The "InclusionZoneGrid" 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

Monday 20th June, 2011
4:16pm

Contents

1 Introduction **1**
2 The "InclusionZoneGrid" Class **2**
2.1 Class slots 2
3 Example: The "standUpIZ" Class **3**
4 On Design Motivation **5**
5 Example: The "sausageIZ" Class **7**
6 Example: The "chainSawIZ" Class **7**
6.1 Snapping to the grid 10
7 The "csFullInclusionZoneGrid" Class **12**
7.1 Class slots 14
7.2 Object Construction and Plotting 14
8 Example: The "pointRelascopeIZ" Class **17**
9 Example: The "perpendicularDistanceIZ" Class **18**
10 Example: The "omnibusPDSIZ" Class **20**
11 Example: The "distanceLimitedIZ" Class **22**
12 Example: The "distanceLimitedMCIZ" Class **24**
13 Example: The "distanceLimitedPDSIZ" Class **25**
14 Example: The "omnibusDLPDSIZ" Class **28**
15 Using `plot3D` **30**
Bibliography **30**

1 Introduction

When we think about building a sampling surface piece by piece from the inclusion zones of individual "Stem" objects, we assign the appropriate attribute value to each grid cell within the inclusion zone for an object, with zero values elsewhere, and then algebraically add this grid layer to the overall "Tract" grid. Thus, we "heap up" the inclusion zone density of the grid cells within the tract, which in the end is a discrete estimate of the sampling surface.

This class allows us to do the geometry associated with the individual "InclusionZone" objects

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

for each "Stem" object in the population. Recall that an "InclusionZone" object has both an "ArealSampling" and "Stem" subclass as slots in its definition. For example, an object of subclass "standUpIZ" would have both a "circularPlot" class object and a "downLog" object making up the overall bounding box. There are methods for the "InclusionZoneGrid" class objects that work with each type of "InclusionZone" object. In general, the methods are all of the name `izGrid`, whose signature objects initiate the appropriate method.

In the following, we show some general constructs of the class with respect to individual subclasses of "InclusionZone" objects they are based on. Graphics play a big role in getting the idea, and each class is a little different, so several illustrations are presented. Whether a surface within an inclusion zone is constant or variable height depends on the sampling method. For example, what we term "canonical" perpendicular distance sampling (PDS) has constant height surfaces, while "omnibus" PDS has surfaces of varying height. In addition, the somewhat odd, but interesting "full chainsaw" method where the entire sausage-based inclusion zone is filled with variable-height individual chainsaw estimates is also shown.

2 The "InclusionZoneGrid" Class

The base class is defined with the slots...

```
R> showClass('InclusionZoneGrid')
```

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

Slots:

Name:    description            iz          grid          data          bbox
Class:     character InclusionZone   RasterLayer    data.frame        matrix

Known Subclasses: "csFullInclusionZoneGrid"
```

2.1 Class slots

- *description*: Some descriptive text about this class.
- *iz*: An object of one of the "InclusionZone" subclasses.
- *grid*: A "RasterLayer" object.
- *data*: A data frame holding the values for each of the per unit area estimates available in the "InclusionZone" object in the columns, with rows for grid cells.

- *bbox*: The overall bounding box for the object, which includes the inclusion zone and the "Stem" subclass object plus the grid. Sometimes the inclusion zone itself includes the stem, but other times it does not.

I made a decision when designing this class to go with the slots above. A possible problem with this design is the very real possiblity of people misunderstanding the class structure. This is because the grid object has values of just zero and `NA`. In other words, it does not store the per unit area estimates within the cells of the grid. These are stored in the `data` slot of the object and can be swapped into the `grid` slot object with a simple command...

```
R> x@grid = setValues(x@grid, x@data[,estimate])
```

where `x` is the "InclusionZoneGrid" object. Unfortunately, again this may cause problems with misunderstanding, down the line as it is going to have to be done whenever one wants an underlying `grid` with real per unit area values. An alternative to this would have been (and still could be) to use a "RasterStack" or "RasterBrick" class for the `grid` slot. This would obviate the need for the `data` slot. But it also has drawbacks, because these objects seem to be designed more for map layers that are algebraically related. I did not want people unwittingly summing layers within this object, for example, and thereby adding things like number of stems and cubic volume. I may reconsider this, however, as the latter approach does have its benefits (even though is takes more storage). Another potential drawback of the latter is that there seems to be no way to name the "layers" within a brick or stack, they simply get assigned numbers, so we'd again have to keep track of this with program code.

When plotting the object, the above substitution gets made automatically, one simply has to specify the desired attribute to plot in the `estimate` argument to the `plot` method. For example, to plot the coverage area surface, specify `estimate = 'coverageArea'` in the plot command. The default is to plot the surface for volume.

3 Example: The "standUpIZ" Class

Refer to the "InclusionZoneClass" vignette for more information on this class.

Here we demonstrate the construction of an "InclusionZoneGrid" object from an object of class "standUpIZ".

