sampSurf: Sampling Surface Simulation for Areal Sampling Designs in R

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

Monday 8th January, 2018

1:42pm

Summary

Sampling methods in natural resources are largely based on probability proportional to size or frequency concepts and are often referred to as areal sampling methods, because they induce an enlarged area about the object within which a random point may fall and select the object. The ability to compare different sampling methods is key to designing inventories and assessing the efficiency of newly developed methods when compared against existing methods. Simulation methods are often used on synthetic populations of individuals with the desired characteristics of some target population in order to assess the adequacy of differing methods. The **R** (R Core Team, 2017) `sampSurf` package (Gove, 2017c) was developed to facilitate the comparison of sampling methods for forested or previously forested target populations of standing trees and downed coarse woody debris. Simulation is accomplished by constructing one or more "sampling surfaces" from which the efficiency (in terms of variance) of a method can be deduced relative to other methods. The theory behind sampling surface estimation and development is reviewed along with the package design and examples showing its use and extension.

Contents

1 Introduction **2**
1.1 Probability sampling 4
1.2 The sampling surface approach to continuous populations 6
1.2.1 Sampling surface estimation 8
2 Package design **10**
2.1 Object orientation and sampSurf 10
2.2 On container classes 13
2.3 The Tract class 13
2.4 The Stem class 14
2.4.1 Collections of "Stem" objects . . . 15
2.5 The ArealSampling class 16
2.6 The InclusionZone class 19
2.6.1 Collections of "InclusionZone" objects 21
2.7 The InclusionZoneGrid class 22
2.8 The sampSurf class 26
3 Survey design and comparison **28**
3.1 Equalizing overall sampling effort 29
3.2 The Smith plot: sampling effort made simple 34
3.3 When is n sufficiently large? 36
4 Monte Carlo Subsampling **40**
4.1 Subsampling "Stem" class objects 41
4.2 Subsampling areal sampling methods . . . 43
5 Extending sampSurf **46**
5.1 Extending the ArealSampling class 47
5.2 Extending the InclusionZone class 49
5.3 Extending the InclusionZoneGrid class . . 51
5.4 Helper function extensions 52
5.4.1 Methods for plotting 52
5.4.2 Other required methods 53
5.5 Remarks 54
5.6 An example 54
6 Conclusions **56**
7 Acknowledgements **57**
Bibliography **57**

1 Introduction

Areal sampling methods are the foundation for most inventories on terrestrial ecosystems in forestry, ecology and related fields. These surveys often employ time- and field-tested methods that are both simple to understand and implement in the field, such as circular or rectangular fixed-area plots[1]. In addition to being straightforward in application, these methods are also known to be unbiased when applied correctly.[2] The efficiencies of these simple methods are not as straightforward to ascertain, however, because they depend on the population being sampled and the attributes of the population to be estimated (e.g., volume, number of stems (density), surface area). Efficiency can also depend on plot shape and size, which are determined by design parameters (i.e., plot radius). Additionally, all of these factors affect the cost, which is often of major consideration in the design of inventories.

In the last half century, many new sampling methods have been proposed for standing trees and downed coarse woody debris (logs hereafter) in forested (or what were once forested) ecosystems. Key among these methods is the idea of using variable radius plots. The first such use was undoubtedly the application of concentric circular fixed-area plots for different sized individuals, but the idea really came to fruition when Walter Bitterlich (Bitterlich, 1948) introduced the so-called angle-count method, also known as "Bitterlich," variable plot, or horizontal point sampling. Under this method, standing trees are selected from a sample point with probability proportional to their

[1] Small square plots are sometimes referred to as quadrats in the ecological literature.

[2] For example, one very important—but often neglected—factor in the correct application of any such methods, is that due regard is given for boundary conditions where "slopover" corrections may be required (e.g., Ducey et al., 2004, and references therein).

basal areas.[3] This results in a conceptual fixed-area circular plot for each tree, whose radius is proportional to the tree's diameter. The probabilistic interpretation of horizontal point sampling was due to Grosenbaugh (1958), and cast probability proportional to size (PPS) sampling methods in stone, as it were, as the way foresters would henceforth think of sampling. In the ensuing years, countless new PPS methods have been put forth for sampling standing trees and down woody debris pools, too many in fact to list here (Gregoire and Valentine, 2008 or Kershaw et al., 2016 should be consulted for a recent exposition of many of these methods). These PPS methods are most often "optimized" for a certain design attribute, though they are flexible enough to allow the estimation of most other common attributes as well. For example, in horizontal point sampling, the design attribute is basal area, and it follows that any quantity that is related to basal area (e.g., volume) will be estimated more efficiently than those attributes such as stem density, that are not optimal in terms of the underlying probability design.

In general, as new sampling methods are developed, properties such as estimator unbiasedness can usually be shown analytically. However, one often must resort to simulation to determine the properties of estimator efficiency for various attributes, because, as mentioned earlier, these results are conditional on the populations of individuals in question. Comparisons of new to existing sampling methods, therefore, routinely involve simulation experiments to see how the proposed new methods 'stack up' against the known performance of existing methods. It is this simulation-based component of sampling method comparison and inventory design that the `sampSurf` package (Gove, 2017c) addresses. The estimator bias (if any) and variance properties can be readily assessed with the components of this package, which also provides a general framework for the addition of new methods under **R**'s `S4` object oriented programming paradigm. But what exactly is a "sampling surface?" This will be addressed in detail in the remainder of this paper, but quite simply, a sampling surface is a three-dimensional representation of the surface generated by the point estimates in sampling for a given attribute using the desired sampling method. To create a sampling surface, a population of synthetic stems is generated over a tract of land whereon a grid of sample points has been determined, such that the attribute estimate at each point yields the surface value for the stems that were selected into the sample at that point. Therefore, the enumeration of estimates at all grid points renders the attribute surface estimate for the population and sampling method for a given sampling resolution. The basic idea has been used in forestry to illustrate the concepts behind certain sampling methods such as critical height sampling (Kitamura, 1962; Iles, 1979) in the past, but it was Williams (2001a,b) who formalized the idea into a general approach useful for evaluating different areal sampling methods. The approach has subsequently been used in several studies (e.g., Williams and Gove, 2003; Gove et al., 2005; Ståhl et al., 2010; Gove and Van Deusen, 2011; Gove et al., 2012a,b); however, the original papers should be consulted for details beyond what is included here.

The concepts behind areal sampling designs and related PPS estimators are described in the following subsections. An overview of the design of the main `sampSurf` class components with examples illustrating their use is then described. Next, some extended examples of how the components might be used to compare different sampling methods is presented. Finally, an example showing how the package can be extended for inclusion of new methods is described. Throughout, design-

[3] The stem cross-sectional area at breast height (1.3 m), usually assuming circular cross-section for convenience.

based inference (Gregoire and Valentine, 2008, p. 9) is assumed, so that the population of trees or logs is regarded as fixed, and inferences are drawn from the theory of repeated sampling under the design. In `sampSurf` both metric (SI) and English (Imperial[4]) units are supported and all diameters, widths and heights and lengths are stored in m (ft), while areas are in m^2 (ft^2), and volumes in m^3 (ft^3). In what follows metric units are used exclusively and assumed consistent with the `sampSurf` storage model. Several vignettes are distributed with the package illustrating its components in more detail than can be shown here, these are cited below where appropriate and should be consulted for more information.

To use the code in this vignette, make sure that `sampSurf` has been installed from *CRAN*[5] using `install.packages`. See the `sampSurf` web page `http://sampsurf.r-forge.r-project.org/` for instructions on installing `sampSurf` and other packages on which it depends. Then make sure the package is loaded to the search list by, e.g.,...

```
R> require(sampSurf, quietly = TRUE)
```

1.1 Probability sampling

The foundations of the sampling surface approach derive from classical probability sampling theory. The target population of interest is composed of N discrete units, in our case standing trees or downed logs. The units are spread across a well defined tract of land, $\mathcal{A}$, with area A. The total amount (τ_z) of an attribute associated with all N individual units on the tract is

$$\tau_z = \sum_{k=1}^{N} z_k \tag{1}$$

where z_k is the amount of the attribute of interest associated with unit k in the population. For example, interest may lie in the amount of carbon or volume in the population of trees or logs on a given tract of land. The average amount of attribute per individual in the population is therefore simply $\mu_z = \tau_z/N$.

Without exception, the sampling methods available in the `sampSurf` package can be envisioned as areal sampling designs. Areal sampling methods rely on an inflation of the areal extent within which a population element can be sampled, known as the object's inclusion zone. The inclusion zone for an element is simply the area in which a randomly located sample point can fall and select the element into the sample. The example in Figure 1 presents two logs with their associated inclusion zones under a circular fixed-area plot sampling protocol known as 'sausage' sampling (Gove and Van Deusen, 2011). Under the sausage method, the inclusion zones have radii of the chosen fixed-area plot radius but are not the shape of a circular plot, rather they are elongated, resembling a sausage in shape, so that the area is proportional to the log's length.

[4]Actually, U.S. Customary Units—see `https://en.wikipedia.org/wiki/Imperial_units` & links therein for an interesting history on the subject.

[5]The Comprehensive R Archive Network

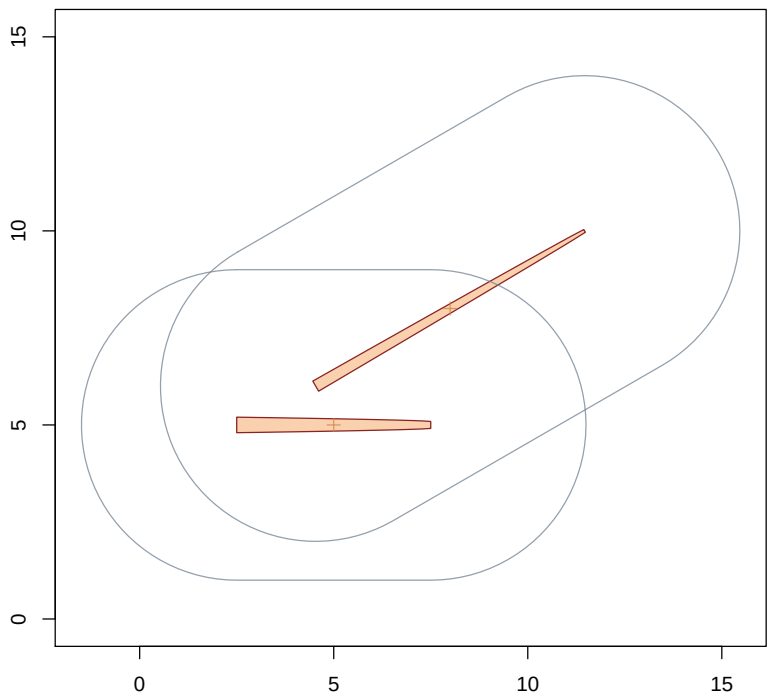


Figure 1: Inclusion zones for two down logs under the 'sausage' sampling protocol with a fixed-plot radius of 4 m. The joint inclusion zone is mapped as the intersection of the two individual zones.

Areal sampling designs are, by definition, probability sampling designs. Each sampling design or method assigns an inclusion probability, $0 < \pi_k \leq 1$, to the individual elements in the population. Under areal sampling designs, the inclusion probability can be determined for each individual based on the design parameters and, in unequal probability designs, some attribute of the individual. Unequal probability designs are also known as probability proportional to size sampling methods. Denote the inclusion area, a_k, as the area of the inclusion zone ($\mathcal{I}_k$) for the kth element in the population, then the element's associated inclusion probability is

$$\pi_k = \frac{a_k}{A} \tag{2}$$

Let z_k denote some quantity from which the target attribute can be deduced[6] on the kth individual. Then the Horvitz-Thompson (HT) estimator corresponding to the kth individual on a single sample

[6] Often this will entail direct measurements of the attribute in question, or measurements leading to a modeled estimate. However, under a number of designs supported in `sampSurf`, the attribute estimate is deduced indirectly using crude Monte Carlo or importance sampling methods.

point is given as

$$\hat{\tau}_{z_k} = \frac{z_k}{\pi_k} \tag{3}$$

while the estimate for all n_s individuals selected on the sth sample point is

$$\hat{\tau}_{z_s} = \sum_{k=1}^{n_s} \hat{\tau}_{z_k} \tag{4}$$

For example, in Figure 1, $n_s = 0$ for all points outside either of the individual zones, $n_s = 2$ within the joint inclusion zone for the two logs, and $n_s = 1$ elsewhere.

It follows that the HT estimator for the total from a sample of size m points is

$$\hat{\bar{\tau}}_z = \frac{1}{m} \sum_{s=1}^{m} \hat{\tau}_{z_s} \tag{5}$$

The design-based variance for the estimator (4) is complicated (Gregoire and Valentine, 2008, p. 216), involving the joint inclusion probabilities between pairs of individuals, $\pi_{k,k'}$, arising from the overlap of the inclusion zones shown in Figure 1. When designing inventories, the variance must be minimized, requiring knowledge of the $\pi_{k,k'}$, which are clearly unavailable for unknown populations. Likewise, in general field surveys, the pair-wise inclusion probabilities are impossible to calculate because doing so involves knowing the placement and often orientation, and attributes such as length, for the individual units in the population. While `sampSurf` does not explicitly determine the pair-wise inclusion zones, the estimation of the surface variance does implicitly include the zones of overlap and therefore reflects the defining factors such as design parameters, size and juxtaposition of individuals noted above.

1.2 The sampling surface approach to continuous populations

A continuous population is one that does not lend itself readily to discretization into individual units, and is defined over some designated physical area such as a water body or a tract of land. Interest in the context of the `sampSurf` package lies only in forest ecosystems supported on a defined tract of land, $\mathcal{A}$, with area A, as defined in the previous section. As was the case for discrete units, there are one or more attributes on the population for which estimates and inferences are desired. Gregoire and Valentine (2008, p. 93) define the attribute density, $\rho(x, y)$, over a land area such as a forested tract, as the amount of attribute per unit area. While the attribute density of interest is continuous because it is defined for all points $p(x, y)$ in the tract, the definition in general does allow for subareas with zero value. For example, if the population to be sampled is the forest floor and the attribute of interest is litter volume over $\mathcal{A}$, then the attribute density is litter volume per

unit land area (m^3 m^{-2}), or litter depth (m) at point p. It follows that the total volume of litter over the area is given as

$$\tau_\rho = \iint_A \rho(x, y)\, \mathrm{d}y\, \mathrm{d}x \tag{6}$$

Discontinuities or interruptions caused by features such as rock outcroppings or streams, for example, where litter depth, ρ, would be zero are perfectly reasonable. From this definition, the average attribute density over the tract is given as $\mu_\rho = \tau_\rho / A$.

Trees and logs are not continuous objects, so at first glance the continuous paradigm may seem inappropriate. However, under areal sampling designs, each unit (tree or log) in the population has an associated inclusion zone with a non-zero areal extent a_k, and an associated inclusion probability π_k, that facilitates the continuous approach. For illustration assume the unit is a down log and interest lies in estimating the aggregate log volume for a tract; the attribute of interest for each log, therefore, is its volume. By the definition of an inclusion zone, any sample point landing within the zone for this log will induce a measurement of the log's volume (often an estimate thereof, with associated measurement, model, etc. error). Referring to the example of the two inclusion zones in Figure 1 for fixed-area plot sampling, one can assign the HT estimate (3) of each log's volume to every point (infinitely many) within its respective inclusion zone and imagine this as the height of a flat surface internal to the zone. In other words, the estimate for each log is spread evenly across its entire inclusion zone's spatial extent. The surface height of the two zones will normally differ, because the volumes of the two logs will not be exactly the same. Any sample point falling within the joint inclusion area where the zones overlap would select both logs, and therefore the surface height within this area would be the cumulative surface height of the two zones. Now, if the tract is large, and the population of logs expands accordingly, then the inclusion zones for all of the logs meet the criterion for a continuous population in the sense that the total attribute density, $\rho(x, y)$, for the N logs in the population is apportioned over all the respective zones creating the *attribute surface*. Areas within the tract where there is no attribute density, because zones do not extend to these regions, are analogous to the rock outcroppings in the litter example above: The attribute density depth is zero in these areas.

In any simulation study of areal sampling methods, the population of individuals, whether trees or logs, is known exactly at fixed locations within $\mathcal{A}$. Under a given design, the inclusion zones can therefore be mapped exactly based on the design parameters and some attribute of the individual units, depending on the method under consideration. Simulation in general seeks to determine (or verify) the unbiasedness and efficiency in terms of variance of the estimator for the sampling methods. In a pure Monte Carlo approach, sample points would then be drawn at random from the population, and for each point the determination of the overall estimate for the point would be made using (5). This approach is somewhat "brute force" in application, because for each random sample point, the determination must be made as to which inclusion zones the point falls into. This necessitates visiting all trees in the population for each point, and applying a point-in-polygon routine to each inclusion zone, although some refinements could be envisioned to speed up the process. If a large enough Monte Carlo sample were taken, then a reliable estimate of the estimator bias and variance for the design can be produced. However, it falls to the chance juxtaposition of the points as to whether a map of the attribute estimates over $\mathcal{A}$ can be produced

at a reasonable resolution for visual assessment.

