The “sampSurf” Class

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

Thursday 20th January, 2011
12:38pm

Contents

1 Introduction

The sampling surface is the culmination of the different classes and methods that have been developed and discussed in other vignettes. It can be created in any one of several ways (i.e., via different constructors); here we illustrate the overall process of creating a “sampSurf” object using one of the constructors (others are detailed below). Please refer to the individual vignette documentation for more details on each of the methods used below.

First we need a “Tract” object (in this case, a “bufferedTract” object) within which we can create the sampling surface…

```
R> cd = c(x=200, y=200)
R> tra = Tract(cellSize = 0.5,
+              cellDims = cd,
```

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

```
+             cellCenter = c(0.25, 0.25),
+             data = 0,
+             units = 'metric',
+             description = 'a 1-hectare tract')


Grid Topology...
                       x      y
cellcentre.offset   0.25   0.25
cellsize            0.50   0.50
cells.dim         200.00 200.00


R> btr = bufferedTract(10, tra)
R> btr


------------------------------------------------------------
a 1-hectare tract
------------------------------------------------------------
Measurement units = metric
Area in square meters = 10000 (1 hectares)

class       : bufferedTract
filename    :
nrow        : 200
ncol        : 200
ncell       : 40000
min value   : 0
max value   : 0
projection  : NA
xmin        : 0
xmax        : 100
ymin        : 0
ymax        : 100
xres        : 0.5
yres        : 0.5

Buffer width =  10
```

Here we have created a 200×200 cell raster grid with resolution of 0.5 meters (i.e., 1 hectare), with origin at $(0, 0)$ meters. Then we create a buffered tract object with a 10-meter buffer internal to the tract. Both versions have all data values as zero to begin so we can build a sampling surface up.

Next, just create a few down logs whose centers lie within the buffer and place them in a "downLogs" container, then make sausage sampling inclusion zones from these logs with plot radius of five meters, and store them in a "downLogIZs" containter. The final step is trivial in concept, just accumulate all of the inclusion zones with the `sampSurf` constructor...

```
R> dlogs = downLogs(5, btr@bufferRect)
R> izsSA = downLogIZs(lapply(dlogs@logs, 'sausageIZ', plotRadius=5))
R> ssSA = sampSurf(izsSA, btr)
```

```
Logs in collection = 5
Heaping log: 1,2,3,4,5,
```

```
R> summary(ssSA)
```

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

Inclusion zone objects: sausageIZ
Estimate attribute: cubicVolume
Measurement units = metric
Number of Logs = 5
True Log volume = 0.47989022
Surface statistics...
  mean = 0.47850501
  bias = -0.0013852094
  bias percent = -0.28865131
  sum = 19140.200
  var = 8.1627881
  st. dev. = 2.8570593
  st. error = 0.014285297
  total # grid cells = 40000
  # of background cells (zero) = 37322
  # of inclusion zone cells = 2678
```

We can see from the summary statistics by comparing the mean of the surface with the true volume for all logs, that the method is unbiased. Finally, we are ready to plot the surface...

```
R> plot(ssSA, useImage=FALSE)
```

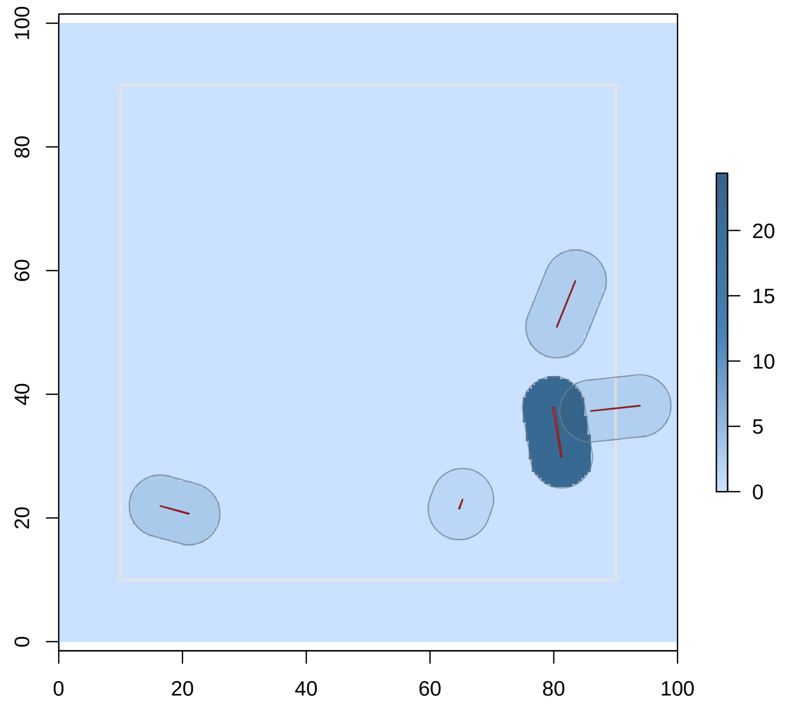

Figure 1: Simple generic "sampSurf" object with some random logs. fig:ss

The result is shown in figure fig:ss 1.

2 The "sampSurf" Class

Now that we have shown the basic individual steps to creating a sampling surface we formally introduce the class itself and its constructors.

The base class is defined with the slots. . .