```
R> tra = Tract(c(x=100, y=100), cellSize = 0.5, units = 'metric',
+              description = 'a 1-hectare tract')
R> btr = bufferedTract(10, tra)
R> btr
```

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

class       : bufferedTract
dimensions  : 200, 200, 1  (nrow, ncol, nlayers)
resolution  : 0.5, 0.5  (x, y)
extent      : 0, 100, 0, 100  (xmin, xmax, ymin, ymax)
projection  : NA
values      : in memory
min value   : 0
max value   : 0

Buffer width =  10
```

```
R> dlogs = downLogs(1, container=btr@bufferRect, buttDiam=c(30,40),
+                   logLen=c(6,10), topDiams=c(0,0.5), solidTypes=c(2,4),
+                   vol2wgt=20.1, wgt2carbon=0.5)
R> sup = standUpIZ(dlogs@logs$log.1, 3)
R> izgSU = izGrid(sup, btr)
```

Here we have created a 200 by 200 cell raster grid in the form of a "Tract" object, having resolution of 0.5 meters, yielding spatial extents of 100×100 meters, (i.e., a 1 hectare tract), with origin at $(0, 0)$ meters. Then we create a buffered tract object with a 10-meter buffer, and drew a single random "downLog" object within the tract, which we used to create an object of class "standUpIZ" with a 3 meter radius for the circular plot. Finally, using the "standUpIZ" object and the underlying buffered tract grid, we create the "InclusionZoneGrid" object that will be aligned to the tract grid.

Now, in the following example we plot this object...

```
R> plot(izgSU)
```

There are two ways to generate the object which depend on how one wants the underlying grid developed. The argument `wholeIZ` determines how this is done, and defaults to `TRUE`. We can see in Figure 1 that the underlying grid covers the entire inclusion zone plus the down log object. If we wanted to just cover the inclusion zone only, with a minimal bounding grid we would do the following (Figure 2)...

```
R> izmbgSU = izGrid(sup, btr, wholeIZ=FALSE)
R> plot(izmbgSU, gridCenters=TRUE)
```

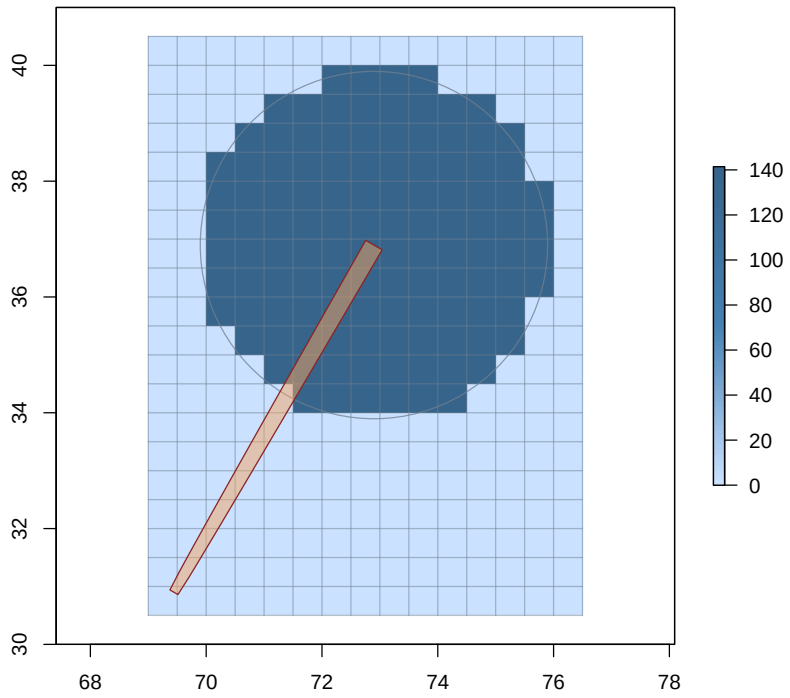

Figure 1: An "InclusionZoneGrid" object based on a "standUpIZ" object.

4 On Design Motivation

Having now seen that we can make the grid object cover more than just the inclusion zone if applicable (i.e., the stem lies partially outside it), we can delve a little more into the motivation for this class. First, it is entirely possible to do approach the sampling surface construction in a more brute-force manner in one of at least two ways...

1. Brute-force method 1...
    - Make an exact duplicate of the tract and assign zeros to all its values.
    - Then overlay the inclusion zone onto this to get a mask.
    - Assign the per acre values to the cells within the masked inclusion zone and zero outside.
    - Algebraically add this layer to the base tract and repeat for all stems.
2. Brute-force method 2...