The sampling surface technique takes a more methodical approach to the simulation problem. The tract is tessellated into square grid cells of common size, with the center of each cell treated as a sample point. The number of sample points is controlled by the grid resolution—the higher the resolution (smaller the grid cell size) the more sample points are produced. In this sense, the analogy to the Riemann sum over an area can be invoked to approximate the integral (6). In the limit as $m \to \infty$ (or the grid cell size goes to zero), the Riemann sum approaches the integral (6). Thus, the sampling surface approach is a full raster coverage of the tract creating a map of the estimation surface whose detail is directly related to the raster grid resolution.

Once the population of inclusion zones is mapped on the tract, a relatively efficient algorithm can be implemented to assign the estimates to each sample point using map overlay routines (see Section 2.7). Furthermore, because of the full coverage (at given resolution) systematic layout of the sampling grid, a visual representation of the sampling surface for the attribute and sampling method under consideration can be easily generated. The roughness of the surface displayed is related to the estimator variance as described below (Williams, 2001a,b).

1.2.1 Sampling surface estimation

`sampSurf` uses two different attribute surfaces in the class structures. The first is closely related to the attribute density, which is defined for the kth individual as (Gregoire and Valentine, 2008, p. 328)

$$\rho_k(x,y) = \begin{cases} \frac{z_k}{a_k} & \forall (x,y) \in \mathcal{I}_k \\ 0 & \text{elsewhere} \end{cases} \tag{7}$$

This generalizes over all inclusion zones for any grid cell point (x, y) to

$$\rho(x,y) = \sum_{k=1}^{N} \rho_k(x,y) \qquad \forall (x,y) \in \mathcal{A} \tag{8}$$

reflecting areas where inclusion zones overlap, such as the joint inclusion zone area depicted in Figure 1. A scaled version of this attribute density surface representation, putting estimates on a per hectare (acre) basis is used in the intermediate "InclusionZoneGrid" class when building a sampling surface (Section 2.7).

A second representation that is closely related to the attribute density uses the HT estimate of the

total over $\mathcal{A}$ at each grid cell, rather than $\rho(x, y)$; *viz.*,

$$\begin{aligned}\varrho_k(x, y) &= A\rho_k(x, y) \\ &= \frac{Az_k}{a_k} \\ &= \frac{z_k}{\pi_k}\end{aligned} \tag{9}$$

Similarly, the surface total at any point is given by

$$\varrho(x, y) = \sum_{k=1}^{N} \varrho_k(x, y) \qquad \forall (x, y) \in \mathcal{A}$$

again, reflecting the joint inclusion zones. This latter representation is used in the actual sampling surface representation for the "sampSurf" class (Section 2.8). Because the attribute total is used in each grid cell, the appropriate estimator for the sampling surface is given as the surface mean

$$\begin{aligned}\hat{\tau}_\varrho &= \frac{1}{m} \sum_x \sum_y \varrho(x, y) \\ &= \frac{1}{m} \sum_{s=1}^{m} \hat{\tau}_{z_s} \\ &= \hat{\bar{\tau}}_z\end{aligned}$$

This surface estimator is the discrete analog of the continuous surface total, τ_ϱ. As m gets large the sum approaches the integral quantity using the same arguments as a Riemann sum from the calculus

$$\begin{aligned}\tau_\varrho &= \lim_{m\to\infty} \frac{1}{m} \sum_x \sum_y \varrho(x, y) \\ &= \lim_{\Delta_A\to 0} \frac{1}{A} \sum_x \sum_y \varrho(x, y)\Delta_x\Delta_y \\ &= \frac{1}{A} \iint_A \varrho(x, y)\,\mathrm{d}x\,\mathrm{d}y\end{aligned}$$

where we have used the fact that the area of each grid cell is given by $\Delta_A \triangleq \Delta_x\Delta_y = \frac{A}{m}$. It is important to note that the interpretation of this last result is in terms of the discrete sampling surface approximation converging to the continuous surface. The estimator $\hat{\bar{\tau}}_z$ (and thus $\hat{\tau}_\varrho$) is unbiased regardless of the number of grid cells $m > 0$: A sample of $m = 1$ provides an unbiased estimate based on sampling theory, provided the estimator is theoretically unbiased. However, when m is small, the number of grid cell centers falling within inclusion zones is also small, and the estimate will be poor, because many zones may be underrepresented in the sample. This is no

different than normal field sampling where a small sample size in a variable population will produce an unbiased, albeit poor, estimate. The small amount of "bias" that is incurred here for small m (and eventually reported by `sampSurf`) for any unbiased sampling method is an "apparent bias" due to an inadequate coverage of the tract area and should not be construed as an estimator bias inherent in any of the areal methods discussed below (see Section 2.8).

The sampling surface variance is related to the roughness of the surface and is given by

$$\mathrm{Var}\left(\hat{\tau}_{z_s}\right) = \frac{1}{(m-1)} \sum_{s=1}^{m} \left(\hat{\tau}_{z_s} - \hat{\tau}_{\varrho}\right)^2 \tag{10}$$

As noted by (Williams, 2001a,b), the sampling surface variance measures the individual grid cell variability in height about a flat plane of height $\hat{\tau}_{\varrho}$. The less variability around $\hat{\tau}_{\varrho}$, the smoother the surface. Therefore, when visually comparing sampling surfaces for different methods over the same population of objects, a smoother surface implies a lower variance. The variance of the estimator can also be calculated as $\mathrm{Var}\left(\hat{\tau}_{\varrho}\right) = \frac{\mathrm{Var}(\hat{\tau}_{z_s})}{m}$, but is of limited use since it scales inversely with the grid cell resolution, whereas $\mathrm{Var}\left(\hat{\tau}_{z_s}\right)$ should provide a relatively consistent variance estimate over different surface resolutions (as long as m is not trivially small, as discussed above). Either can be used for comparisons among different methods, but (10) is reported by `sampSurf` and will therefore be used in what follows.

2 Package design

2.1 Object orientation and `sampSurf`

The `sampSurf` package makes use of the `S4` object oriented programming paradigm within **R**. Only the briefest overview is presented here, more details can be found in Chambers (2008) and Gentleman (2008). The object oriented concepts central to `S4` are classes and methods. Classes define the structure of objects that will eventually be instantiated and used in programming and fully support class hierarchies through inheritance where the parent, or superclass, can have one or more child or subclasses defined from it. Classes encompass a strict definition of their contents, stored in slots, requiring an associated type for each slot. For example, a class definition may specify that a particular slot can contain only an object of class "vector," which would also include any subclass object of "vector," such as "matrix." Beyond this, the class mechanism allows for such options as multiple inheritance and validity checking; the latter is in addition to the automatic slot type checking and is under the control of the class designer. In general, objects are constructed from the class structure using `new`, which does slot type checking and calls the validity checking code if present. The classes in `sampSurf` are often fairly complex, requiring graphical representations for many object slots. The direct use of `new` in such cases can be error prone. `S4` also provides for customized object initialization in a more general way through `initialize` methods called from `new`. However, a slightly different approach was taken within `sampSurf` known more generally as 'constructor' methods. Object constructors and the `initialize` function conceptually perform

similar tasks, but constructor methods within `sampSurf` adopt the name of the class much like creating an object of class "matrix" by calling the `matrix` method. Simple arguments are passed to the constructor methods to first create the slot component objects, then the object itself is finally created (using `new`) and returned by the constructor. Because validity checking is performed at object creation, the user can be certain that a valid object has been returned from the constructor if no error has been encountered. This approach places the burden of creating the required slot objects on the constructor method, whereas the user must create them when using `new` and `initialize`.

The base S4 paradigm adheres to the original functional programming design of **R** (Chambers, 2008, p. 43). Functions are applied to objects of different classes through the definition of generic functions and associated methods. Generic functions in a sense define the overall use that a function is to be put to: The `summary` and `plot` functions are two familiar examples. But the main role of the generic itself is to set up the mechanism by which the associated methods are dispatched (Chambers, 2008, p. 397). Dispatching is done based on the method signature arguments, and because there can be more than one signature argument, S4 supports multiple dispatch (Gentleman, 2008, p. 69). Therefore, using `plot` as an example, the correct method is found and used based on `plot`'s two signature arguments `x` and `y`. The dichotomy between the generic and the methods is pedagogical, as the complete generic consists of the generic function itself, plus all its methods (Chambers, 2008, p. 396); e.g., the `plot` function. A main advantage to using generic functions is the link to object class structure in the sense that a subclass object method can access the method code from its superclass through `callNextMethod`, allowing for the building of functional complexity through the class hierarchy. In `sampSurf`, generics are used both to define new functionality, such as constructor functions for the package classes, and to add support for existing functionality (e.g., `plot`, `summary`, etc.) by adding new methods to existing generics.

The main class structure in `sampSurf` is shown in Table 1. Several classes are virtual, establishing the base structure common to all their subclasses. In a conceptual overview of Table 1, a collection of individuals comprising the population of interest is derived from the "Stem" class on a rasterized land area represented through the "Tract" class. An areal sampling method is chosen through the "ArealSampling" class and combined with the population objects so that inclusion zones can be determined for each via the "InclusionZone" class. This collection of individuals comprising the population with associated inclusion zones under a given areal sampling method on a particular tract of land area are then individually combined into a sampling surface through the "sampSurf" class. In all but one case in Table 1, the constructor functions for the individual classes go by the same name as the class itself. The class structure and associated constructor functions will be discussed in more detail in the following sections. In addition, each of the major classes is described in full detail in package vignettes and these should be consulted along with the extensive online help for the package. Methods added for existing generic functions will be mentioned, but the main source is **R**'s help facility for these functions.

In the development of a sampling surface simulation one can choose to either construct the components individually, or apply a simple "sampSurf" constructor that will automatically construct the intermediate objects. In the former approach, it is very important that the units of measure, which can be either "English" or "metric," be compatible in each of the component objects. These

Table 1: Major class structure in `sampSurf`.

Class	Subclasses	Container Classes	Remarks
"*Stem*"	"downLog" "standingTree"	"downLogs" "standingTrees"	Base container class: "*StemContainer*"
"Tract"	"bufferedTract" "mirageTract"		"RasterLayer" subclass "RasterLayer" subclass
"*ArealSampling*"	"circularPlot" "pointRelascope" "perpendicularDistance" "distanceLimited" "angleGauge" "lineSegment"		
"*InclusionZone*"	Figure 2	"downLogIZs" "standingTreeIZs"	Base container class: "*izContainer*"
"InclusionZoneGrid"	 "csFullInclusionZoneGrid" "mirageInclusionZoneGrid"		Constructor: `izGrid` Constructor: `izGrid` Constructor: `izGridMirage`
"sampSurf"			
"monte"			Inventory design: Section 3.3

Note: Classes in italic are virtual in this table.

are generally checked in the object validation routines to insure consistency. In addition, many classes of objects allow for the specification of coordinate reference systems (CRS) at the time of their creation (Bivand et al., 2008, p. 82). These must also be compatible between objects and be commensurate with the basic measurement units involved; while some validation checking is done in this respect, the user should be careful to choose commensurate systems. For example, if units specified for creating one of the "Stem" class objects is "English," then the "Tract" object on which they are situated must also be created in the same units. The default system of units in all classes within `sampSurf` is "metric," while the default CRS is `NA` for user defined; this latter information is reported under the heading of "`spatial units:`" in object summary output.

2.2 On container classes

sampSurf implements a very basic container class structure for storing collections of objects. Container classes are found in other object oriented languages such as C++ and Java and generally allow storage of many different object types. The container classes in sampSurf are very rudimentary and are customized to specific classes of objects. In general, the objects in the collection are stored in a slot of class list. Various constructors are provided to create the collections; however, there are currently no editing methods available for, e.g., deleting or adding objects to the collection. In all cases, the simplest thing to do is extract the list from the container, perform the desired editing on the list, and reconstruct the container object. This insures that the summary information about the collection is always correctly updated. For example, container objects have a bbox slot, which stores the overall spatial bounding box in matrix form for the collection. Deleting an element in the collection without updating this bbox summary slot could cause problems when subsequently using the collection. R lists themselves are inadequate for storing collections of objects needed within sampSurf for several reasons. Most importantly, raw list objects have no further class structure so creating methods for common generics such as summary, plot, etc., to act on disparate collections of objects is not feasible. Treating the collection as an S4 (container) class allows the adaptation of methods from common generics as well as the simple creation of class-specific collection statistics within the constructor. The individual container classes shown in Table 1 will be discussed more completely in the sections below related to the object types they store.

2.3 The "Tract" class

The "Tract" class (Gove, 2011d) provides the specification for the simulated land area extents within which the population to be sampled from will exist. The "Tract" class is a direct subclass of "RasterLayer" in the raster package (Hijmans, 2012). As such, it benefits from all of the methods pertaining to its superclass. The "Tract" class has several constructors, all of which assume that the units used are compatible to whatever map coordinate reference system is requested (Bivand et al., 2008, p. 82). One of the simplest constructors assumes the origin is at $(x, y) = (0, 0)$ and accepts the x and y extents as the signature object, along with the raster cell size; *viz.*,

```
R> tract = Tract(c(x = 71, y = 71), cellSize = 0.5)
R> buffTr = bufferedTract(bufferWidth = 10, tract)
```

In this example, a one-half hectare tract has been created with resolution of 0.5 m. The "bufferedTract" subclass assigns an internal buffer of desired width (in this case, 10 m) to an existing "Tract" object as in the example above. "bufferedTract" objects are very useful in the construction of other objects (such as "Stem" and "InclusionZone"), because an appropriately chosen buffer facilitates the construction of these objects such that they will lie wholly within the overall tract extents. This will be demonstrated in the various classes discussed below.

An alternative is to use the "mirageTract" class (Table 1) rather than a "bufferedTract" for the

correction of boundary overlap. The details are given in Gove (2013b) along with examples. Essentially, one would simply substitute the "mirageTract" object for the `buffTr` object above in the subsequent code in this vignette to enable the mirage method (Schmid-Haas, 1969) of boundary correction. The generalized mirage method as described by Lynch and Gove (2014) is implemented in the package.

2.4 The "Stem" class

The "Stem" class is used to define individual down logs or standing trees corresponding to subclasses "downLog" and "standingTree", respectively. Objects of the respective classes are instantiated using constructors of the same name as the class, and can be generated synthetically or from field measurements. For example, an individual "downLog" object with geometric center at $(x, y) = (10, 15.2)$ might be simulated with...

```
R> dlog = downLog(buttDiam = 34.2, topDiam = 9.4, logLen = 8, logAngle = pi/4,
+                 centerOffset = c(x = 10, y = 15.2), solidType = 4,
+                 species = 'eastern white pine', vol2wgt = 21.8,
+                 description = 'minimal decay, no bark', wgt2carbon = 0.5)
R> dlog


Object of class: downLog
------------------------------------------------------------
minimal decay, no bark
------------------------------------------------------------

Stem...
  Species:  eastern white pine
  units of measurement:  metric
  spatial units:  NA
  location...
    x coord:  10
    y coord:  15.2
    (Above coordinates are for log center)
  Spatial ID: log:6jy249ks

downLog...
  Butt diameter = 0.342 meters (34.2 cm)
  Top diameter = 0.094 meters (9.4 cm)
  Log length = 8 meters
  Log volume = 0.44403689 cubic meters
  Log surface area = 6.5189369 square meters
  Log coverage area = 2.0746667 square meters
```

```
  Log biomass = 9.6800043
  Log carbon = 4.8400021
  Volume to weight conversion = 21.8
  Weight to carbon conversion = 0.5
  Log angle of lie = 0.78539816 radians (45 degrees)
  Taper parameter =  4

Taper (in part)...
    diameter length
1 0.34200000    0.0
2 0.33572050    0.4
3 0.32927346    0.8
4 0.32264470    1.2
5 0.31581794    1.6
6 0.30877430    2.0
```

The arguments to the constructor and their relation to the class slots are described in detail in Gove (2011c). All arguments have default values except the bulk density (`wgt2vol`) and the biomass to carbon conversion (`wgt2carbon`), because the former especially, is species specific. Each log is represented internally in the class structure by taper data. `sampSurf` provides a default taper function with associated volume, surface and coverage area equations. The `solidType` argument controls the shape of the log in the default equations where a value of two produces a cone, a positive value less than two generates a neiloid, and a larger value yields a paraboloid, with a maximum value of ten. These arguments, along with the associated diameters (`buttDiam`, `topDiam`), log length (`logLen`) and angle of lie (`logAngle`) allow for a large mixture of possible synthetic logs within a population. If field measurements have been taken on a population of logs, then these would be used instead. If taper data were available either from direct measurements or an alternative taper equation, there is a version of the constructor that takes a data frame representing the taper as the first argument, and constructs the log from this. The taper data are important for calculation of stem attributes like volume, and for graphical representation of the object (e.g., Figure 1). Because taper data on real logs and trees is often irregular, it is left to the user to insure that the data entered are reasonable. Standing trees are defined similarly to downed logs, and reflect only the main stem of the tree: No branching is supported in either class at this point.

