. URL `https://CRAN.R-project.org/package=sampSurf`. `sampSurf` package vignette. 25

J. H. Gove. *The "Tract" Class.* `sampSurf` package vignette, `http://cran.r-project.org/web/packages/sampSurf/index.html`, 2013b. URL `https://CRAN.R-project.org/package=sampSurf`. `sampSurf` package vignette. 3

J. H. Gove. Some refinements on the comparison of areal sampling methods via simulation. *Forests*, 8(393):1–24, 2017a. 2

J. H. Gove. The ssWavelets package: Wavelet functionality for package sampSurf, 2017b. URL `http://sswavelets.r-forge.r-project.org/`. 2

J. H. Gove and G. P. Patil. Modeling the basal area-size distribution of forest stands: a compatible approach. *Forest Science*, 44(2):285–297, 1998. 5

J. H. Gove, T. G. Gregoire, M. J. Ducey, and T. B. Lynch. A note on the estimation of variance for big BAF sampling. *Forest Ecosystems*, 2020. (In review). 3, 7, 11, 15, 16, 24, 55, 58, 59

T. G. Gregoire. Notes on variance of VBAR. August 2009. 12, 18

T. G. Gregoire and H. T. Valentine. *Sampling strategies for natural resources and the environment.* Applied environmental statistics series. Chapman & Hall/CRC, N.Y., 2008. 12, 15

J. A. Kershaw, M. J. Ducey, T. Beers, and B. Husch. *Forest Mensuration.* Wiley-Blackwell, 5th edition, 2016. 11, 12, 18

T. B. Lynch, J. H. Gove, T. G. Gregore, and M. J. Ducey. An approximate point-based alternative for the estimation of variance under big BAF sampling. *In Preparation*, 2020. 2, 58, 59

D. D. Marshall, K. Iles, and J. F. Bell. Using a large-angle gauge to select trees for measurement in variable plot sampling. *Canadian Journal of Forest Research*, 34:840–845, 2004. 18

T. R. Yang, Y. H. Hsu, Kershaw, Jr., E. McGarrigle, and D. Kilham. Big BAF sampling in mixed species forest structures of northeastern north america: influence of count and measure BAF under cost constraints. *Forestry*, 90:649–660, 2017. 18

F. Yates. *Sampling Methods for Censuses and Surveys.* Charles Griffin and Co. Ltd., London, 2nd edition edition, 1953. 15