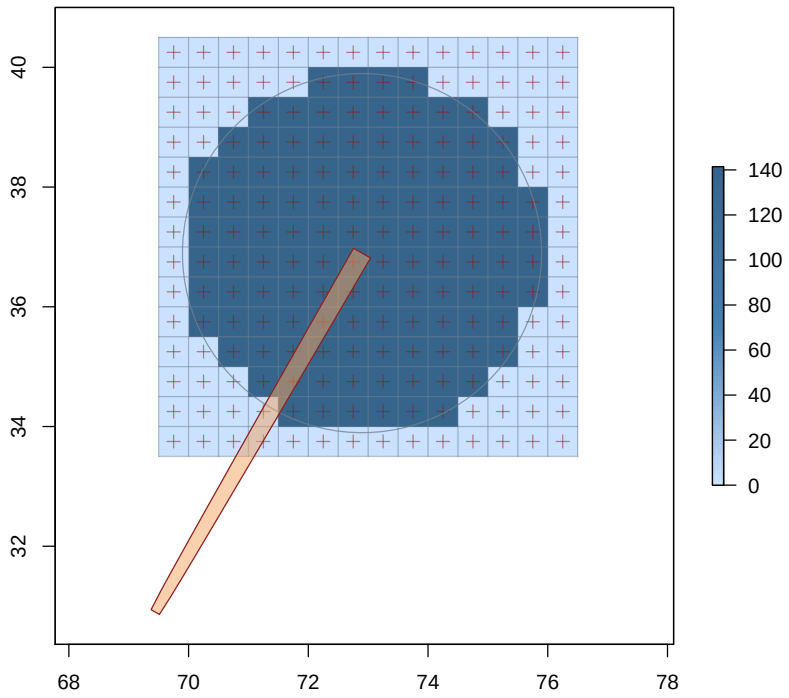

Figure 2: A "InclusionZoneGrid" object based on a "standUpIZ" object where only the actual inclusion zone is covered by the bounding grid, and grid cell centers are plotted as an option.

- Use the `rasterize` function within "raster" with `overlap='sum'` on the "InclusionZone" objects to sum them into the base tract grid.
- Repeat for all stems.

The problem with each of these approaches, even though they work, is the time it takes to implement them. In each case they are working on the entire tract and take significantly more time to do the accumulation than overlaying onto the smaller grid within the "InclusionZoneGrid" class. This is compounded if the tract is large, or the resolution is small, which we want usually for better estimates. This, along with the thought that perhaps the following approach is simpler to understand because one can see the individual surface components graphically, is why I elected to go this route.

In slightly more detail, the steps in accumulating the surface under the "InclusionZoneGrid" scheme are...

1. Create an "InclusionZoneGrid" object.
2. Expand its grid to the extent of the tract.
3. Add this expanded grid to the tract surface.
4. Repeat for all stems.

Conceptually it is similar to the other approaches except that the actual overlay is done on a minimal bounding grid for the object, and therefore is much faster. Expanding the grid mask in the "InclusionZoneGrid" object takes little effort, and adding it is the same in all steps.

5 Example: The "sausageIZ" Class

Using the same grid and tree as above for an example with sausage sampling, we have...

```
R> saus = sausageIZ(dlogs@logs$log.1, 3)
R> izgSAUS = izGrid(saus, btr, wholeIZ=FALSE)
R> plot(izgSAUS)
```

Notice in Figure 3 that whether we specify using the whole inclusion zone or not is immaterial for sausage sampling, because the entire log is always included within the zone. Also note that the minimal bounding grid is always calculated to include the entire zone, even if there is an extra cell padding all around. This is trivial and is necessary for making sure all cells within the zone get assigned the correct value.

6 Example: The "chainSawIZ" Class

Here are a couple similar examples for the "chainSawIZ" class. Gove and Van Deusen (2011) discuss how the inclusion zone is really just a point, the center point of the circular plot that intersects the downed log, rather than the plot itself. Therefore, under this method, we assign the sampling surface value to only that one grid cell that contains the circular plot center point.

```
R> csaw = chainSawIZ(dlogs@logs$log.1, plotRadius = 3,
+                    plotCenter = coordinates(dlogs@logs$log.1@location)[1,] + c(1,-1))
R> izgCSaw = izGrid(csaw, btr, wholeIZ=FALSE)
R> izgCSaw
```