2.4.1 Collections of "Stem" objects

The constructors for individual "downLog" and "standingTree" objects create individual objects. Collections of objects forming a population to be used in the sampling surface simulations can be constructed using the associated container classes (Table 1). Various constructors are available with various signature arguments. For example, one can provide a `list` of respective "Stem" subclass objects, or simply specify the number of objects to be drawn from within a given "Tract" object as follows...

```
R> trees = standingTrees(5, buffTr, dbhs = c(20, 40), solidTypes = c(1, 3),
+                        description = 'a small tree population')
R> trees


Object of class: standingTrees
------------------------------------------------------------
a small tree population
------------------------------------------------------------
Container class object...
  Units of measurement:  metric

  Encapulating bounding box...
        min       max
x 26.551974 58.204533
y 15.784988 56.566191

  There are 5 trees in the population
  Population tree volume =  3.0259228 cubic meters
  Population tree surface area =  40.753596 square meters
  Average volume/tree =  0.60518457 cubic meters
  Average surface area/tree =  8.1507192 square meters
  Average height/tree =  9.63 meters
(**All statistics exclude NAs)
```

The diameters at breast height (DBH) from which the trees will be drawn are passed as a range in the `dbhs` argument, as is the geometric taper (shape parameter) for the trees (`solidTypes`). Other arguments are also available. One drawback to this approach is that the diameters, taper, heights, etc., of the trees are associated randomly from the ranges specified, so this has the possibility of generating trees that can be unrealistic if the ranges are wide (i.e., very short thick or tall thin trees). For more realistic simulations, it is more appropriate to construct the individual trees or logs using the respective constructors, collect the objects into a list, and then construct the container class from the list of objects. Object locations are drawn at random in the default "StemContainer" constructors. This is another reason to perhaps prefer individual generation of object locations by a more realistic algorithm (e.g., Kershaw Jr. et al., 2010; Valentine et al., 2000). Alternatively, the spatial point pattern simulation methods in `spatstat` (Baddeley and Turner, 2005) could be used for generating other spatial distributions with known statistical properties.

2.5 The "ArealSampling" class

The choice of areal sampling method for use in a set of simulations is made through the "ArealSampling" class (Gove, 2011a). The sampling methods supported are shown in Table 1, but many

of these have different protocols that enlarge the actual pool of supported methods through the "InclusionZone" class. Methods for sampling down logs include fixed-area circular plot sampling (Gove and Van Deusen, 2011; Van Deusen and Gove, 2011), point relascope sampling (Gove et al., 1999), perpendicular distance sampling (Williams and Gove, 2003) and distance limited sampling (Gove et al., 2012b). Fixed-area circular plots can also be applied to sampling standing trees; additionally, horizontal point sampling (Bitterlich, 1948; Grosenbaugh, 1958) is also supported along with critical height sampling and several newer variants (Lynch and Gove, 2013). All these methods are PPS methods as discussed earlier and all but fixed-area plots (depending upon the protocol) require some stem dimension to develop the inclusion zone. Therefore, none of the "ArealSampling" classes in Table 1 other than "circularPlot" and "lineSegment" has any inherent graphical information associated with the objects from the respective classes. Because the information content required for "ArealSampling" objects is often simple, two different examples are presented to illustrate the idea.

In the first example, a fixed-area plot of one-quarter hectare in area is created in the associated "circularPlot" object. The object has a spatial representation as a "SpatialPolygons" object (package `sp`), and the number of points comprising the perimeter is under user control with a default as shown below...

```
R> plotRad = sqrt(10000 / (pi * 4))
R> (cp.as = circularPlot(radius = plotRad, centerPoint = c(x = 20, y = 30)))


Object of class: circularPlot
------------------------------------------------------------
fixed area circular plot
------------------------------------------------------------

ArealSampling...
  units of measurement:  metric

circularPlot...
  radius = 28.209479 meters
  area = 2500 square meters (0.25 hectares)
  spatial units:  NA
  spatial ID: cp:05mhs1y2
  location (plot center)...
    x coord:  20
    y coord:  30
  Number of perimeter points: 101 (closed polygon)
```

As a second example, consider horizontal point sampling, where a variable radius circular plot is associated with each tree, the radius of which, depends on the size of the tree diameter. Trees are

selected with probability proportional to their basal areas using an angle gauge, often in practice in the form of a wedge prism...

```
R> (ag.as = angleGauge(baf = 5))


Object of class: angleGauge
------------------------------------------------------------
angle gauge method
------------------------------------------------------------

ArealSampling...
  units of measurement:  metric

angleGauge...
  Angle (ν) in degrees = 2.5625587 (153.75352 minutes)
  Angle (ν) in radians = 0.044725087
  Angle diopters (Δ) = 4.4754933
  Gauge constant (k) = 0.04472136
  Plot radius factor (prf) = 0.2236068 meters per cm (22.36068 meters per meter)
  Plot proportionality factor (α) = 44.7 meters per meter
  --Points...
    Basal area factor (baf) = 5 square meters per hectare
  --Lines...
    Diameter factor (df) = 134.16408 cm per hectare for a line segement of 20 meters
    Diameter factor (DF) = 11.18034 m per hectare for a line segement of 20 meters
```

An "angleGauge" object was constructed for a basal area factor 5 metric prism, where each tree selected on a sample point by the projected angle counts equally for 5 m^2 ha^{-1} of basal area. The plot radius factor for this angle denotes the distance in meters to the plot perimeter from the tree center for each cm or m of tree diameter at breast height (1.3 m). The other information reported is described in detail in standard texts on forest sampling such as Gregoire and Valentine (2008, Chapter 8) and Kershaw et al. (2016, Chapter 11). Because the radius of the plot is determined by the diameter of the tree, the determination of horizontal point sampling inclusion zones requires additional information about each tree in the population; hence, no spatial information is present in this object. This is more the norm with "ArealSampling" subclasses. Notice that "angleGauge" objects can be used when sampling from both points or along lines, and the appropriate information for both is shown.

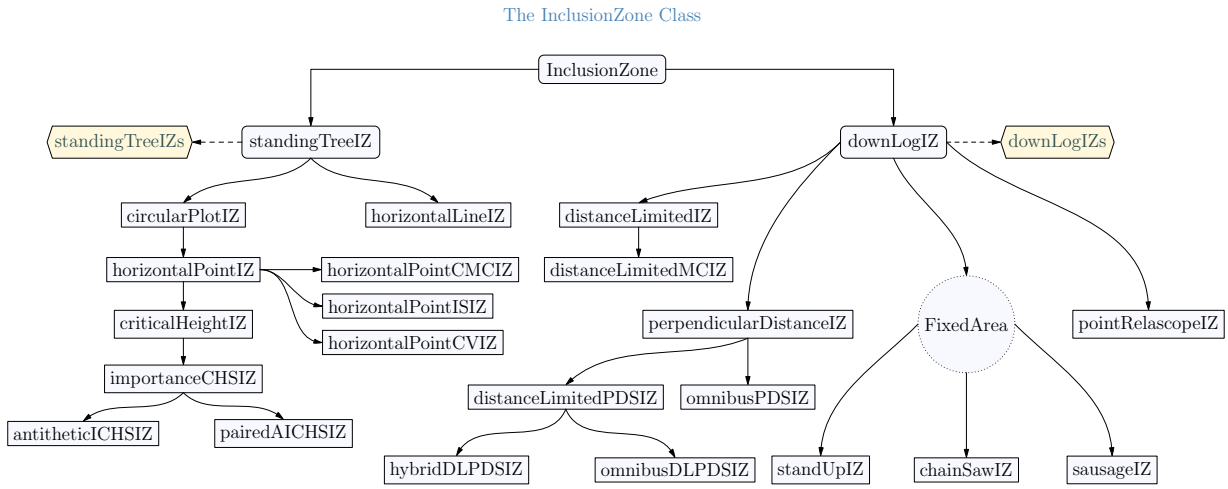


Figure 2: The "InclusionZone" class hierarchy illustrating the different inclusion zone possibilities in `sampSurf`. The "InclusionZone," "standingTreeIZ" and "downLogIZ" classes are virtual classes, while "standingTreeIZs" and "downLogIZs" are container classes.

2.6 The "InclusionZone" class

Figure 2 presents a class hierarchy for the "InclusionZone" class. The base class and the two main branches ("standingTreeIZ" and "downLogIZ") are virtual, with the two branches being associated with the corresponding "Stem" subclasses. There are a total of 34 different sampling protocols represented in the diagram corresponding to the methods shown. All of these subclasses are discussed in detail in the vignette for this class (Gove, 2012b).

This count comes from two sets of extra protocols that are not shown in the diagram. First, for standing trees, horizontal point sampling has three Monte Carlo (§ 4) variants that are shown to the right, and each of these has the option of applying the corresponding form of Monte Carlo subsampling (crude Monte Carlo, importance sampling, or control variate sampling) to the stem either directly or via antithetic sampling (the latter option is not shown)[7]. Note also that each stem may be subsampled multiple times (§ 4) if desired at each given sampling point within its inclusion zone under the protocol chosen. Of course, critical height sampling (CHS) (Kitamura, 1962) is also a Monte Carlo subsampling sampling method, but the variants for CHS are listed directly in Figure 2.

Second, for downed logs, all of the subclasses that derive from perpendicular distance sampling ("perpendicularDistanceIZ") actually have three associated sample selection protocols where logs can be selected with probability proportional to either volume, surface or coverage area. This is similar to the three fixed-area plot protocols (Gove and Van Deusen, 2011), which are explicitly shown.

[7]See the help for these methods; e.g., `methods?horizontalPointCMCIZ` for the antithetic argument option.

As previously mentioned, the "InclusionZone" class is where "Stem" and "ArealSampling" subclass objects are merged to form spatial representations for subsequent sampling surface determination. A simple example involves creating an "InclusionZone" subclass object for so-called 'sausage' sampling using the "sausageIZ" constructor (Figure 2)...

```
R> (sa.iz = sausageIZ(dlog, plotRadius = 5))


Object of class: sausageIZ
------------------------------------------------------------
inclusion zone for downed log "sausage" sampling method
------------------------------------------------------------

InclusionZone...
  Units of measurement:  metric
  Per unit area blowup factor: 63.075638 per hectare

  Object bounding box...
        min       max
x 2.1722151 17.828427
y 7.3722151 23.028427

downLog component estimates...
  Spatial ID: log:6jy249ks
  Number of logs: 63.075638 per hectare
  Volume: 28.00791 cubic meters per hectare
  Surface area: 411.1861 square meters per hectare
  Coverage area: 130.86092 square meters per hectare
  Length: 504.6051 meters per hectare
  Biomass (woody): 610.57244 per hectare
  Carbon content: 305.28622 per hectare

sausageIZ...
  Spatial ID: saus:f0zt98m2
  radius = 5 meters
  area = 158.53982 square meters ( 0.01585  hectares)
  Number of perimeter points: 101 (closed polygon)
```

Plotting the object would result in a display similar to Figure 1, where two such objects have been created to show the common joint inclusion area. Internally, this constructor creates a "circularPlot" object from the `plotRadius` argument, which is then used to construct the representation of the zone; this can be seen in the last section of the object summary above.

2.6.1 Collections of "InclusionZone" objects

The virtual "izContainer" class has two subclasses, "standingTreeIZs" and "downLogIZs" that can be used to hold collections of "InclusionZone" objects (Table 1, Figure 2). As an example, the following code snippet creates a collection of "horizontalPointIZ" objects to represent the inclusion zones under horizontal point sampling for standing trees...

```
R> rPlot.tr = Tract(c(x = 50, y = 50), cellSize = 0.5)
R> rPlot.btr = bufferedTract(bufferWidth = 10, rPlot.tr)
R> trees = standingTrees(10, rPlot.btr, dbhs = c(20, 40))
R> hps.izs = standingTreeIZs(trees, iZone = 'horizontalPointIZ',
+                            angleGauge = ag.as,
+                            description = 'horizontal point sampling IZs')
R> hps.izs


Container object of class: standingTreeIZs
------------------------------------------------------------
horizontal point sampling IZs
------------------------------------------------------------
There are 10 inclusion zones in the population
Inclusion zones are of class: horizontalPointIZ
Units of measurement:  metric
Summary of inclusion zone areas in square meters...
       Min.     1st Qu.      Median        Mean     3rd Qu.        Max.
  68.679629   99.720194  115.475050  130.859839  164.615021  218.074848
        var        SDev
2386.543086   48.852258

Encapulating bounding box...
        min       max
x 8.6601781 44.001343
y 4.6212815 45.788463
```

The penultimate line applies the inclusion zone constructor to each tree in the list slot of the container for standing tree objects using the angle gauge specifications created in Section 2.5. Both the "standingTree" objects and their associated "horizontalPointIZ" objects are created within a small buffered one quarter-hectare "bufferedTract" plot object. The summary of the object shows statistics on the inclusion zone areas (a_k) in the collection.[8] Figure 3 illustrates the spatial representation of the collection, and was created using the `plot` method for the "izContainer" class;[9] *viz.*,

[8] This can also be represented graphically using the `hist` method on the collection object, i.e., `hist(hps.izs)`.

[9] Note that the slight transparent shading internal to the zones in Figure 3 can be turned off using `izColor = NA`

```
R> plot(hps.izs, axes = TRUE)
```

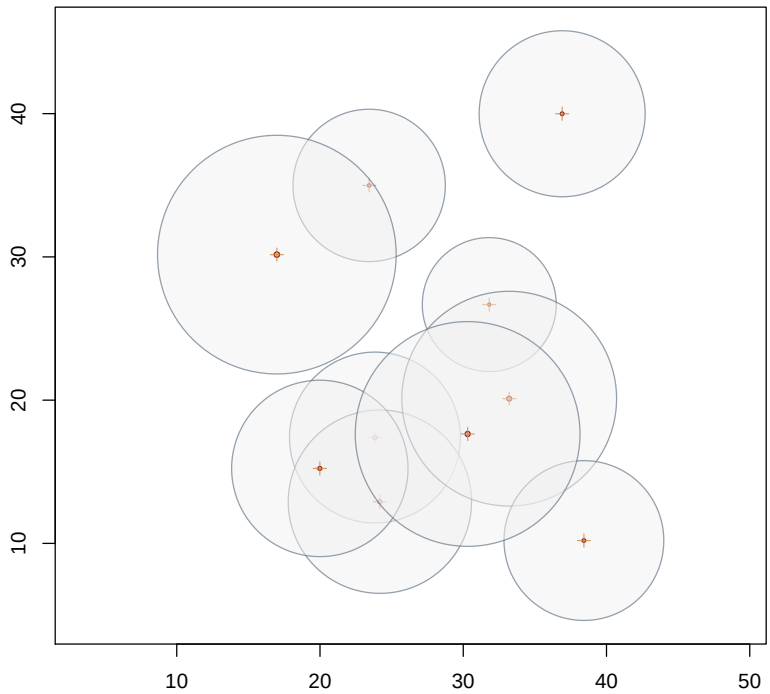


Figure 3: A plot of a "standingTreeIZs" container object under horizontal point sampling showing the collection of inclusion zones and associated trees.

2.7 The "InclusionZoneGrid" class

There must necessarily be some mechanism to associate the vector representation of the different "InclusionZone" object perimeters with the underlying "Tract" object, and thereby determine which grid cells fall within the different object inclusion zones. The "InclusionZoneGrid" class performs this function, and is largely transparent to the user. Objects are constructed automatically when building the actual "sampSurf" objects (Section 2.8) and in general, may never be accessed by the casual user of the system. However, in the sense of the task performed, they are the underlying workhorse to the construction of the final surface, and can be easily constructed, displayed, and manipulated, independently of creating a larger "sampSurf" object.

as an argument in the call to `plot`.

Earlier it was mentioned that one could pursue a "brute force" method of Monte Carlo sampling of the tract, where for each randomly generated point, one would need to identify the inclusion zones for the individuals that would be sampled. To accomplish this, one might apply a point-in-polygon search on each inclusion zone in the population, for example. The approach used in `sampSurf` can be refined somewhat because of the association between sample points and grid cell centers. The two general steps taken in the definition of an "InclusionZoneGrid" object are the following, which are applied to each "InclusionZone" object individually: *(i)* build an underlying temporary grid that minimally, but fully covers the inclusion zone and is rectified to the "Tract" object, and *(ii)* determine those grid center points that lie within the inclusion zone. These steps can be applied efficiently using methods found in the `raster` and `sp` packages.