```
R> showClass('sampSurf')
```

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

```
Slots:

Name:  description izContainer       tract    estimate   surfStats
Class:   character  downLogIZs       Tract   character        list
```

2.1 Class slots

- *description*: Some descriptive text about this class.
- *izContainer*: A "'downLogIZs" object, which is a collection of "downLogIZ" objects (see vignette entitled: "The InclusionZone Class"); i.e., this is a container class object.
- *tract*: The underlying "Tract" (or subclass) object which will hold the sampling surface.
- *estimate*: The attribute estimate for the tract; i.e., volume ("cubicVolume"), or number of stems ("Density"), per unit area.[1]
- *surfStats*: A `list` object with the summary statistics for the sampling surface for estimate attribute `object@estimate`.

2.2 More Details

There are just a few things that go on behind the scenes within the construction of the sampling surface object that are good to know (without having to look at the code). The object constructor has two main steps...

1. Create "InclusionZoneGrid" objects from each of the individual sausage "InclusionZone" objects within the "downLogIZs" container.
2. Use `heapIZ` to accumulate the individual "InclusionZoneGrid" objects from the first step into the tract.

The above is applied within a simple loop, one object at a time. Then when the "heaping" has been completed, the "sampSurf" object is constructed. It may take some time to construct the object because there is a fair bit of computation going on within these two steps.

[1]Other attributes can be added as required in the future.

3 Object Creation: Other Constructors

The first constructor is shown in the example above. It takes as the signature both the "InclusionZone" container object and the "Tract" object. The second constructor for sampling surface objects generates the logs, inclusion zones, and surface from scratch. It takes the number of logs and a "Tract" object, along with the type of inclusion zone to generate and simulates the surface from that. This method is simpler if no collection of logs is available. An example follows for the sausage and standUp sampling protocols...

```
R> ssSA = sampSurf(2, btr, iZone = 'sausageIZ', plotRadius=5,
+                  buttDiam=c(30,50), startSeed=102)
```

```
Logs in collection = 2
Heaping log: 1,2,
```

```
R> summary(ssSA)
```

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

Inclusion zone objects: sausageIZ
Estimate attribute: cubicVolume
Measurement units = metric
Number of Logs = 2
True Log volume = 1.3466069
Surface statistics...
  mean = 1.3515718
  bias = 0.0049648839
  bias percent = 0.36869585
  sum = 54062.872
  var = 75.190628
  st. dev. = 8.671253
  st. error = 0.043356265
  total # grid cells = 40000
  # of background cells (zero) = 38906
  # of inclusion zone cells = 1094
```

```
R> ssSU = sampSurf(2, btr, iZone = 'standUpIZ', plotRadius=5,
+                  buttDiams=c(30,50), startSeed=102)
```

```
Logs in collection = 2
Heaping log: 1,2,
```

```
R> summary(ssSU)
```