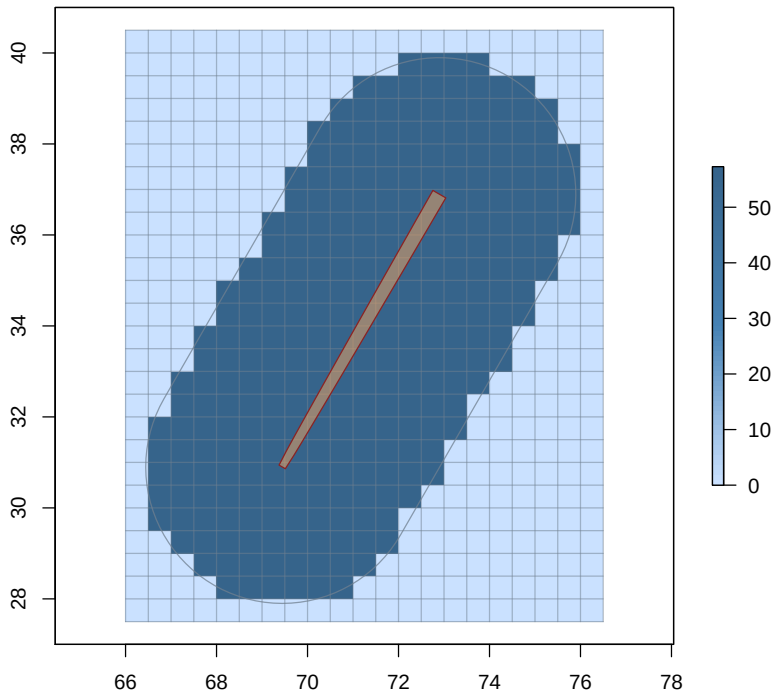

Figure 3: A "InclusionZoneGrid" object based on a "sausageIZ" object showing that the stem and inclusion zone are covered by the bounding grid.

```
Object of class: InclusionZoneGrid
------------------------------------------------------------
chainSaw grid point inclusion zone grid object
------------------------------------------------------------

InclusionZone class: chainSawIZ
  units of measurement:  metric

Grid class: RasterLayer
Number of grid cells = 1
Cell dimensions: (nrows=1, ncol=1)
Grid cell values**...
  gridValues Freq
1          0    1
**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
Min.     87.53   353.7   1932        1562        497.3    1759  879.7
1st Qu.  87.53   353.7   1932        1562        497.3    1759  879.7
Median   87.53   353.7   1932        1562        497.3    1759  879.7
Mean     87.53   353.7   1932        1562        497.3    1759  879.7
3rd Qu.  87.53   353.7   1932        1562        497.3    1759  879.7
Max.     87.53   353.7   1932        1562        497.3    1759  879.7


  Encapulating bounding box...
        min       max
x 69.171709 75.170199
y 29.898866 36.978535
```

```
R> plot(izgCSaw, gridCenters=TRUE)
```

Figure 4 shows the concept for a given circular plot location showing volume in cubic meters per hectare. This clearly shows that only one grid cell gets assigned a value. Note in the above that there is only a single grid cell in the object comprising the "InclusionZoneGrid".

We can also show the minimal bounding grid that includes the whole circular plot plus the log, as we have done in previous examples. Figure 5 presents this graphically.

```
R> izgCSaw = izGrid(csaw, btr, wholeIZ=TRUE)
R> plot(izgCSaw, gridCenters=TRUE, estimate='Density',
+      showPlotCenter=TRUE, izCenterColor = 'white')
```

As with the other figures, the overall minimal bounding grid in Figure 5 has all grid cells other than the one at the center of the circular plot set to zero. Therefore, doing any subsequent map algebra will have no effect on the sampling surface using this enlarged grid. Showing this minimal bounding grid is more an effort to help illustrate the underlying concepts.

It is especially important to recognize that the center point of the circular plot does not necessarily lie at the center of any given grid cell. Depending upon the cell resolution used, this can make the placement of this respective cell look "off-center" with respect to the circlular plot's perimeter itself. It is only true that the center point falls somewhere within the inclusion zone grid cell, which could even be right on the edge. This is shown in Figure 5 where we use a white cross for the circular plot center, and is demonstrated below in terms of actual coordinates...

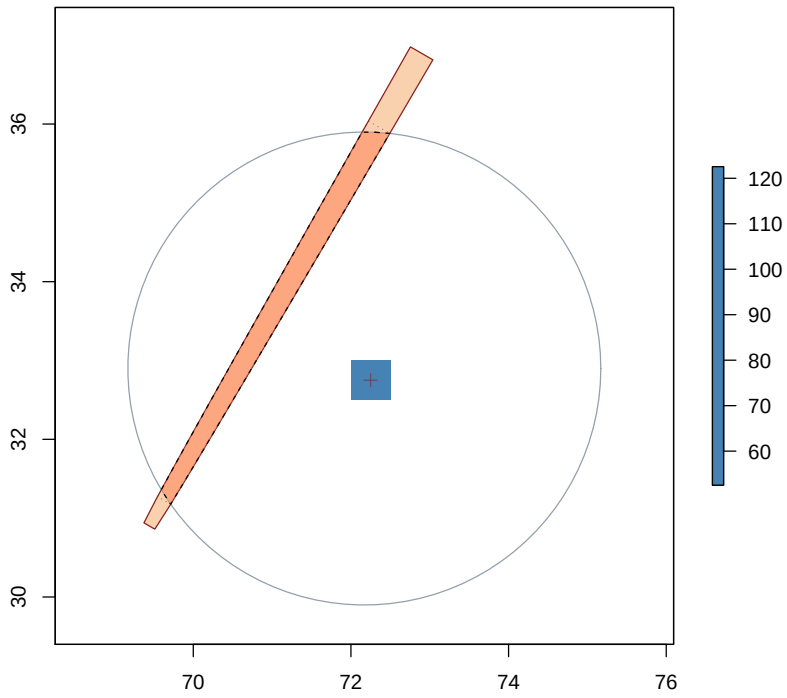

Figure 4: A "InclusionZoneGrid" object based on a "chainSawIZ" object showing that the inclusion zone is only a single point, and therefore is assigned to just one grid cell.

```
R> cpt = perimeter(csaw, whatSense='point') #circular plot centerpoint
R> cn = cellFromXY(izgCSaw@grid, cpt)       #cell number for plot center point
R> xy = xyFromCell(izgCSaw@grid, cn)        #cell center point
R> rbind(coordinates(cpt), xy)
```

```
             x        y
[1,] 72.170199 32.898489
[2,] 72.250000 32.750000
```

6.1 Snapping to the grid

As another example, suppose we wanted to develop a figure that shows essentially the same depiction as in Figure 5, but also including the overall sausage inclusion zone, and the plot center exactly