In general, the areal sampling methods and protocols in `sampSurf` will fall into one of two categories within the "InclusionZoneGrid" class. Many methods assign values to the internal grid cells within an inclusion zone in a spatially independent manner, producing a surface of constant height within the zone for all estimate attributes. There are exceptions to this rule, however, where the estimate for a given attribute depends on the juxtaposition of the log and sample point. This second category of spatially dependent sampling methods produce variable height surfaces within the inclusion zone. As will be illustrated in the example of Section 5.3, defining a constructor function for methods producing flat surfaces is trivial—essentially the estimate can be assigned *en masse* to all cells within the inclusion zone using vectorization. On the other hand, for sampling methods that produce variable height surfaces within an inclusion zone, the constructor algorithm must visit each cell within the zone, and determine the estimate. In the case of several protocols for sampling down woody debris, this entails determining measurements on the log conditional on the individual sampling point locations. Because these measurements vary from point to point, they produce different estimates at each grid center location. Any of the Monte Carlo based sampling protocols will induce variable height surfaces and require more complex constructor functions for some attributes of interest. Note that even in the case where sampling protocols can produce variable estimates, often there are certain attributes under the protocol for which constant estimates are also produced. For example, when using the crude Monte Carlo protocol for distance limited sampling (Gove et al., 2012b) ("distanceLimitedMCIZ" class, Figure 2), estimating aggregate log length or density (number of logs) will produce flat surfaces because the only log dimension that enters into the estimator for these attributes is the individual log length, which is not spatially dependent on the sample point location; estimates of all other attributes require a measurement of log diameter (or a function thereof) that will depend on the location of the sample point, generating variable height surfaces. Other examples of protocols generating variable height surfaces are discussed in Ducey et al. (2008), Gove et al. (2005), Gove and Van Deusen (2011), and Gove et al. (2012b) for down logs, while Lynch and Gove (2013) provide a discussion for standing trees under various CHS protocols.

The concepts described above for this class can be aptly illustrated using the crude Monte Carlo protocol for the distance limited sampling method just discussed. In the following an "InclusionZoneGrid" class object is created for this protocol...

```
R> dl = distanceLimited(3)
R> dlmc.iz = distanceLimitedMCIZ(dlog, dls = dl)
R> (dlmc.izg = izGrid(dlmc.iz, buffTr))


Object of class: InclusionZoneGrid
------------------------------------------------------------
distanceLimitedMCIZ inclusion zone grid object
------------------------------------------------------------

InclusionZone class: distanceLimitedMCIZ
  units of measurement:  metric

Grid class: RasterLayer
Number of grid cells = 462
Cell dimensions: (nrows=21, ncol=22)
Grid cell values**...
  gridValues Freq
1          0  187
2       <NA>  275
**Note: data slot values get swapped with zero-valued grid cells as necessary.

Per unit area estimates in the data slot (for cells inside IZ only)...
        volume Density Length surfaceArea coverageArea biomass carbon depth
Min.     24.94   208.3   1667       722.8        230.1   543.8  271.9     1
1st Qu.  62.56   208.3   1667      1144.7        364.4  1363.9  681.9     1
Median   91.68   208.3   1667      1385.7        441.1  1998.6  999.3     1
Mean     91.98   208.3   1667      1356.3        431.7  2005.1 1002.5     1
3rd Qu. 123.49   208.3   1667      1608.2        511.9  2692.0 1346.0     1
Max.    148.61   208.3   1667      1764.2        561.6  3239.7 1619.8     1


  Encapulating bounding box...
   min  max
x  4.5 15.5
y 10.0 20.5
```

Note in the summary for the object that the full surface of values is stored for each attribute within a "dataframe" object; and it is easily verified from this information that the surface is constant for length and density estimation, but variable for all other attributes as noted above. The actual grid values within the "RasterLayer" slot of the object are either zero for cells within the inclusion zone, or `NA` for those outside the zone. When the object is used, say for plotting as below, the desired attribute values get swapped into the "RasterLayer" object. Thus, in the following plot,

we must specify the attribute for the surface to be rendered using the `estimate` argument to the `plot` method for this class. The representation in Figure 4 clearly shows the variable height surface within the log's inclusion zone under this sampling protocol.

```
R> plot(dlmc.izg, estimate = 'carbon', gridCenters = TRUE)
```

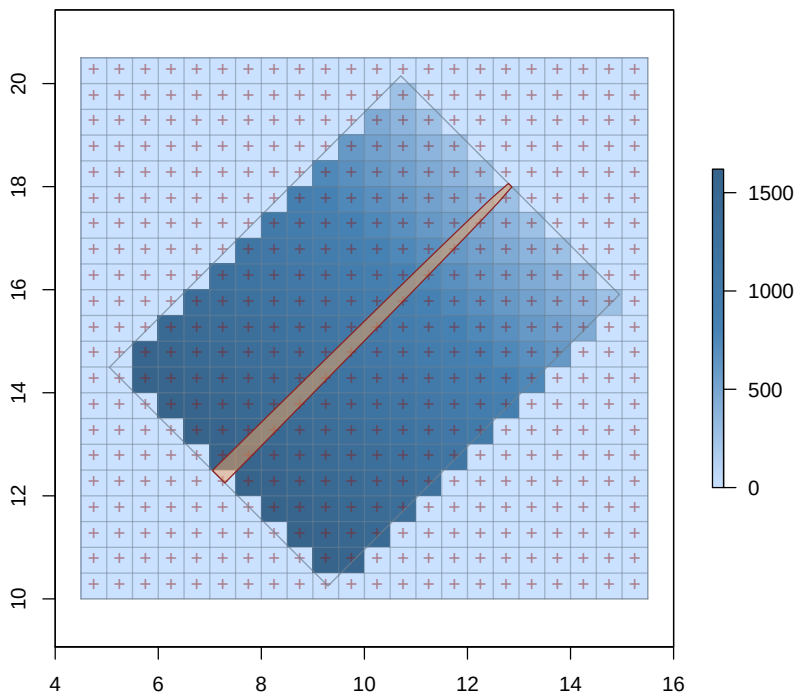


Figure 4: Graphical representation of a "InclusionZoneGrid" subclass object under the crude Monte Carlo protocol for distance limited sampling "distanceLimitedMCIZ"; the attribute surface is for carbon content.

The grid cell centers have been displayed in Figure 4 to illustrate that only those grid cells whose centers fall within the inclusion zone receive values for the particular attribute estimate. The offset to the polygon from the underlying grid might at first appear as a 'misalignment' problem, but it is not. The extent to which this visual effect appears depends on the underlying grid resolution, the shape of the inclusion zone for a given sampling method, and the orientation of the zone with respect to the grid; it can in some cases also depend on the location of the object (log or tree) center, which can be anywhere within a cell, and is never 'snapped' to the cell center. It is clear in Figure 4 that the effect is due to these factors, and that, e.g., increasing the grid resolution would lessen the effect, because more grid cell centers would fall within the zone at finer resolutions. This subject will be revisited again in Section 5.

Finally, it is very important to note that the estimate values stored in objects of class "InclusionZoneGrid" are per unit area (per hectare or acre) estimates, $\rho_k(x, y)$, not totals. The reason for this is that even though the objects are rectified to a "Tract" object in terms of alignment and resolution, the full "Tract" object is still disassociated from the "InclusionZoneGrid" object at this point, that association comes in the next section when building the sampling surface for the tract. Further details on the "InclusionZoneGrid" class are provided in the class vignette (Gove, 2011b).

2.8 The "sampSurf" class

The "sampSurf" class (Gove, 2012c) brings together all of the component classes discussed thus far into the formation of the sampling surface. Briefly the steps to building the surface are (note that appropriate subclasses are substituted as necessary) *(i)* construct a tract on which everything will reside using the "Tract" class, *(ii)* populate the tract with a collection of "Stem" objects, *(iii)* create the appropriate "ArealSampling" class object and *(iv)* combine the "Stem" objects with the desired sampling method through the appropriate "InclusionZone" class, *(v)* and finally construct the "sampSurf" class object. A careful reading of these steps will reveal that the generation of "InclusionZoneGrid" objects is missing. As noted in Section 2.7, it is unnecessary in general to construct these objects, the task is left to the actual "sampSurf" constructors. For each inclusion zone in the population, the appropriate "InclusionZoneGrid" constructor method for the `izGrid` generic is applied by the `sampSurf` constructor. In this case, the underlying "Tract" object is available, and the surface is built by 'heaping' the individual "InclusionZoneGrid" objects using simple raster addition. Note that in applying these steps for building the sampling surface the background grid cells (sample points) that are not covered by inclusion zones need not be visited in the simulation, producing a relatively efficient graphical method for surface creation.

Two constructor methods are available for generating sampling surfaces. The first relies on the user to apply the individual steps outlined in the previous paragraph, and is detailed in the following sections. The second, allows the generation of sampling surfaces 'on the fly' by specifying some simple information on the type of sampling method desired. An example of the latter follows using the same crude Monte Carlo protocol as in the last section, but for aggregate log length, which will produce constant surface height within individual inclusion zones as noted above[10]...

```
R> nlogs = 40
R> dlmc.ss = sampSurf(nlogs, buffTr, iZone = 'distanceLimitedMCIZ',
+                     dls = dl, estimate = 'Length', runQuiet = TRUE,
+                     description = 'constant internal surface heights',
+                     startSeed = 9872)
R> sampSurf::summary(dlmc.ss)
```

[10]Normally, one can simply type `summary(dlmc.ss)` to get the summary shown—and this will work for you. I believe that the KNITR environment in which the commands are processed has not registered the `summary` method correctly for "sampSurf" objects (the reason escapes me), hence the `sampSurf::summary` qualification used here.

```
Object of class: sampSurf
------------------------------------------------------------
constant internal surface heights
------------------------------------------------------------

Inclusion zone objects: distanceLimitedMCIZ
Measurement units = metric
Number of logs = 40
True log volume = 7.0982121 cubic meters
True log length = 208.31 meters
True log surface area = 123.97157 square meters
True log coverage area = 39.439462 square meters
True log biomass = NA
True log carbon = NA

Estimate attribute: Length
Surface statistics...
  mean = 208.33333
  bias = 0.023333333
  bias percent = 0.011201255
  sum = 4200833.3
  var = 205506.79
  st. dev. = 453.32857
  cv % = 217.59771
  surface max = 3360.6667
  total # grid cells = 20164
  grid cell resolution (x & y) = 0.5 meters
  # of background cells (zero) = 16114
  # of inclusion zone cells = 4050
```

```
R> plot(dlmc.ss)
```

The summary result records the slight 'apparent bias' that was alluded to previously in Section 1.2.1. There it was noted that this bias arises simply from the fact that it is not possible to sample infinitely many points on the tract. To put it in context, the 0.023333 m of length is just 2.3333 cm distributed over a population of logs totaling 208.31 m, or 0.0112 percent, and is on the order to be expected by any numerical approximation of this resolution (0.5 m).[11] Note that because this is a probability proportional to length method, all inclusion zone estimates are the same so a count of the "in" logs (logs whose inclusion zones overlap the grid point) on any grid point yields an estimate of aggregate length (Figure 5).

[11] The result here is actually quite low, a change in the population will change this result; try running the above with a different random seed, `startSeed`, to verify this. You may get an order of magnitude higher in 'bias percent', which is still quite acceptable.

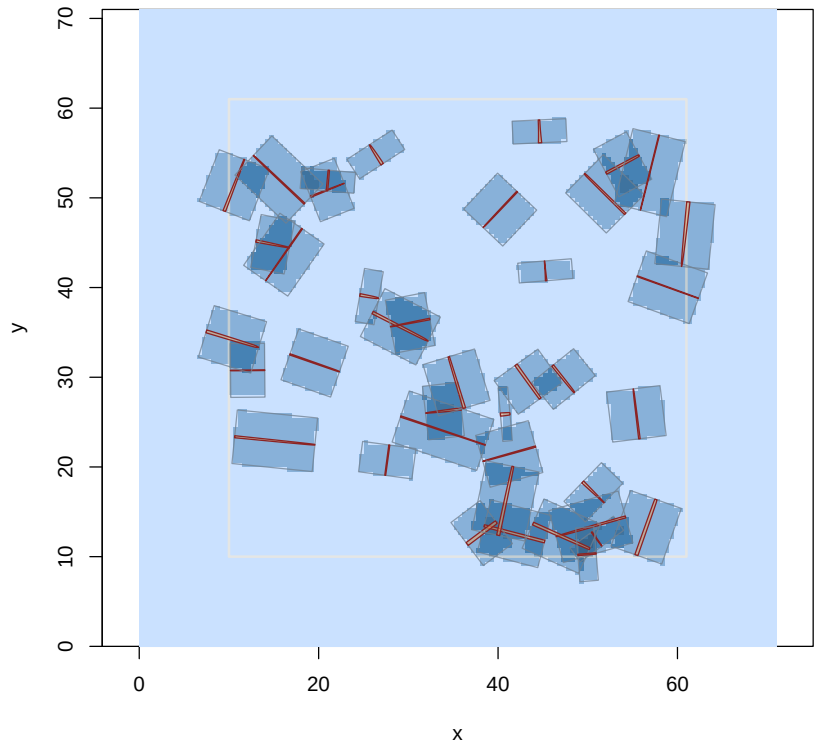


Figure 5: A sampling surface simulation using the crude Monte Carlo protocol of distance limited sampling for the estimation of aggregate log length in the population.

Lastly, "sampSurf" objects store the attribute estimate values for the surface within the object as the attribute total, $\varrho(x, y)$, for each grid cell. While working with the individual "InclusionZoneGrid" objects, the conversion is made from per hectare (or acre) to total within the constructor methods.

3 Survey design and comparison

Now that the basic structure and some examples of the components of the `sampSurf` package have been demonstrated, it is time to turn attention to some of the more useful aspects of the package. In general, as mentioned earlier, when either designing a new sampling method or when comparing extant methods for the design of a field inventory, one is presented with similar questions to be addressed. For new methods, this includes determining whether the method is unbiased. This can often be shown analytically and, therefore, may not require simulation. However, a second and no less important question concerns the efficiency of the method in terms of variance. This is often a

more difficult question to answer without resorting to some form of simulation experiment, and it is where a tool like `sampSurf` can be of some benefit.

The previous sections presented examples of the components of `sampSurf` that were often unrelated through the progression of objects in order to demonstrate a small number of the options and capabilities in the overall package. In this section, we turn our attention to a comparison of just two sampling methods from the standpoint of survey design. The same population of logs is used on the same tract to facilitate comparison of the two sampling methods. Inclusion zones are then determined for the population of logs and the corresponding sampling methods, and sampling surfaces developed. Two related methods are presented; the first seeks to equilibrate the sampling effort between methods being compared, while the second uses graphical comparisons. Both are useful in their own right, though the second is arguably simpler to implement in practice.

3.1 Equalizing overall sampling effort

There is a subtle but import point in the execution of a simulation experiment such as this that needs to be considered. It is what shall be termed the 'effective sample size' and will be defined as the number of grid cells covered by inclusion zones under a given sampling method—the overall sampling effort underlying the surface. The effective sample size is different for each population of "Stem" objects (including their sizes and possibly orientations), grid cell resolution, and sampling method. For example, note that in scrutinizing Figure 5 and the summary for the object, that only a small percentage (4050 out of 20164) of grid cells is sampled, having been covered by the inclusion zones for the objects. Adjusting something as simple as the limiting search distance design parameter under this sampling method would change this figure for the same population of logs and tract. Indeed, the point we are after is to try to determine fairly commensurate effective sample sizes between the methods that are being compared, in order to have approximately the same sampling effort over the tract for each method. The reason for this is the fact that the variance of the surface will decrease as the inclusion zones for individual objects increase (Williams, 2001a,b). This follows because the surface will tend to become smoother as the attribute density is spread over a larger area. As mentioned earlier, the variance of the surface is simply the grid cell variability around the mean surface plane and will decrease as the effective sample size (area covered by inclusion zones) increases. This has been demonstrated for fixed-area circular plots (Williams, 2001a) and distance limited sampling (DLS) (Gove et al., 2012b). The concept is analogous to the relationship of variance to plot size as first pointed out by Smith (1938), and corroborated by others in related fields (e.g., Freese, 1961; Swallow and Wehner, 1986).

Adjusting the effective sample size is normally accomplished by adjusting the design parameters for the sampling methods, which in turn affect the areas of the individual inclusion zones. Assume in what follows that the tract resolution and population are fixed. In some cases, such as comparing protocols within the same sampling method, no adjustment is necessary. This is the case, for example, when comparing the two protocols for distance limited sampling, since, for a given distance limit, the inclusion zones are the same regardless of which of the two measurement protocols are used

(Gove et al., 2012b). A similar argument can be made for comparing measurement protocols under perpendicular distance sampling for a given selection protocol (Gove et al., 2012a).[12] However, when comparing the same measurement protocol under different selection protocols for perpendicular distance sampling, the design parameters would require adjustment to arrive at similar effective sample sizes (Gove et al., 2012a). Comparing unrelated methods, such as distance limited sampling and perpendicular distance sampling can be problematic, because the shapes of the inclusion zones are quite different, the latter depending on log taper. An alternative way to approach this problem might be to apply a modeling approach similar to Smith (1938) to inclusion zones, rather than plots (e.g., Wensel and John, 1969) as described in the following section (§ 3.2).

In the examples that follow the above concepts are illustrated using two closely related sampling methods that are by now somewhat familiar. The distance limited sampling method forms a rectangular inclusion zone around a downed log as shown in Figures 4 and 5 with width equal to D_l m on either side of the log's needle,[13] yielding inclusion area $a_{k_d} = 2D_lL_k$, where L_k is the length of the kth log. The sausage sampling method adds a half circle with radius $R \equiv D_l$ m to each end of the rectangle to form the inclusion zone (Figure 1); the inclusion area for this method is therefore $a_{k_s} = 2RL_k + \pi R^2$. Setting a_{k_s} equal to the inclusion zone area for the log of average length in the population under distance limited sampling and solving for R will give design parameters that yield effective sample sizes that are resonably comparable when comparing the two methods (Gove et al., 2012b).