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

Inclusion zone objects: standUpIZ
Estimate attribute: cubicVolume
Measurement units = metric
Number of Logs = 2
True Log volume = 1.3466069
Surface statistics...
  mean = 1.3446755
  bias = -0.0019313799
  bias percent = -0.14342567
  sum = 53787.022
  var = 133.61430
  st. dev. = 11.559165
  st. error = 0.057795826
  total # grid cells = 40000
  # of background cells (zero) = 39373
  # of inclusion zone cells = 627
```

The examples above generate the same set of logs for each example so the we can directly compare the bias and precision of the methods. This is accomplished by passing `startSeed` with the same random number seed in each call (it gets passed on to the `downLogs` constructor). Note especially that we must pass along any arguments that internal constructors require. For example, within this version of `sampSurf`, we know from the above example (and some knowledge of how the class objects are constructed) that we will be creating "downLogs" and "downLogIZs" objects within this routine. Therefore, we must, for example, pass the `plotRadius` argument so that the `sausageIZ` constructor will know what size circular plot is required—there is no default for this argument.

3.1 The Chainsaw method

This is always a problem because the inclusion zone is a point as explained elsewhere. However, in sampling surface simulations, all points (grid cells) within the sausage-shaped ("sausageIZ") inclusion zone will intersect the log. So when generating a "sampSurf" object for the chainsaw method, we must have a set of "sausageIZ" inclusion zone objects available for the log population first. In the constructor with `signature(object = 'downLogIZs', tract='Tract')`, the "downLogIZs" object must contain sausage inclusion zones; but in addition, we must tell the constructor that we want to use the chainsaw method to generate the sampling surface within these zones. This is done by specifying `wantChainSaw = TRUE` in the `sampSurf` method call. This is also done for the constructor with `signature(object = 'numeric', tract='Tract')`. Additionally, in this second case, we must also specify `iZone = 'sausageIZ'` in the method call. Two examples, that are not run because of the length of time to construct even one full chainsaw inclusion zone are shown below for these respective signatures. . .

```
R> ss.ch1 = sampSurf(izsSA, btr, wantChainSaw=TRUE)
R> ss.ch2 = sampSurf(2, btr, iZone='sausageIZ', wantChainSaw=TRUE,
+                    plotRadius=5, buttDiam=c(40,50))
```

Note especially that we are able to pass along other arguments as needed. For example, in the second invocation, we pass an argument for the determination of the fixed-area plot radius used to generate the sausage inclusion zone, as well as an argument used in sample log generation for `downLogs`.

It can not be stress enough that the chainsaw method is an anomoly with regard to the fixed-area plot methods in that it must visit every cell within the inclusion zone, decide whether there is a plot-log intersection, and then calculate the volume, etc., of the intersected section. This generates a non-constant surface within the inclusion zone and take a long time to calculate if the resolution is high or the log and plot radius are large, encompassing a large number of grid cells. Another method that has uneven surface height within the inclusion zone is the critical point relascope method, but that is not yet implemented.

4 Plotting With "rgl"

A method has been added to the `raster` package generic `plot3D`. To display a surface, one must first have the `rgl` package installed. Then simply issue the command on the "sampSurf" class object to display it, and the `rgl` commands to save it to a file as desired. . .

```
R> require(rgl)
R> plot3D(ssSA)
R> rgl.postscript(paste(getwd(),'/figures/ss3D.ps',sep=''))
R> rgl.close()
```

Figure fig:ssrgl 2 shows a sampling surface with a population of 25 logs. The sausage method was used to sample the logs on a one hectare tract. This example shows one view of the surface, which may be rotated and magnified as desired under the `plot3D` function used to interface with `rgl`.

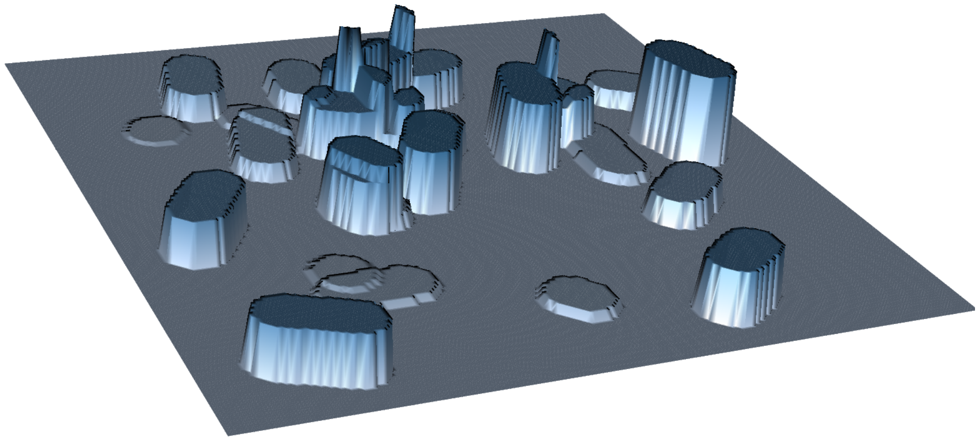

Figure 2: One view of a sampling surface using `rgl`. fig:ssrgl

5 Other Sampling Methods

Some methods may require different arguments to the "sampSurf" constructor with first signature argument "numeric". In general, the help documentation is the best place to go for more information. However, here we present an illuastration using the point realscope sampling method as an example, because it shows an argument that is required as a replacement for `plotRadius` in the fixed-area plot sampling methods.