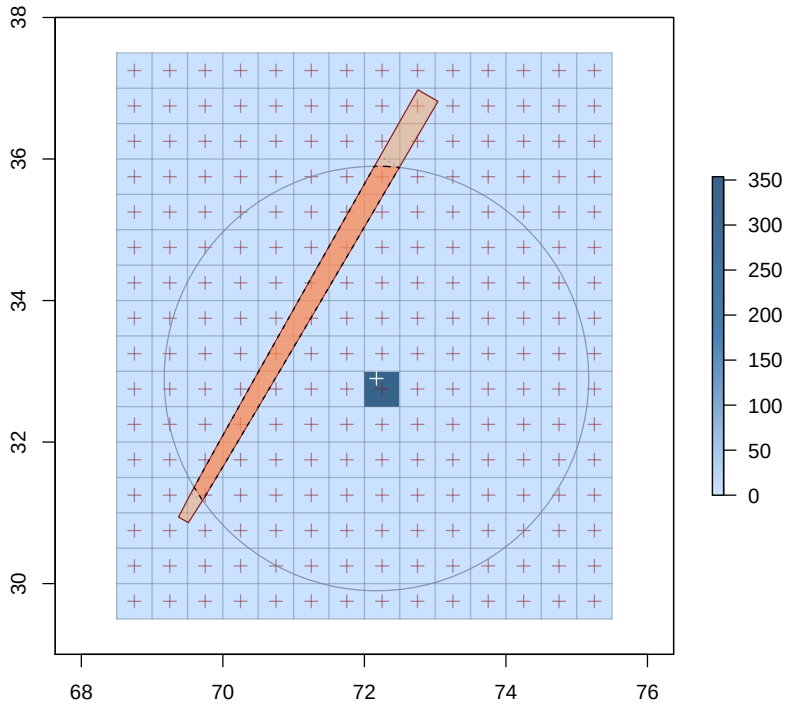

Figure 5: A "InclusionZoneGrid" object based on a "chainSawIZ" object showing the minimal bounding grid for the entire inclusion zone object; this also illustrates that the inclusion zone is only a single point, since only one grid cell is non-zero valued in terms of number of stems per unit area estimate.

aligned to a grid cell center. First we would make a "Tract" object that just holds the sausage inclusion zone object, the log, and the chainsaw inclusion zone object (complete with plot radius as in Figure 5). We can use the above steps to advantage to snap the plot centerpoint to the grid as follows...

```
R> xyExtent = c(x=10, y=10)
R> tra2 = Tract(xyExtent, cellSize=0.5)
R> dl = downLog(buttDiam=40, topDiam=15, logLen=6.5, logAngle=pi/4,
+               centerOffset=xyExtent/2 )
R> cn = cellFromXY(tra2, c(x=6.5, y=4.5))
R> (cpt = xyFromCell(tra2, cn)[,,drop=TRUE])  #coerce to vector from matrix
```

```
   x    y
6.75 4.25
```

```
R> izCS = chainSawIZ(dl, plotRadius = 2, plotCenter = cpt) #cell-based chainsaw iz
R> izgCS = izGrid(izCS, tra2)                  #and chainsaw iz grid object
R> hiz = heapIZ(izgCS, tra2)                   #heap it into the tract object
R> izSaus = sausageIZ(dl, plotRadius=2)        #overall sausage inclusion zone
```

We use the `heapIZ` method in the above to "heap" the "InclusionZoneGrid" object for the single grid cell corresponding to the chainsaw method at that point, onto the tract. The rest of the idea behind this should be fairly standard and is covered in more detail in the vignettes for the respective objects.

To plot this object, we need to basically build it up from scratch, the result is shown in Figure 6...

```
R> plot(hiz, axes=TRUE, gridLines=TRUE)
R> plot(dl, add=TRUE)
R> plot(izSaus, add=TRUE, izColor=NA, lty='dashed', izBorder='gray40')
R> plot(izCS, add=TRUE, izColor=NA, showPlotCenter=TRUE,
+      izCenterColor='white', ltyBolt='solid')
```

7 The "csFullInclusionZoneGrid" Class

Gove and Van Deusen (2011) describe a certain protocol for the chainsaw method that leads directly to the sausage method. They also show by simulation how the chainsaw method is biased for whole-log attributes, because the full inclusion zone for the log is the sausage zone under this particular protocol. To show this, every grid point within the sausage inclusion zone has to be estimated individually with its midpoint acting as the center of the circular plot, and then applying the chainsaw method to that plot. Again, this is repeated for every grid cell within the sausage inclusion zone.

This necessitates a new class, actually a subclass of "InclusionZoneGrid" with one extra slot, and some more validity checking, as well as a new constructor for the objects. The extra slot is just used to store the result of *each* "chainSawIZ" object applied to each of the internal inclusion zone grid cells.