The following code snippet creates the population of logs that will be used in the simulations. It then creates the "distanceLimited" ("ArealSampling" subclass) object and combines the two to generate a population of distance limited sampling inclusion zones ("distanceLimitedIZ") on the buffered tract created in Section 2.3. The collection of logs is then coerced to a "data.frame" and both the average length and inclusion zone area are determined in order to find the effective sample size for the sausage method. Lastly, the radius, R, for sausage sampling that will yield an effective sampling effort similar to that of distance limited sampling with a distance limit of D_l m is found by solving the quadratic given above for R; *viz.*,

```
R> nlogs = 25
R> dlogs = downLogs(nlogs, buffTr, buttDiams = c(20, 40), logLens = c(2, 10),
+                   startSeed = 12354)
R> dl = distanceLimited(distanceLimit = 5)
R> dls.izs = downLogIZs(dlogs, iZone = 'distanceLimitedIZ', dls = dl)
R> logs = as(dlogs, 'data.frame')
R> avgLen = mean(logs[,'logLen'])
R> avgArea = mean(unlist(lapply(dls.izs@iZones, area)))
R> avgArea - 2*dl@distanceLimit*avgLen  #both are equivalent
```

[12] Under perpendicular distance sampling logs can be selected into the sample with probability proportional to volume, coverage, or surface area, and the inclusion zones are different for each protocol.

[13] The term 'needle' refers to a straight longitudinal axis that is determined in crooked or branch logs, often from the most proximal point (of previous limb attachment, or stump) at the large end, to the most distal tip on the small end. On straight logs, it corresponds directly to the pith of the log (for examples, see de Vries, 1986, p. 269).

```
[1] 0


R> a = pi
R> b = 2 * avgLen
R> c = -avgArea
R> D = b^2 - 4 * a * c
R> R = (-b + sqrt(D)) / (2 * a)
R> R


[1] 2.8579466
```

A reasonable plot radius under sausage sampling corresponding to a distance limit of $D_l = 5$ m is therefore approximately 2.9 m for this population of logs[14]. The population of inclusion zones for sausage sampling can now be constructed as...

```
R> saus.izs = downLogIZs(dlogs, iZone = 'sausageIZ', plotRadius = R)
R> c(avgArea, mean(unlist(lapply(saus.izs@iZones, area))))


[1] 59.896 59.896
```

The average inclusion zone sizes are indeed the same, and should, therefore, lead to similar effective sample sizes for comparison of the methods. But it should be noted that the distribution of inclusion areas will be different for the two methods, so this type of analysis yields only a reasonably close, not perfect, sampling effort for comparisons. The respective sampling surface can now be created as...

```
R> dls.ssv = sampSurf(dls.izs, buffTr, estimate = 'volume')


Number of logs in collection = 25
Heaping log: 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,


R> saus.ssv = sampSurf(saus.izs, buffTr, estimate = 'volume')


Number of logs in collection = 25
Heaping log: 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,
```

[14] In practice, one would probably opt for using a radius of 3 m for simplicity in the field.

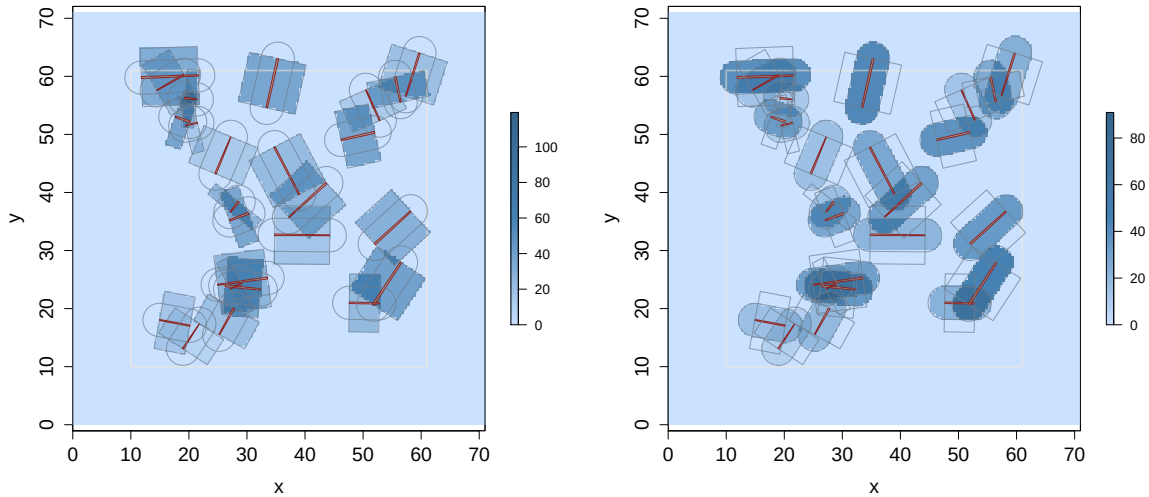
```
R> sampSurf::summary(dls.ssv)


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

Inclusion zone objects: distanceLimitedIZ
Measurement units = metric
Number of logs = 25
True log volume = 7.5427602 cubic meters
True log length = 149.74 meters
True log surface area = 115.73779 square meters
True log coverage area = 36.816255 square meters
True log biomass = NA
True log carbon = NA

Estimate attribute: volume
Surface statistics...
  mean = 7.5551136
  bias = 0.012353352
  bias percent = 0.16377761
  sum = 152341.31
  var = 253.67651
  st. dev. = 15.927226
  cv % = 210.81385
  surface max = 119.3505
  total # grid cells = 20164
  grid cell resolution (x & y) = 0.5 meters
  # of background cells (zero) = 15587
  # of inclusion zone cells = 4577


R> sampSurf::summary(saus.ssv)


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

Inclusion zone objects: sausageIZ
Measurement units = metric
```

```
Number of logs = 25
True log volume = 7.5427602 cubic meters
True log length = 149.74 meters
True log surface area = 115.73779 square meters
True log coverage area = 36.816255 square meters
True log biomass = NA
True log carbon = NA

Estimate attribute: volume
Surface statistics...
  mean = 7.5285768
  bias = -0.014183409
  bias percent = -0.18804004
  sum = 151806.22
  var = 255.09389
  st. dev. = 15.971659
  cv % = 212.14712
  surface max = 90.946799
  total # grid cells = 20164
  grid cell resolution (x & y) = 0.5 meters
  # of background cells (zero) = 15631
  # of inclusion zone cells = 4533
```

Plotting the results is trivial...

```
R> plot(dls.ssv, useImage=FALSE, cex.axis = 1.4, cex.lab = 1.4,
+      xlab = 'x', ylab = 'y')
R> plot(saus.izs, add = TRUE, showLog = FALSE, izColor = NA)
R> plot(saus.ssv, useImage=FALSE, cex.axis = 1.4, cex.lab = 1.4,
+      xlab = 'x', ylab = 'y')
R> plot(dls.izs, add = TRUE, showLog = FALSE, izColor = NA)
```

Approximately 22.7 percent of the cells were covered for distance limited and 22.5 percent for the sausage method, showing a comparable sampling effort for the two simulations. Regardless of the statistic used for comparison, the two methods appear to be indistinguishable for this particular population of logs. One interesting point to consider in visually comparing the methods in Figure 6 is simply the way the total attribute density gets spread over the respective inclusion zones between the two methods. This difference results in different surfaces, not only because the estimators are slightly different, but because of the way the total attribute density accumulates over the tract. Even though both methods are essentially equal in sampling effort, the juxtaposition of the major axis of the zones relative to the logs (on average, perpendicular to the log for distance limited, parallel for sausage), creates a considerably different surface visually, but again with very similar overall statistics.



(a) Distance limited (b) Sausage

Figure 6: Left: A sampling surface simulation using the default Horvitz-Thompson protocol of distance limited sampling for the estimation of aggregate volume in the population with $D_l = 5$ m. Inclusion zones for sausage sampling with a plot radius of $R = 2.9$ m are shown for comparison. Right: The sausage sampling surface has been plotted with outlines of distance limited inclusion zones.

3.2 The Smith plot: sampling effort made simple

Smith (1938) was evidently the first to discover what may be considered a universal 'empirical law' describing the relationship between sample plot size and variance, which can be described by a simple model.[15] Denote the average inclusion zone area to be $\bar{a}$, then the general variance function for Smith's law is

$$\mathrm{Var}\,(\bar{a}) = V\bar{a}^{-\beta} \tag{11}$$

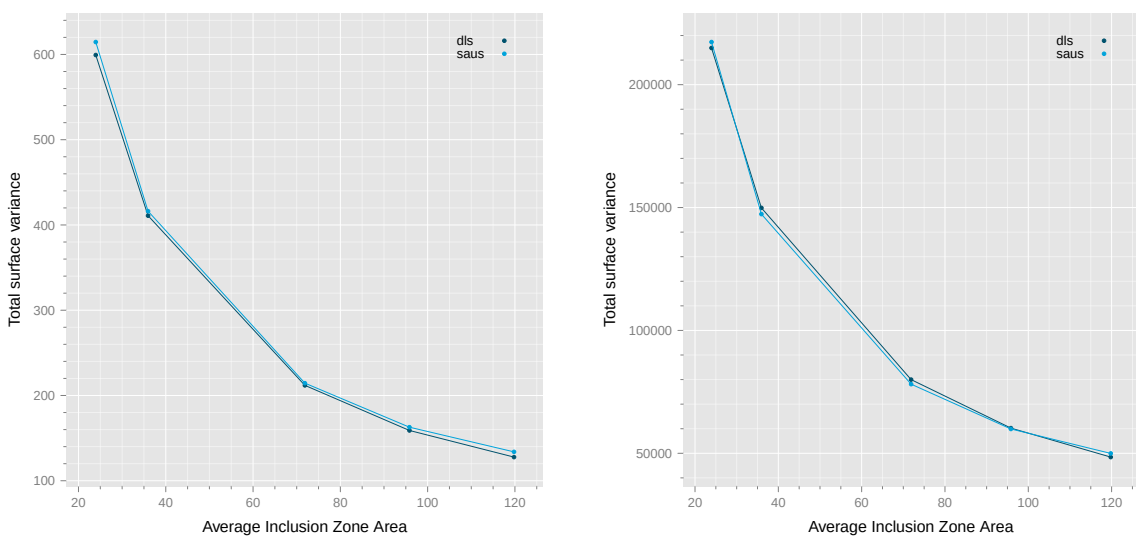
where $\mathrm{Var}\,(\bar{a})$ is the variance in volume (or other attribute; e.g., length, basal area) among plots of size $\bar{a}$, while V, a scaled variance whose definition depends on that of plot size, and $0 \leq \beta \leq 1$ the slope parameter, are both constants that can be estimated if desired. The model, (11) can be used to model variance functions for different sampling methods (not limited to trees or logs), but for the current application it simply serves to illustrate that the variance will decline in a close to exponential manner as the average plot—or inclusion zone—area increases for each given sampling method. Recent comprehensive reviews of the literature pertaining mostly to the law's application in forestry are given by Lynch (2017a,b) and Yang et al. (2017). Also, Gove (2017a) reviewed the

[15] It is interesting to note that while Deming (1950, p. 206) mentions Smith, he apparently attributes the principle discovery of the variance function (11) to a set of unpublished studies by Mahalanobis, evidently culminating in Mahalanobis (1940) and extended in Mahalanobis (1944).

usefulness of this relationship by way of graphical application only for comparing sampling methods, and extended its use to the application of wavelet analysis (§ 6) to sampling surface results.

To demonstrate the usefuleness of the Smith plot, the population of down logs created in § 3.1 is again used with DLS and sausage sampling, but this time a total of four 'plot' (average inclusion zone) sizes are used. Thus, there are four simulations for each sampling method, one for each plot size, all on the same population. In the following example, we have again equalized the sampling effort as in § 3.1; however, this was done only to make a nice looking plot where all the points align, and is by no means necessary for the use of Smith's relation (11) or the Smith plot.[16] The **R** method `makeHFSPlot` used below simply creates these sets of four simulations for both sampling methods. In the first line, the results are for volume, while in the second, length. The routine calls the `sampSurf` routine `smithPlot` to generate the `lattice` plot objects...

```
R> hfs.v = makeHFSPlot(nlog = 25, estimate='volume', Dl = c(2,3,6,8,10),
+                      showPlot = FALSE, startSeed = 12354)
R> hfs.l = makeHFSPlot(nlog = 25, estimate='Length', Dl = c(2,3,6,8,10),
+                      showPlot = FALSE, startSeed = 12354)
```



(a) Volume (b) Length

Figure 7: The Smith plots for (a) volume and (b) length.

The documentation for `smithPlot`[17] notes that two objects are returned (invisibly) from the call to the method. One, as just mentioned, is the `lattice` plot object, which may be plotted as in

[16] If non-equalized 'plot' sizes are used, then it is possible for the lines to cross as an artifact, when they would not otherwise. This can also be due to too few simulations per method.

[17] Type `?smithPlot` for this information.

Figure 7. The other is the data frame object that was used to create the plot; this is shown in the following for volume only...

```
R> hfs.v$hfs.plt$df


            sampMeth          id      mean       var        sd   izArea
Dl.2             dls        Dl.2 7.5392786 599.33502 24.481320  23.9584
Dl.3             dls        Dl.3 7.5348431 410.89357 20.270510  35.9376
Dl.6             dls        Dl.6 7.5551961 211.86871 14.555711  71.8752
Dl.8             dls        Dl.8 7.5476569 158.97343 12.608466  95.8336
Dl.10            dls       Dl.10 7.5487018 127.75650 11.302942 119.7920
plotRad.1.4     saus plotRad.1.4 7.5398181 614.56947 24.790512  23.9584
plotRad.2       saus   plotRad.2 7.5651835 416.20177 20.401024  35.9376
plotRad.3.2     saus plotRad.3.2 7.5361459 214.47686 14.645028  71.8752
plotRad.3.9     saus plotRad.3.9 7.5330131 162.81084 12.759735  95.8336
plotRad.4.6     saus plotRad.4.6 7.5452096 133.87877 11.570599 119.7920
```

Notice that the average inclusion zone areas are indeed equalized (see the `izArea`) column between the two methods. In addition, both the row names and the `id` column in the data frame show the design parameter values that produced these average inclusion zone sizes. For example, a DLS parameter of $D_l = 2$ m in the first row, corresponds to a plot radius of 1.4 m under sausage sampling, and both share $\bar{a} = 23.9584$ m^2.

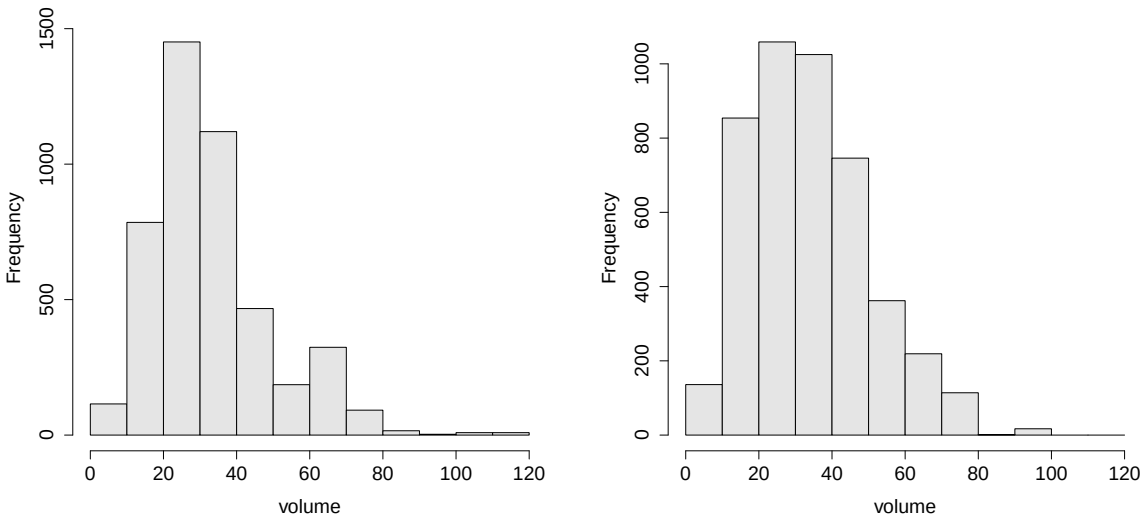
3.3 When is n sufficiently large?

One approach that can be of some help in comparing sampling methods is to look at the number of samples required for nominal confidence interval coverage of the sample mean. It is well-known from the Central Limit Theorem that under random sampling, the distribution of sample means converges to a Gaussian distribution as the sample size increases, regardless of the shape of the underlying population sampling distribution (Barrett and Goldsmith, 1976; Barrett and Nutt, 1979, p. 38). The sampling surface grid cell estimates stored within a "sampSurf" object provides the population sampling distribution necessary to perform convergence analysis for a given attribute and stem population. The "monte" class was designed to handle such simulation experiments for "sampSurf" as well as more general objects (Gove, 2012d).