```
R> etract = Tract(c(x=100,y=100), cellSize=0.5, units='English')
```

```
Grid Topology...
                       x      y
cellcentre.offset   0.25   0.25
cellsize            0.50   0.50
cells.dim         200.00 200.00
```

```
R> ebuffTr = bufferedTract(15, etract)
R> (angle = .StemEnv$rad2Deg(2*atan(1/2)))
```

```
[1] 53.130102
```

```
R> prs.as = pointRelascope(angle, units='English')
R> prs.ss = sampSurf(5, ebuffTr, iZone='pointRelascopeIZ', units='English',
+                    prs=prs.as, butDiams=c(8,16), logLens=c(6,16))
```

```
Logs in collection = 5
Heaping log: 1,2,3,4,5,
```

The point of the above example is that the sampling surface constructor mentioned required two additional bits of information to be passed on to other routines. The `units` specifies, for example, that the down logs to be created are in "English" units, to go along with the correct units in the "bufferedTract" and "pointRelascope" objects. More importantly, the "pointRelascopeIZ" constructor requires an argument `prs` which is of type "pointRelascope", in order to define the relascope information for the inclusion zones which will be constructed internally within this function call. There is nothing else really new here, and nothing mysterious going on. We are just sending the appropriate information along as required buy the constructors for different objects. A plot of this sampling surface is shown in Figure 3.

```
R> plot(prs.ss, axes=TRUE, useImage=FALSE)
```

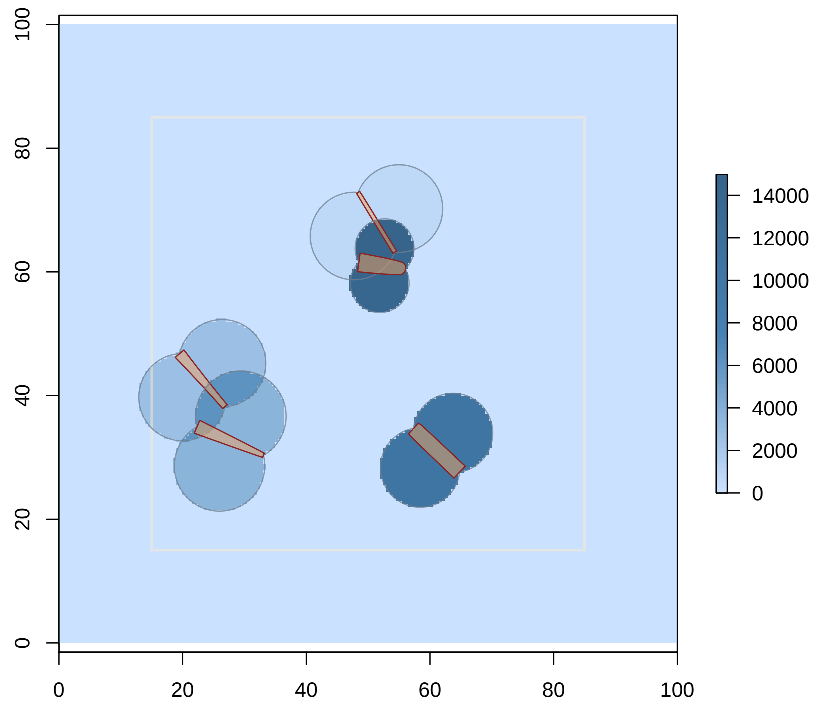

Figure 3: "sampSurf" surface with a few logs under point relascope sampling.

fig:ssprs