The class is defined with the slots...

```
R> showClass('csFullInclusionZoneGrid')
```

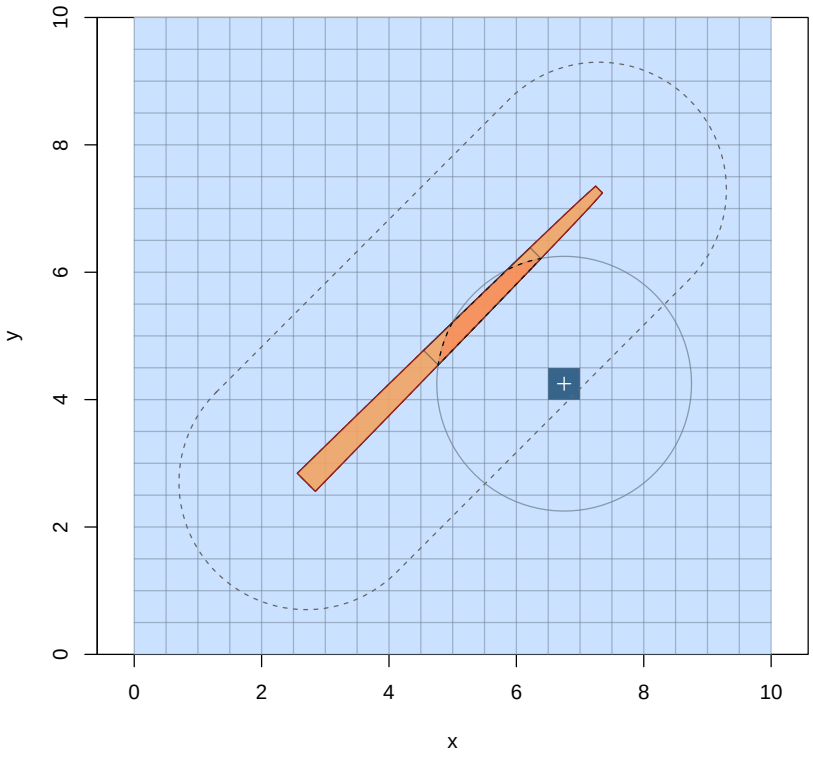

Figure 6: A built-up plot showing not only the "InclusionZoneGrid" object for one cell of the chainsaw method, but also the sausage object inclusion zone as would be determined by protocol 1 of Gove and Van Deusen (2011).

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

Slots:

Name:           chiz   description            iz          grid          data
Class:          list     character InclusionZone   RasterLayer    data.frame

Name:           bbox
Class:        matrix

Extends: "InclusionZoneGrid"
```

7.1 Class slots

- *chiz*: This is a list object containing `NA`s for cells outside the inclusion zone, but containing the full set of "InclusionZoneGrid" objects corresponding to each grid cell within the inclusion zone. As mentioned above, the grid cell center is used as the center point of the circular plot that defines the chainsaw intersection with the log.

The nice thing about the subclass extension for this new object is that only one slot was added; therefore, all of the functions that work on "InclusionZoneGrid" objects will also work on objects of this new class "csFullInclusionZoneGrid". These include the `print`, `show`, `summary`, and `plot` routines.

In addition, because each of the componets of the list in `chiz` is of class "InclusionZoneGrid" (or `NA`), we can apply any of the methods for that class on the individual slots. For example, we can plot them as in Figure 4, and look at how the chainsaw method works as we step from one cell to the next, showing the intersections of the circular plots with the log.

7.2 Object Construction and Plotting

Now, object construction takes quite a while because it has to compute the chainsaw intersections, etc., for each internal sausage grid cell. So here we use an existing object to demonstrate the idea. In the following, we show how to make an object of class "csFullInclusionZoneGrid" using its constructor, but do not evaluate it (we use the existing object instead); the steps leading to its creation are also shown...

```
R> btLog = sampleLogs(1, buttDiam=c(30,40), sampleRect=buffTr@bufferRect,
+                     logLen=c(4,6), topDiams=c(0, 0.5) )
R> btLog = downLogs(btLog)
R> btLog = btLog@logs$log.1
R> btLog.sa = sausageIZ(btLog, 3)
R> btLog.izgsa = izGrid(btLog.sa, buffTr)
R> btLog.izgFCS = izGridCSFull(btLog.izgsa, buffTr)
```

where `buffTr` is essentially the same as `btr` used in the previous examples.

```
R> btLog.izgFCS
```

```
Object of class: csFullInclusionZoneGrid
------------------------------------------------------------
```

```
Full chainSaw-sausage inclusion zone grid object
------------------------------------------------------------

InclusionZone class: sausageIZ
  units of measurement:  metric

Grid class: RasterLayer
Number of grid cells = 375
Cell dimensions: (nrows=15, ncol=25)
Grid cell values**...
  gridValues Freq
1          0  245
2       <NA>  130
**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
Min.      0.02701   353.7    2.397       1.758       0.2131    0.5888    0.2944
1st Qu.  27.42000   353.7  498.400     459.900     146.0000  597.7000  298.8000
Median   77.31000   353.7  926.600     929.600     295.9000 1685.0000  842.7000
Mean     85.24000   353.7  942.400     972.500     309.1000 1858.0000  929.2000
3rd Qu. 143.30000   353.7 1387.000    1475.000     469.5000 3124.0000 1562.0000
Max.    186.50000   353.7 1935.000    2120.000     673.7000 4065.0000 2033.0000


  Encapulating bounding box...
   min  max
x 61.0 73.5
y 61.5 69.0
```

One thing to note in particular, is that in the printed summary of the object above, the summary statistics vary here because the grid cells are composed of individual chainsaw estimates; this is not true for methods where the cells internal to the inclusion zone only take a single constant value (e.g., stand-up, sausage, point relascope, etc.).

And plotting the object is as usual...