In the example of the last section, the population sampling distributions for aggregate volume for each of the two sampling surface objects can be visualized using the `hist` method for the class...

```
R> dls.hist = hist(dls.ssv, cex.axis = 1.4, cex.lab = 1.4)


Histogram is zero-truncated: 15587 zeros excluded.
```

```
R> hist(saus.ssv, breaks = dls.hist$breaks, cex.axis = 1.4, cex.lab = 1.4)


Histogram is zero-truncated: 15631 zeros excluded.
```



(a) Distance limted (b) Sausage

Figure 8: The zero-truncated sampling distributions for the surfaces shown in Figure 6 under distance limited (left) and sausage sampling (right).

By default, `hist` shows the zero-truncated sampling distributions for "sampSurf" objects. The reason for this is that the number of background cells (zero valued) can be large, and depends on both the log population, grid cell resolution, and tract extents—all factors that can produce large zero-inflation. So, while zero is a reasonable estimate (no logs selected on a sample point) and can be included, they will be ignored here. Figure 8 presents the sampling distribution for volume for both surface estimates. In both cases the distribution is mildly positively skewed, with the distance limited method showing a few slightly larger estimates, which is in accord with the larger maximum value of the surface for this method shown earlier in the object summary.

The Monte Carlo experiments of repeated sampling from the respective population distributions can be accomplished using the `monte` constructor generic. The following examples draw 2500 Monte Carlo samples from each population for different sample sizes ($n = 10, 25, 50$) and presents in terms of capture rates for each method...

```
R> dlsv.monte = monte(dls.ssv, n = c(10, 25, 50), mcSamples = 2500,
+                     type = 'normalTheory', startSeed = 38015)
R> dlsv.monte
```

```
Estimate attribute = volume

Population...
  Mean = 33.284097
  Variance = 261.20903
  Standard Deviation = 16.161963
  Total = 152341.31
  Size (N) = 4577
  Zero-truncated = TRUE
  Sample sizes (n) = 10, 25, 50
  Finite population corrections = 0.9978, 0.9945, 0.9891
  Variance of the mean = 26.063833, 10.391291, 5.1671107
  Standard error of the mean = 5.1052751, 3.2235526, 2.273128

Normal theory results...
Number of Monte Carlo samples = 2500
Sample sizes: n = 10, 25, 50
Sample summary statistics (mean values)...
               n.10        n.25        n.50
mean     33.3605519  33.2119185  33.3620688
var     265.1566816 261.1493005 262.9704523
stDev    15.5813876  15.8699587  16.0612071
VarMean  26.4577357  10.3889151   5.2019543
stErr     4.9218818   3.1653116   2.2589570
lowerCI  22.2264816  26.6790365  28.8225247
upperCI  44.4946221  39.7448005  37.9016129

Percentage of confidence intervals (95%) that caught the population mean...
 n.10  n.25  n.50
94.08 94.48 95.12

Bootstrap results...
  No bootstrap information available.


R> sausv.monte = monte(saus.ssv, n = c(10, 25, 50), mcSamples = 2500,
+                      type = 'normalTheory', startSeed = 38015)
R> sausv.monte


Estimate attribute = volume

Population...
  Mean = 33.489129
```

```
  Variance = 265.33203
  Standard Deviation = 16.289015
  Total = 151806.22
  Size (N) = 4533
  Zero-truncated = TRUE
  Sample sizes (n) = 10, 25, 50
  Finite population corrections = 0.9978, 0.9945, 0.9890
  Variance of the mean = 26.474669, 10.554748, 5.2481071
  Standard error of the mean = 5.1453541, 3.2488071, 2.2908747

Normal theory results...
Number of Monte Carlo samples = 2500
Sample sizes: n = 10, 25, 50
Sample summary statistics (mean values)...
               n.10       n.25        n.50
mean     33.3887993  33.358711  33.4799880
var     262.1226674 264.200132 263.8309945
stDev    15.7076671  16.074307  16.1579622
VarMean  26.1544413  10.509722   5.2184176
stErr     4.9617185   3.205984   2.2724435
lowerCI  22.1646122  26.741885  28.9133418
upperCI  44.6129864  39.975537  38.0466342

Percentage of confidence intervals (95%) that caught the population mean...
 n.10  n.25  n.50
93.44 93.64 94.68

Bootstrap results...
  No bootstrap information available.
```

The results again show that the two sampling methods are very comparable and evidently converge in terms of coverage at similar rates such that somewhere around $n = 50$ sample points would be required. However, a couple caveats are in order with any analysis of this kind. First, the results will depend on the number of Monte Carlo samples drawn for each sample size, and individual runs can be quite variable, such that in many cases a larger number of Monte Carlo samples would be reasonable. In repeated runs on these populations, a sample size of $n = 50$ produced the desired nominal coverage on average for both methods, similar to those shown in the results above. Second, these results are applicable only for populations with similar sampling distributions; even though this is a half-hectare tract, the results would generalize to a larger area provided it had similar variability. However, if there is much deviation from the population surface characteristics (e.g., more highly skewed sampling distribution), then extrapolation to a larger population and area would be inappropriate. In a more general context where the goal is to judge the overall performance (i.e., covering a range of possible underlying sampling distributions) of one sampling

method against another, it would be more appropriate to simulate multiple realizations of each sampling surface with various different populations of logs and average the results to get a better idea of the estimator performance when not conditioned on a particular log population. Lastly, it should also be clear based on the discussion in the previous section that changing the design parameters (D_l and R) will change the results, because the total attribute density will be spread differently over the tract for the same log population.

There are many factors that complicate analyses like the ones performed here, which makes comparison of sampling methods and subsequent efficient inventory design a difficult problem for a population with unknown characteristics. Log populations, grid resolution and associated tract extents, and sampling methods with their respective design parameters to be compared all play a central role. Several studies (Affleck, 2008; Gove et al., 2012a,b) have noted that the sampling distributions for population attributes are often less well-behaved than those shown here and can vary from negative exponential through multimodal; and in several PPS methods when sampling for the design attribute, the sampling distributions are discrete at levels that are multiples of certain design parameters.[18] These realizations of the underlying sampling distribution led Affleck (2008) to question whether even bootstrap intervals would be a more robust alternative in such cases. To help answer this question in future studies, the "monte" class and associated constructor generic provides the capability for bootstrap intervals to be calculated as desired in addition to, or in lieu of the usual normal theory intervals. Finally, histograms showing the distribution of sample means at each sample size can be displayed using the `hist` method on the respective "monte" objects as follows...

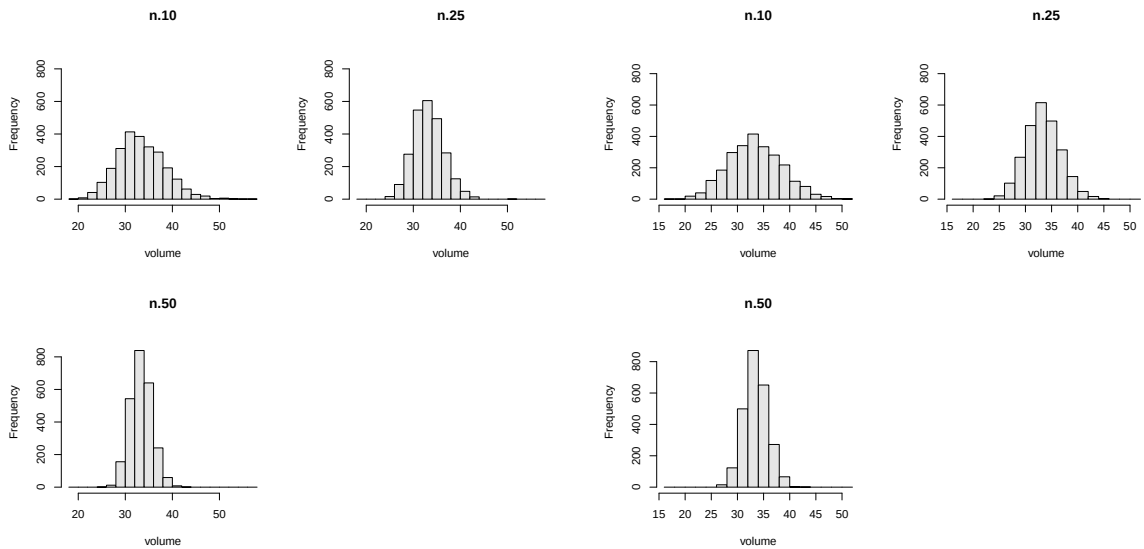
```
R> hist(dlsv.monte)
R> hist(sausv.monte)
```

Note how the distribution of the sample mean for volume in Figure 9 gets progressively more normally distributed as the sample size increases—lending added confidence to the normal theory approach to sample size determination for this example.

4 Monte Carlo Subsampling

This section describes some features within `sampSurf` that apply on the individual "Stem" class level. While we still refer to it as 'Monte Carlo' sampling, it is therefore different than what was described in § 3.3, where Monte Carlo sampling was applied to a `sampSurf` object. In this section Monte Carlo sampling is used to subsample a "standingTree" or "downLog" object at certain points along the bole (or log) and record measurements such as diameter and height (length) at those points. These samples then allow us to estimate the volume of the stem in an unbiased fashion using Monte Carlo integration (MCI). `sampSurf` supports three main methods of Monte

[18] For example, perpendicular distance sampling with probability proportional to volume will produce sampling distributions in discrete steps of the volume factor design parameter.



(a) Distance limited (b) Sausage

Figure 9: The sampling distributions for mean volume from the Monte Carlo subsampling results applied to the populations in Figure 6 under distance limited (left) and sausage sampling (right).

Carlo integration: crude Monte Carlo (CMC), importance sampling, and control variate sampling (CVS). Each of these also has an antithetic sampling counterpart, which is also supported. These methods are described in detail in Gregoire and Valentine (2008, p. 106). In addition, `sampSurf` has several built-in 'proxy' functions for use with these methods that range from very simple for crude Monte Carlo, to fairly complex for the proxy that employs the same taper function that is used by default in `sampSurf` "Stem" class objects (Gove, 2011c, p. 8). The Monte Carlo integration methods and their implementation in `sampSurf`, including details on available proxy functions (and how to write your own) are thoroughly discussed in Gove (2013a).

4.1 Subsampling "Stem" class objects

This section demonstrates how the MCI subsampling methods can be employed in `sampSurf` independent of any areal sampling approach, while § 4.2 illustrates how these may be coupled with areal sampling methods in `sampSurf`. The following example uses importance sampling to form the MCI estimate for the stem to arrive at an estimate of tree volume...

```
R> stree = standingTree(dbh = 12, topDiam = 2, height = 30, solidType = 2.4,
+                       units = 'English')
R> stree.is = importanceSampling(stree, n.s = 20, startSeed = 545)
R> stree.is
```

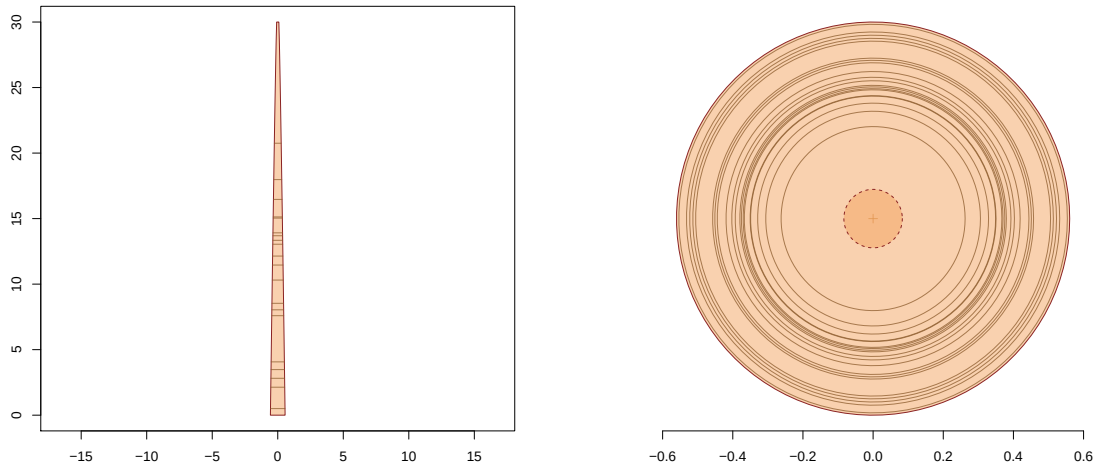
```
Object of class: importanceSampling
------------------------------------------------------------
Importance Sampling
------------------------------------------------------------
Original Stem object class: standingTree
Proxy taper function: gvProxy
Full tree height = 30
Segment height bounds = 0 to 30
True volume = 12.787049
Volume estimate = 12.580296
Relative error % = -1.6168878
Variance estimate = 0.072020457
0.95% confidence Interval = 12 to 13.14
Number of samples n = 20


R> stree.is@hgt.s


 [1] 13.04472276 20.75114463 13.34345096  7.59193508  3.47748327 13.69156283
 [7]  2.13432361  8.03958754  4.06721210 15.12756008  0.50207012 17.97646001
[13] 15.03907246 12.14110829 10.30960091 13.91387687  8.53622240 11.45597947
[19] 16.46490212  2.81414273
```

Note that the `n.s` argument to `importanceSampling` specifies the number of subsample measurements within the stem, in this case 20. Figure 10 presents a graphical portrayal of these results. Notice that importance sampling concentrates the sample points within the tree to the lower part of the bole with the maximum height sampled within the tree at 20.8 ft. The diameter range for the sampled points is 13.3 to 6.29 in.[19] Importance sampling gives a very close estimate to the true volume of the tree. Use of a different proxy, fewer or different[20] subsample points, antithetic sampling, etc., will all change this estimate. The other MCI methods, CMC and CVS can be employed on this stem in the same manner as importance sampling.

The following code was used to generate Figure 10...

```
R> plot(stree.is, axes = TRUE)
R> plot(stree.is, renderAs = 'crossSection')
R> axis(1)
```

[19] The diameters corresponding to the heights in the `hgt.s` slot are stored in the `diam.s` slot for the object.
[20] As specified by the `startSeed` function argument.



(a) Profile

(b) Cross Section

Figure 10: Importance sampling applied to a single tree showing the sampling points in (a) profile view, and (b) cross-sectional view.

4.2 Subsampling areal sampling methods

This section describes the methods available in `sampSurf` that allow combining the individual stem subsampling protocols from § 4.1 with common areal sampling methods as a way to generate unbiased estimates of volume, carbon and biomass. These methods will be termed 'spatially unstructured' or 'spatially unconstrained' for reasons that will be described presently. Other Monte Carlo integration methods are available in `sampSurf`, but they are different than what is described in this section in the sense that the selection of the subsampling point on the stem depends on the juxtaposition of the inventory sample point to the log or tree, based on distance or orientation, or both. These methods are therefore 'spatially constrained' or 'spatially structured,' whereas the MCI subsampling described in this section is independent of the inventory sample point's location with respect to the stem. For "downLog" objects, the 'chainsaw' method is perhaps the most complicated in the sense that the MCI estimate for volume at any given inventory sample (i.e., raster cell) point is determined by the distance and orientation (e.g., at the ends) of the sample point to stem (Gove and Van Deusen, 2011). A simpler method, where the estimate for volume is determined only by the perpendicular distance of the sample point from the stem is the Monte Carlo variant of distance limited sampling (Gove et al., 2012b). For "standingTree" class objects within `sampSurf`, critical height sampling and its variants (Figure 2) are also spatially constrained MCI sampling methods. They are again somewhat different from the methods discussed in this section as the subsample point on the tree depends on the distance to the tree; interested readers can refer to Lynch and Gove (2013) for more information and other references on the subject.

One important point is that the spatially-constrained methods subsample the stem at *only one point* so that MCI is based only on this point. In contrast, with the spatially unconstrained methods discussed here, where the methods of § 4.1 are coupled with areal sampling methods, any number of subsample points can be taken from a given inventory point location to develop the MCI estimate for the stem on that inventory point. Both schemes might be envisioned as two-stage sampling where the stem is selected in the first stage via areal sampling, and then subsampled for measurements in the second; the second-stage MCI estimate of volume for the tree is then expanded by the appropriate areal sampling factors.

Currently, there is only one set of two-stage estimators available within `sampSurf` as depicted in Figure 2. However, it is straightforward to incorporate Monte Carlo subsampling with any of the standard areal sampling methods, the code used to create the currently supported methods under horizontal point sampling can be used as a guide for extending to other methods. The following presents an example where horizontal point sampling with subsampling via CMC is used to generate a sampling surface corresponding to the collection of trees in § 2.6.1...

```
R> hpscmc.izs = standingTreeIZs(trees, iZone = 'horizontalPointCMCIZ',
+                               angleGauge = ag.as,
+                               description = 'horizontal point sampling CMC IZs')
R> hpscmc.ss = sampSurf(hpscmc.izs, rPlot.btr)


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


R> sampSurf::summary(hpscmc.ss)


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

Inclusion zone objects: horizontalPointCMCIZ
Measurement units = metric
Number of trees = 10
True tree volume = 5.4930582 cubic meters
True tree basal area = 0.6542992 square meters
True tree surface area = 83.118769 square meters
True tree biomass = NA
True tree carbon = NA

Estimate attribute: volume
Surface statistics...
```

```
mean = 5.4776587
bias = -0.015399527
bias percent = -0.28034523
sum = 54776.587
var = 89.870933
st. dev. = 9.4800281
cv % = 173.06715
surface max = 53.671919
total # grid cells = 10000
grid cell resolution (x & y) = 0.5 meters
# of background cells (zero) = 6546
# of inclusion zone cells = 3454
```

The results above are for only one subsample point per tree, which makes it very similar to CHS, again the difference between the two comes from the spatial structure in the latter. There is one other potentially subtle point where the two differ. Under CHS the entire tree has a well distributed set of diameters sampled from the ground to the tip within the inclusion zone as a consequence of the spatial constraint of the sampling points. However, CMC subsampling at each inventory point within the inclusion zone can select any point on the stem from any grid point within the inclusion zone (spatially unconstrained). This means that there is no guarantee that a uniform distribution of subsample points within the tree will arise from the inventory points within the inclusion zone; indeed, some points on the tree may be selected more than once, and it could happen just by chance, that all points are clustered near the tip of the tree, providing a poor, though asymptotically unbiased MCI estimate. This situation is unlikely in practice, especially with a reasonable grid resolution providing many estimates within the inclusion zone per stem. Also, larger trees, where more complete coverage of the stem is important for volume estimation will have more inventory points, and thus subsamples, due to the larger inclusion area under HPS (of course this would not be true for, e.g., simple fixed area plot sampling).

A graphical representation of the sampling surface for HPS+CMC is shown in Figure 11...

```
R> plot(hpscmc.ss, useImage = FALSE)
```

Notice that the surface height (volume) is variable within each tree's inclusion zone (independent of the overlap areas). The variability within each tree's inclusion zone represents the different estimates for the tree's volume at each grid point. A pure HPS sampling surface of the same trees would be completely flat within each tree's inclusion zone (again, independent of the overlap areas).[21]

[21] It is very simple to verify this, and instructive to view both surfaces using the display capabilities mentioned in § 6.

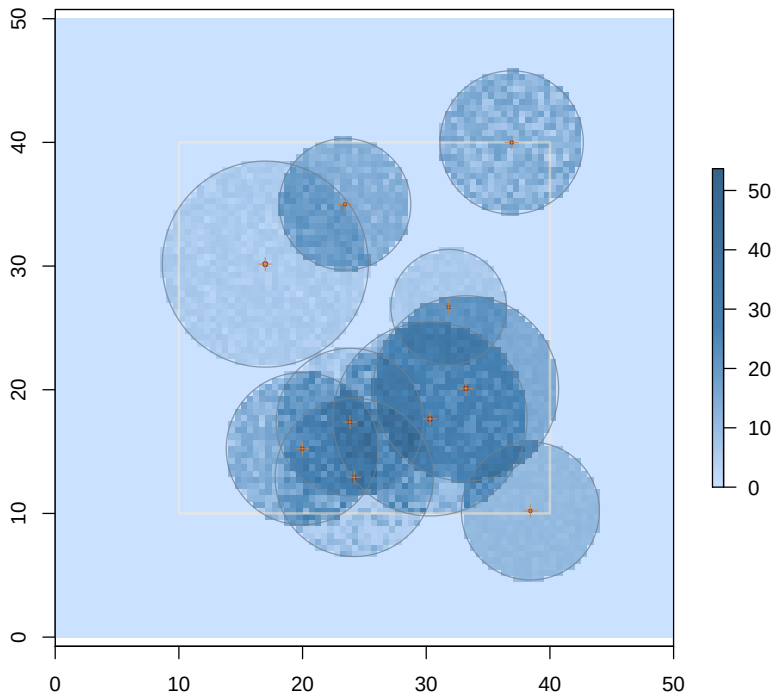


Figure 11: The sampling surface results for two-stage HPS with CMC subsampling.

5 Extending sampSurf

The S4 class-method structure of the package facilitates the extension of the package through the addition of new sampling methods. The discussion in this section is limited to the addition of a new sampling method, though extensions of other major classes such as "Stem" and "Tract" or their subclasses are certainly possible. The essential steps necessary for adding a fixed-area square plot method for sampling standing trees is developed. The code presented is a functional, but "bare bones" version. No validity error checking is done on the new object or in the methods to conserve space. There are ample examples of how these should be added to the code presented below in the package code itself

The example for square plots adds the functionality to the user's workspace (`.GlobalEnv`). A consequence of this is that a number of new "hidden" objects are automatically created by **R** in the workspace in order to keep track of the new S4 classes and methods and reconcile them with the already existing code in the `sampSurf` package. The steps below are not exhaustive and more is required if the code were to be added to the package proper; the full set of steps required for

this, including documentation, is found in (Gove, 2012a). In some cases, methods such as `summary` will work at least partially on new subclasses through inheritance. To save space these methods are not extended here, there are many examples in the package illustrating their implementation.

The steps presented below are roughly in the order in which one would logically extend the package, with the exception that some of the helper methods (e.g., `plot`, `perimeter`) are presented in the final subsection. In the actual sequence of coding, it would be advantageous to define these methods as their related objects are developed in order to be able to check the objects graphically.

5.1 Extending the "ArealSampling" class

The first logical step in adding a new sampling method or protocol is to add the required code to implement the method as a subclass of "ArealSampling". The "circularPlot" class that is in `sampSurf` and the new class for sampling with square plots to be added are two examples of probability proportional to frequency sampling methods that do not depend on some dimension of the "Stem" object to be able to map the inclusion zone. Therefore, for these objects, we can construct the graphical components of the object in the "ArealSampling" subclass in general, though when it comes to actually constructing an inclusion zone, the plot location will depend on the location of the associated "Stem" object. The class definition follows

```
R> require(sampSurf)
R> setClass('squarePlot',
+          representation(radius = 'numeric',
+                         area = 'numeric',
+                         perimeter = 'SpatialPolygons',
+                         location = 'SpatialPoints'
+                        ),
+          contains = 'ArealSampling', where = .GlobalEnv)
R> showClass('squarePlot')


Class "squarePlot" [in ".GlobalEnv"]

Slots:

Name:           radius            area       perimeter        location
Class:         numeric         numeric SpatialPolygons   SpatialPoints

Name:      description           units
Class:       character       character

Extends: "ArealSampling"
```

Note that these are not generalized square plots because they can take on only one orientation. A subclass could be defined that would allow any orientation and also allow for differing radii for rectangular plots.[22]

In addition to the `units` and `description` slots that the class inherits from the base class, the following new slots are defined...

- `radius` The radius or half-width of the square plot.
- `area` The exact area of the plot.
- `perimeter` A "SpatialPolygons" object (Bivand et al., 2008, p. 41) available for plotting the perimeter of the square plot.
- `location` The location of the center point of the plot as a "SpatialPoints" object (Bivand et al., 2008, p. 30).

Each of the slots such as `radius` and `area` are in the appropriate units and should be consistent, this is something validity checking would normally assure when creating an object.

The construction of objects of class "squarePlot" is best accomplished using a constructor function as described previously, an example of which follows

```
R>   setGeneric('squarePlot',
+              function(radius, ...) standardGeneric('squarePlot'),
+              signature = c('radius'), where = .GlobalEnv)


[1] "squarePlot"


R> setMethod('squarePlot', signature(radius = 'numeric'),
+ function(radius,
+          units = 'metric',
+          centerPoint = c(x = 0, y = 0),
+          spUnits = CRS(projargs = as.character(NA)),
+          spID = paste('sp', .StemEnv$randomID(), sep = ':'),
+          ... )
+ {
+     if(radius < 0)
```

[22] Note that the argument `where=.GlobalEnv` is not necessary in general, it is used here to make sure that the class is assigned in the user's workspace, thus circumventing KNITR's locked environment where code chunks normally get processed—removing it will cause an error in the `assign`ment of the class when this code is run in KNITR, not at the command line.

```
+       radius = abs(radius)
+     area = 4 * radius * radius
+     sqPlot = matrix(c(centerPoint['x'] + radius*c(-1, -1, 1, 1, -1),
+                       centerPoint['y'] + radius*c(-1, 1, 1, -1, -1)),
+                     nrow = 5)
+     pgSqPlot = Polygon(sqPlot)
+     pgsSqPlot = Polygons(list(sqPlot = pgSqPlot), ID = spID)
+     spSqPlot = SpatialPolygons(list(pgsSqPlot = pgsSqPlot),
+                                 proj4string = spUnits)

+     loc = matrix(centerPoint, nrow = 1)
+     colnames(loc) = names(centerPoint)
+     location = SpatialPoints(loc, proj4string = spUnits)

+     return(new('squarePlot', radius = radius, area = area,
+                perimeter = spSqPlot, units = units, location = location))
+ },
+ where = .GlobalEnv)


[1] "squarePlot"
```

The function determines the size of the plot and its location, then goes on to build a "SpatialPolygons" object from a matrix representation of the plot corners that is a closed polygon. Finally, the "SpatialPoints" object for the center point is made and the object is created using the general `new` method.[23]

5.2 Extending the "InclusionZone" class

Like circular plots, a square fixed-area plot should be aligned with its center at the pith of the tree at ground level. The class definition for the construction of square plots is very simple as most of the slots and behavior are inherited from the "StandingTreeIZ" class

```
R> setClass('squarePlotIZ', representation(squarePlot = 'squarePlot'),
+           contains = 'standingTreeIZ', where = .GlobalEnv)
```

The `squarePlot` slot is the only new slot, and holds an object of that class.[24] The rest of the slots are described in Gove (2012b).

[23] Again, using the `where = .GlobalEnv` is *not* necessary in practice, only within KNITR.
[24] Use `showClass('squarePlotIZ')` to see the inherited slots as well for this class.

A constructor function is again required to make new objects of class "squarePlotIZ". First the generic is defined and then the method itself; *viz.*,

```
R>   setGeneric('squarePlotIZ',
+              function(standingTree, plotRadius, ...)
+                standardGeneric('squarePlotIZ'),
+              signature = c('standingTree', 'plotRadius'),
+              where = .GlobalEnv
+             )


[1] "squarePlotIZ"


R> setMethod('squarePlotIZ',
+           signature(standingTree = 'standingTree', plotRadius = 'numeric'),
+ function(standingTree, plotRadius, description = 'square plot IZ',
+          spID = paste('sp', .StemEnv$randomID(), sep = ':'),
+          spUnits = CRS(projargs = as.character(NA)), ...)
+ {
+     units = standingTree@units
+     loc = coordinates(standingTree@location)[1,]
+     squarePlot = squarePlot(plotRadius, units = units, centerPoint = loc,
+                             spID = spID, spUnits = spUnits)
+     bbox = bbox(perimeter(squarePlot))
+     baFactor =  ifelse(standingTree@units == .StemEnv$msrUnits$English,
+                        .StemEnv$baFactor['English'],
+                        .StemEnv$baFactor['metric'])
+     unitArea = ifelse(standingTree@units == .StemEnv$msrUnits$English,
+                       .StemEnv$sfpAcre,
+                       .StemEnv$smpHectare)
+     puaBlowup = unitArea / squarePlot@area
+     puaEstimates = list(standingTree@treeVol * puaBlowup,
+                         puaBlowup,
+                         standingTree@dbh^2 * baFactor * puaBlowup,
+                         standingTree@surfaceArea * puaBlowup,
+                         standingTree@biomass * puaBlowup,
+                         standingTree@carbon * puaBlowup
+                        )
+     names(puaEstimates) =
+              .StemEnv$puaEstimates[c('volume', 'Density', 'basalArea',
+                                      'surfaceArea', 'biomass', 'carbon')]
+     return( new('squarePlotIZ', standingTree = standingTree,
+                 squarePlot = squarePlot, bbox = bbox, spUnits = spUnits,
+                 description = description, units = units,
```

```
+                 puaBlowup = puaBlowup, puaEstimates = puaEstimates) )
+ },
+ where = .GlobalEnv)


[1] "squarePlotIZ"
```

Several things are going on in the above method. First, note that the routine accepts a `plotRadius` argument rather than an object of class "squarePlot". The reason for this is the latter is created from the tree location in appropriate units. Second, an overall graphical bounding box is calculated for the object.[25] To simplify the code, it is assumed that the square plot will always be larger than the tree's diameter. This is not, in general, a safe assumption and the "circularPlotIZ" constructor adds the requisite code to safeguard against this assumption being violated. Next, the per unit area estimates are calculated for several different attributes using constants and standardized names from the internal `.StemEnv` environment, and finally the object is create with a call to `new`.

5.3 Extending the "InclusionZoneGrid" class

There will rarely be a need to define a new subclass to the "InclusionZoneGrid" base class. The only sampling method that requires this in the package thus far is that for the "chainsaw" protocol (Gove and Van Deusen, 2011) for sampling down woody debris with a fixed-area plot. It will, however, be required to define a constructor function for this class for most new sampling methods. Methods that have a flat sampling surface—where the estimates within an object's inclusion zone are all the same for any given attribute—will have a trivial definition. This is the case with the "InclusionZoneGrid" constructor for "squarePlotIZ" objects. In other cases, where the inclusion zone surface varies, it will be necessary to develop an algorithm that visits every grid point within the zone and assign the appropriate estimate to each cell based on the protocol estimator. Examples of methods where this is the case have been given earlier.

For any sampling method with a constant surface, the following function suffices as a model, so that one can simply replicate this function with the appropriate signature to handle any such new method. Examples in the code can be examined for varying-surface methods.

```
R> setMethod('izGrid', signature(izObject = 'squarePlotIZ', tract = 'Tract'),
+ function(izObject, tract, description = 'squarePlotIZ grid object',
+          wholeIZ = TRUE, ...)
+ {
+     return( izGridConstruct(izObject = izObject, tract = tract,
+                             description = description,
+                             wholeIZ = wholeIZ, ...) )
```

[25] See the `perimeter` method defined below.

```
+ },
+ where = .GlobalEnv)


[1] "izGrid"
```

The method has one call to a function that creates the background grid and registers it to the grid of the associated "Tract" object, while assigning every point within the inclusion zone a zero value, with background cells assigned `NA`. A dataframe object contains the values for each of the different attribute estimates; these can be swapped into the grid as desired for estimates of different attributes.

5.4 Helper function extensions

The only functions that are extended here to the new sampling method are those that are actually required by the methods above or by the `sampSurf` constructor itself. As mentioned previously, methods like `summary` should also be extended for full functionality of the new objects.

5.4.1 Methods for plotting

The different `plot` methods build on each other and so, as mentioned above, would normally be defined as the class structure and constructor methods are defined in the programming flow. Here they are collected into one place to facilitate comparison. Note that `suppressWarnings` may be required when arguments are passed that do not match any of those in the base graphics graphical parameters list (`par`), which will throw a warning. We have illustrated its typical use in the first method below...

```
R> setMethod('plot', signature(x = 'squarePlot', y = 'missing'),
+ function(x, axes = FALSE, ...) {
+     suppressWarnings(
+       plot(perimeter(x), axes = axes, border = .StemEnv$izBorderColor,
+            asp = 1, ...) )
+     callNextMethod(x, pchIZCenter = 3, ...)
+     return(invisible())
+ },
+ where = .GlobalEnv)


[1] "plot"
```

This `plot` function draws the perimieter of the square plot in the default border color and then calls the superclass method for displaying the center point. The method for "squarePlotIZ" uses the previous function; *viz.*,

```
R> setMethod('plot', signature(x = 'squarePlotIZ', y = 'missing'),
+ function(x, axes = FALSE, add = FALSE, ... ) {
+     if(!add)
+       callNextMethod(x, axes = axes, asp = 1, ...)
+     plot(x@squarePlot, axes = axes, add = TRUE, ...)
+     plot(x@standingTree, add = TRUE, ...)
+     return(invisible())
+ },
+ where = .GlobalEnv)


[1] "plot"
```

The `callNextMethod` call sets up the extents for the plot from the overall bounding box if it is a new plot. Otherwise, the square plot (inclusion zone) and then the tree are displayed. It is reasonable to make, e.g., the display of the tree optional—this is done in other methods—but again for succinctness, this option was not included here.

5.4.2 Other required methods

The only other required definitions are for `perimeter` and `area` methods for both "squarePlot" and "squarePlotIZ" class objects, these are very simple...

```
R> setMethod('perimeter', signature(object = 'squarePlot'),
+   function(object, ...) return(object@perimeter), where = .GlobalEnv)


[1] "perimeter"


R> setMethod('perimeter', signature(object = 'squarePlotIZ'),
+   function(object, ...) return(perimeter(object@squarePlot)), where = .GlobalEnv)


[1] "perimeter"


R> setMethod('area', signature(x = 'squarePlot'),
+   function(x, ...) return(x@area), where = .GlobalEnv)
```

```
[1] "area"


R> setMethod('area', signature(x = 'squarePlotIZ'),
+   function(x, ...) return(area(x@squarePlot)), where = .GlobalEnv)


[1] "area"
```

5.5 Remarks

The different methods that require extension will depend on the nature of the class hierarchy that is established for the new sampling method. Some new subclasses do not add new slots that require new methods. The horizontal point sampling class, "horizontalPointIZ", is one such class. It is closely related to sampling with fixed-area circular plots and the inheritance structure requires little new functionality. This is because each inclusion zone object corresponding to each tree in the population has a variable-sized circular plot associated with it that acts, for all intents and purposes, like a regular fixed-area plot (Figure 3). So there is little to do when extending the classes and methods for this sampling method. For example, the method for constructing an "InclusionZoneGrid" object for horizontal point sampling is the exact same as for the fixed-area "circularPlotIZ" superclass, so that superclass method is automatically dispatched on via inheritance, requiring no new coding. It should be clear then that a little thought in defining future class structures could save much work in coding new methods.