```
R> plot(btLog.izgFCS, gridCenters=TRUE, showNeedle=TRUE)
```

As mentioned above, because the `chiz` slot contains a list of "InclusionZoneGrid" objects, for each grid cell within the inclusion zone, or `NA` for grid cells outside the zone, we can step through the internal cells one at a time, looking at summaries or plotting them. One way to do this would be the following...

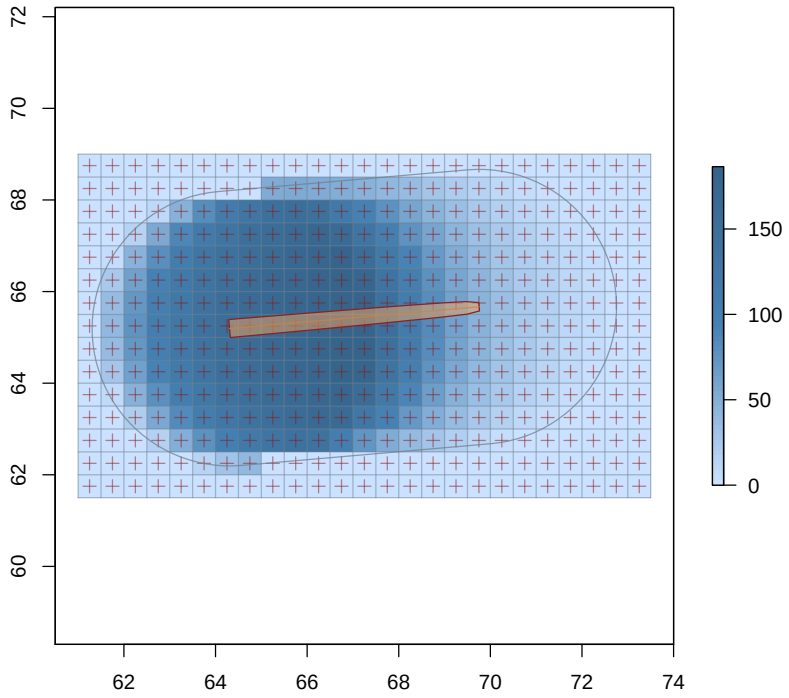

Figure 7: A "csFullInclusionZoneGrid" object based on a "sausageIZ" object.

```
R> cdx = ifelse(is.na(btLog.izgFCS@chiz), FALSE, TRUE)
R> csl = btLog.izgFCS@chiz[cdx]
R> length(csl)
```

```
[1] 245
```

```
R> sapply(csl[1:4], class)
```

```
          izgCS.34            izgCS.35            izgCS.36            izgCS.37
"InclusionZoneGrid" "InclusionZoneGrid" "InclusionZoneGrid" "InclusionZoneGrid"
```

Again, we could then plot the slivers we are interested in, stepping through to see how the chainsaw method slices the log up for each individual grid cell.

8 Example: The "pointRelascopeIZ" Class

Here we present an example for the point relascope sampling method (Gove et al. 1999, Gove et al. 2001)...

```
R> (angle = .StemEnv$rad2Deg(2*atan(1/2)))
```

```
[1] 53.130102
```

```
R> prs.as = pointRelascope(angle, units='metric')
R> prs.iz = pointRelascopeIZ(dlogs@logs$log.1, prs=prs.as)
R> (izgPRS = izGrid(prs.iz, btr))
```

```
Object of class: InclusionZoneGrid
------------------------------------------------------------
pointRelascopeIZ inclusion zone grid object
------------------------------------------------------------

InclusionZone class: pointRelascopeIZ
  units of measurement:  metric

Grid class: RasterLayer
Number of grid cells = 672
Cell dimensions: (nrows=24, ncol=28)
Grid cell values**...
  gridValues Freq
1          0  400
2       <NA>  272
**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
Min.     39.67   99.21  686.5       578.2          184   797.4  398.7
1st Qu.  39.67   99.21  686.5       578.2          184   797.4  398.7
Median   39.67   99.21  686.5       578.2          184   797.4  398.7
Mean     39.67   99.21  686.5       578.2          184   797.4  398.7
3rd Qu.  39.67   99.21  686.5       578.2          184   797.4  398.7
Max.     39.67   99.21  686.5       578.2          184   797.4  398.7
```