Finally, no new methods should ever be required for generating objects of class "sampSurf" itself. The `sampSurf` constructors know how to work with any sampling method that applies to standing trees or downed logs.

5.6 An example

Having defined all of the necessary components for sampling standing trees with fixed-area square plots, a simple illustration of how to use the classes follows. First, make a collection of large diameter trees on a small plot ("Tract") so the trees will resolve in the figure. The other steps follow as in the previous examples. Recall that the sampling surface default is for volume estimation.

```
R> stract = Tract(c(x = 25, y = 25), cellSize = 0.5)
R> sbuffTr = bufferedTract(5, stract)
R> strees = standingTrees(4, sbuffTr, dbhs = c(30, 60), startSeed = 98921)
R> sp.izs = standingTreeIZs(strees, iZone = 'squarePlotIZ', plotRadius = 4)
R> spv.ss = sampSurf(sp.izs, sbuffTr)
```

```
Number of trees in collection = 4
Heaping tree: 1,2,3,4,


R> sampSurf::summary(spv.ss)


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

Inclusion zone objects: squarePlotIZ
Measurement units = metric
Number of trees = 4
True tree volume = 6.0132777 cubic meters
True tree basal area = 0.71954 square meters
True tree surface area = 55.229433 square meters
True tree biomass = NA
True tree carbon = NA

Estimate attribute: volume
Surface statistics...
  mean = 6.0132777
  bias = 0
  bias percent = 0
  sum = 15033.194
  var = 94.458785
  st. dev. = 9.7189909
  cv % = 161.62551
  surface max = 34.427044
  total # grid cells = 2500
  grid cell resolution (x & y) = 0.5 meters
  # of background cells (zero) = 1664
  # of inclusion zone cells = 836
```

Note that the simple **sampSurf** constructor could also be used, but the long way provides an illustration of all the main component functions. Plotting the surface is then trivial...

```
R> plot(spv.ss, gridLines = TRUE, useImage = FALSE)
```

Finally, Figure 12 presents the two-dimensional rendering of the sampling surface generated in the example above. As mentioned in Section 2.7, the figure shows what might be misconstrued as a

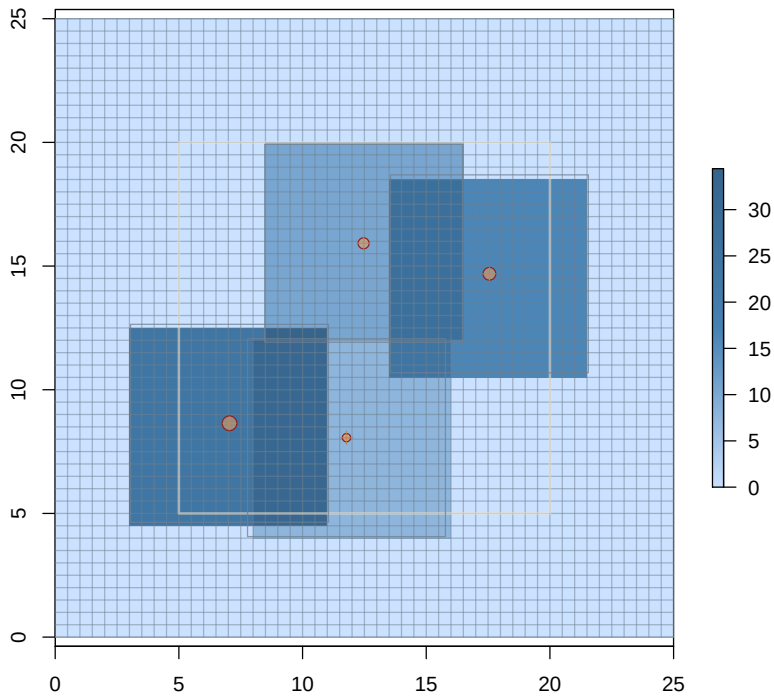


Figure 12: A volume estimation sampling surface example for square fixed-area plots with standing trees.

registration problem between the polygon and the underlying grid-based inclusion zones for several of the objects. In this case, it appears more obvious than in Figure 4 because of the square polygonal inclusion zones that conform to the underlying grid. The apparent misalignment of inclusion zone polygon to underlying grid occurs solely because the center of the tree is not exactly aligned with one of the of the grid cell intersections; the background grid has been shown in this example to more clearly illustrate the phenomenon. As in the "InclusionZoneGrid" example (Section 2.7), if the grid cell resolution were increased, the two would eventually align for all trees in the figure.

6 Conclusions

The `sampSurf` package was designed and implemented using the powerful S4 class-methods paradigm, which allows great flexibility and extensibility within the **R** framework. The package also draws heavily from the `sp` and `raster` packages, both written largely in S4, providing more opportunities for class extensions in the future. The main classes and flow of the system for conducting simula-

tions has been reviewed here. An example of extending the package for a new sampling method has also been presented to demonstrate the flexibility of the design under S4. Other components to `sampSurf` exist that have not been discussed here. The extensive online help system can be accessed from the **R** prompt with `package?sampSurf`, and provides a mechanism for accessing documentation on all of the components of the system including those not discussed here. In addition, there are several vignettes available in the package that have also been cited. These provided examples and more details on class structure and creation.

The fact that `sampSurf` is built upon the feature-rich `sp` and `raster` spatial packages, which provide many of the graphical and spatial capabilities, is an asset that can not be overemphasized, as the package was conceived to be one where graphical presentation plays an important role not only for research, but also for teaching these different sampling methods. Additional graphical capabilities also exist through the suggested package `rgl` (Alder and Murdoch, 2017), which provides real-time three-dimensional rendering of several of the object classes (e.g., "InclusionZoneGrid" and "sampSurf") through an extension to `raster`'s `plot3D` method.

A companion package, `ssWavelets` (Gove, 2017b), increases the potential usefulness of the `sampSurf` package. Wavelets can be used to decompose the sampling surface variance by scale (distance/area) in a sort of analysis of variance manner. The package provides a useful graphical analysis framework, again based on the `sp` and `raster` packages (and, of course, `sampSurf`) for visual presentation. Analytical summaries are also provided, much like those in `sampSurf`. An introduction to the use of wavelets in areal sampling is found in Gove (2017a). The *R-Forge* web site provides vignettes further illustrating the use of the package.

The supported sampling methods and protocols (Table 1 and Figure 2) includes a number of different techniques, but the current list of 34 (including the different protocols associated with some methods) is still modest considering the breadth of existing areal sampling methods that have been developed to date in forestry, ecology, and related fields. As the package matures further and includes more methods, it has the potential to become a platform for a comprehensive collection of sampling methods that can be used for method comparison, and as an aid in the design of inventories and other new methods in the future.

7 Acknowledgements

The author expresses his appreciation to Dr.'s Mark J. Ducey and Harry T. Valentine for their helpful reviews of this manuscript.

References

D. L. R. Affleck. A line intersect distance sampling strategy for downed wood inventory. *Canadian Journal of Forest Research*, 38:2262–2273, 2008. 40

D. Alder and D. Murdoch. *rgl: 3D Visualization Device System (OpenGL)*, 2017. URL http://CRAN.R-project.org/package=rgl. R package version 0.98.1. 57

A. Baddeley and R. Turner. spatstat: An R package for analyzing spatial point patterns. *Journal of Statistical Software*, 12(6):1–42, 2005. 16

J. P. Barrett and L. Goldsmith. When is n sufficiently large? *The American Statistician*, 30:67–70, 1976. 36

J. P. Barrett and M. E. Nutt. *Survey Sampling in the Environmental Sciences: A Computer Approach*. COMPress, Inc., Wentworth, NH, USA, 1979. 36

W. Bitterlich. Die winkelzählprobe. *Allg. Forst Holzwirtsch. Ztg.*, 59(1–2):4–5, 1948. 2, 17

R. S. Bivand, E. J. Pebesma, and V. Gómez-Rubio. *Applied Spatial Data Analysis with* R. Springer-Verlag, 2008. 12, 13, 48

J. M. Chambers. *Software for Data Analysis: Programing with* R. Springer-Verlag, 2008. 10, 11

P. G. de Vries. *Sampling Theory for Forest Inventory*. Springer-Verlag, 1986. 30

W. E. Deming. *Some theory of sampling*. John Wiley (Dover), 1950. 34

M. J. Ducey, J. H. Gove, and H. T. Valentine. A walkthrough solution to the boundary overlap problem. *Forest Science*, 50(4):427–435, 2004. 2

M. J. Ducey, M. S. Williams, J. H. Gove, and H. T. Valentine. Simultaneous unbiased estimates of multiple downed wood attributes in perpendicular distance sampling. *Canadian Journal of Forest Research*, 38:2044–2051, 2008. 23

F. Freese. A relationship of plot size to variability. *Journal of Forestry*, 59:679, 1961. 29

R. Gentleman. R *Programming for Bioinformatics*. Chapman & Hall/CRC, Boca Raton, 2008. 10, 11

J. H. Gove. *The "*ArealSampling*" Class*, 2011a. URL http://CRAN.R-project.org/package=sampSurf. sampSurf package vignette. 16

J. H. Gove. *The "*InclusionZoneGrid*" Class*, 2011b. URL http://CRAN.R-project.org/package=sampSurf. sampSurf package vignette. 26

J. H. Gove. *The "*Stem*" Class*, 2011c. URL http://CRAN.R-project.org/package=sampSurf. sampSurf package vignette. 15, 41

J. H. Gove. *The "Tract" Class*, 2011d. URL http://CRAN.R-project.org/package=sampSurf. sampSurf package vignette. 13

J. H. Gove. *Extending the sampSurf Package*, 2012a. URL http://CRAN.R-project.org/package=sampSurf. sampSurf package vignette. 47

J. H. Gove. *The "InclusionZone" Class*, 2012b. URL http://CRAN.R-project.org/package=sampSurf. sampSurf package vignette. 19, 49

J. H. Gove. *The "sampSurf" Class*, 2012c. URL http://CRAN.R-project.org/package=sampSurf. sampSurf package vignette. 26

J. H. Gove. *"monte": When is n Sufficeintly Large?*, 2012d. URL http://CRAN.R-project.org/package=sampSurf. sampSurf package vignette. 36

J. H. Gove. *Monte Carlo Sampling Methods in sampSurf*, 2013a. URL http://sampsurf.r-forge.r-project.org/. sampSurf vignette, only available on R-Forge. 41

J. H. Gove. *The Mirage Method in sampSurf*, 2013b. URL http://sampsurf.r-forge.r-project.org/. sampSurf vignette, only available on R-Forge. 14

J. H. Gove. Some refinements on the comparison of areal sampling methods via simulation. 2017a. Submitted. 34, 57

J. H. Gove. The ssWavelets package: Wavelet functionality for package sampSurf, 2017b. URL http://sswavelets.r-forge.r-project.org/. 57

J. H. Gove. *sampSurf: Sampling Surface Simulation for Areal Sampling Methods*, 2017c. URL http://CRAN.R-project.org/package=sampSurf. R package version 0.7-4. 1, 3

J. H. Gove and P. C. Van Deusen. On fixed-area plot sampling for downed coarse woody debris. *Forestry*, 84(2):109–117, 2011. 3, 4, 17, 19, 23, 43, 51

J. H. Gove, A. Ringvall, G. Ståhl, and M. J. Ducey. Point relascope sampling of downed coarse woody debris. *Canadian Journal of Forest Research*, 29(11):1718–1726, 1999. 17

J. H. Gove, M. S. Williams, G. Ståhl, and M. J. Ducey. Critical point relascope sampling for unbiased volume estimation of downed coarse woody debris. *Forestry*, 78:417–431, 2005. 3, 23

J. H. Gove, M. J. Ducey, H. T. Valentine, and M. S. Williams. A comprehensive comparison of the perpendicular distance method for sampling downed coarse woody debris. *Forestry*, 2012a. (In Press). 3, 30, 40

J. H. Gove, M. J. Ducey, H. T. Valentine, and M. S. Williams. A distance limited method for sampling downed coarse woody debris. *Forest Ecology and Management*, 282:53–62, 2012b. 3, 17, 23, 29, 30, 40, 43

T. G. Gregoire and H. T. Valentine. *Sampling Strategies for Natural Resources and the Environment*. Champan & Hall/CRC, Boca Raton, 2008. 3, 4, 6, 8, 18, 41

L. R. Grosenbaugh. Point-sampling and line-sampling: Probability theory, geometric implications, synthesis. Occasional Paper 160, Southern Forest Experiment Station, USDA Forest Service, 1958. 3, 17

R. J. Hijmans. *raster: Geographic Analysis and Modeling with Raster Data*, 2012. URL http://CRAN.R-project.org/package=raster. R package version 1.9-92. 13

K. Iles. Some techniques to generalize the use of variable plot and line intersect sampling. In W. E. Frayer, editor, *Forest resource inventories workshop proceedings, volume 1*, pages 270–278, Fort Collins, CO, 1979. Colorado State University, Colorado State University. 3

J. A. Kershaw, M. J. Ducey, T. Beers, and B. Husch. *Forest Mensuration*. Wiley-Blackwell, 5th edition, 2016. 3, 18

J. A. Kershaw Jr., E. W. Richards, J. B. McCarter, and S. Oborn. Spatially correlated stand structures: A simulation approach using copulas. *Computers and Electronics in Agriculture*, 74: 120–128, 2010. 16

M. Kitamura. On an estimate of the volume of trees in a stand by the sum of critical heights. *Kai Nichi Rin Ko*, 73:64–67, 1962. 3, 19

T. B. Lynch. Optimal plot size or point sample factor for a fixed total cost using the Fairfield Smith relation of plot size to variance. *Forestry*, 90:211–218, 2017a. 34

T. B. Lynch. Optimal sample size and plot size or point sampling factor based on cost-plus-loss using the Fairfield Smith relationship for plot size. *Forestry*, 2017b. Advanced Access Online. 34

T. B. Lynch and J. H. Gove. An antithetic variate to facilitate upper-stem height measurements for critical height sampling with importance sampling. 43(12):1151–1161, 2013. 17, 23, 43

T. B. Lynch and J. H. Gove. The unbiasedness of a generalized mirage boundary correction method for monte carlo integration estimators of volume. 44(7):810–819, 2014. 14

P. C. Mahalanobis. A sample survey of the acreage under jute in bengal. *Sankhyā*, 4(4):511–530, 1940. 34

P. C. Mahalanobis. On large-scale sample surveys. *Philisophical Transactions of the Royal Society*, B231:329–451, 1944. 34

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2017. URL http://www.R-project.org. ISBN 3-900051-07-0. 1

P. Schmid-Haas. Stichprobenam waldrand (sampling at the edge of the forest). *Mitt Schweiz Anst Forstl Versuchswes*, 45(3):234–303, 1969. 14

H. F. Smith. An empirical law describing heterogeneity in the yields of agricultural crops. *Journal of Agricultural Science*, 28:1–23, 1938. 29, 30, 34

G. Ståhl, J. H. Gove, M. S. Williams, and M. J. Ducey. Critical length sampling: A method to estimate the volume of downed coarse woody debris. *European Journal of Forest Research*, 129: 993–1000, 2010. 3

W. H. Swallow and T. C. Wehner. Optimum plot size determination and its application to cucumber yield trials. *Euphytica*, 35:421–432, 1986. 29

H. T. Valentine, D. A. Herman, J. H. Gove, D. Y. Hollinger, and D. S. Solomon. Initializing a model stand for process based projection. *Tree Physiology*, 20:393–398, 2000. 16

P. C. Van Deusen and J. H. Gove. Sampling coarse woody debris along spoked transects. *Forestry*, 84(2):93–98, 2011. 17

L. C. Wensel and H. H. John. A statistical procedure for combining different types of sampling units in a forest inventory. *Forest Science*, 15(2):307–317, 1969. 30

M. S. Williams. New approach to areal sampling in ecological surveys. *Forest Ecology and Management*, 154:11–22, 2001a. 3, 8, 10, 29

M. S. Williams. Nonuniform random sampling: An alternative method of variance reduction for forest surveys. *Canadian Journal of Forest Research*, 31:2080–2088, 2001b. 3, 8, 10, 29

M. S. Williams and J. H. Gove. Perpendicular distance sampling: An alternative method for sampling downed coarse woody debris. *Canadian Journal of Forest Research*, 33:1564–1579, 2003. 3, 17

T. R. Yang, Y. H. Hsu, J. A. Kershaw, 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*, 2017. Advanced Access Online. 34