```
  Encapulating bounding box...
  min max
x  64  78
y  28  40
```

```
R> plot(izgPRS)
```

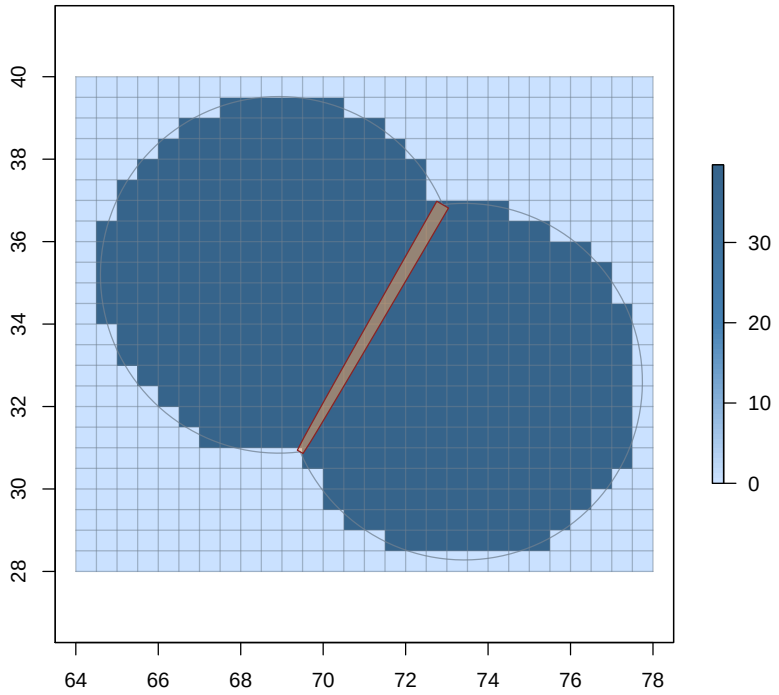

Figure 8: A "InclusionZoneGrid" object based on a "pointRelascopeIZ" object showing that the stem and inclusion zone are covered by the bounding grid.

9 Example: The "perpendicularDistanceIZ" Class

Here we present an example for the perpendicular distance sampling method (Williams and Gove 2003, Williams et al. 2005, Ducey et al. 2008), this example happens to be for volume estimation. Please note that, because we know the log's true volume from simulation, we can in fact estimate

all the other attributes for the log. Normally, however, we do not know the true volume from field measurements, so we would only be able to estimate volume under "canonical" PDS, and would use the "omnibus" variant PDS given in the next section to estimate these other quantities...

```
R> pdsmet = perpendicularDistance(kpds=50, units='metric')
R> iz.pdsv = perpendicularDistanceIZ(dlogs@logs$log.1, pdsmet)
R> (izgPDS = izGrid(iz.pdsv, btr))
```

```
Object of class: InclusionZoneGrid
------------------------------------------------------------
perpendicularDistanceIZ inclusion zone grid object
------------------------------------------------------------

InclusionZone class: perpendicularDistanceIZ
  units of measurement:  metric

Grid class: RasterLayer
Number of grid cells = 342
Cell dimensions: (nrows=19, ncol=18)
Grid cell values**...
  gridValues Freq
1          0  161
2       <NA>  181
**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
Min.       100   250.1   1731        1457        463.9    2010   1005
1st Qu.    100   250.1   1731        1457        463.9    2010   1005
Median     100   250.1   1731        1457        463.9    2010   1005
Mean       100   250.1   1731        1457        463.9    2010   1005
3rd Qu.    100   250.1   1731        1457        463.9    2010   1005
Max.       100   250.1   1731        1457        463.9    2010   1005


  Encapulating bounding box...
  min  max
x  68 77.0
y  30 39.5
```

```
R> plot(izgPDS)
```

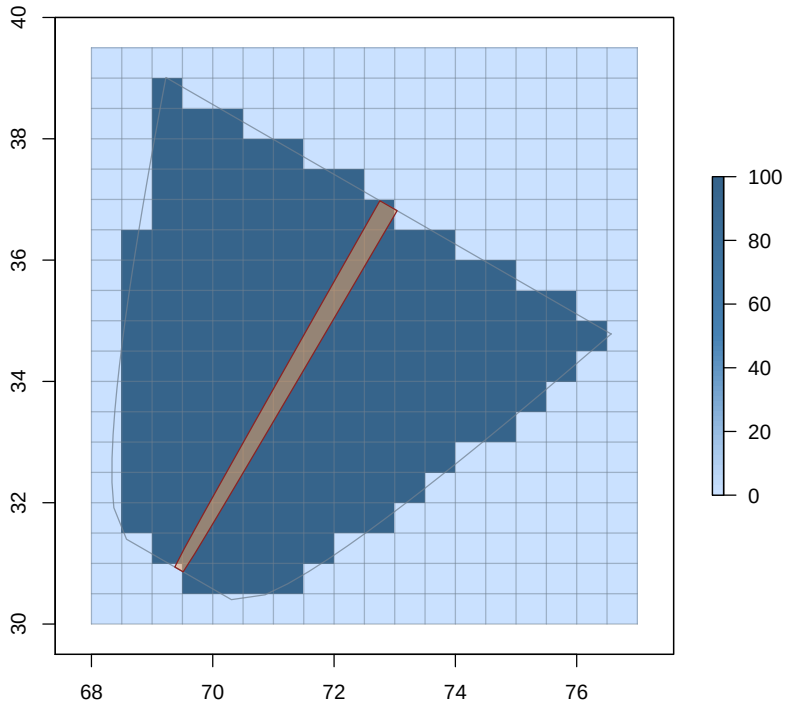

Figure 9: A "InclusionZoneGrid" object based on a "perpendicularDistanceIZ" object showing that the stem and inclusion zone are covered by the bounding grid.

As noted in *The InclusionZone Class* vignette, the inclusion zone for PDS is developed directly from the dataframe in the `taper` slot of the "downLog" object. It can be seen here that if that taper approximation is poor because it uses too few points, it has the potential to exclude grid cells that would otherwise normally be included. This in turn could lead to unexpected "simulation bias" in the sampling surface result (the same thing can happen from using too large a grid cell). Thus, it is fairly important to use a good number of log sections in the taper dataframe to avoid this possibility.

10 Example: The "omnibusPDSIZ" Class

An extension to "canonical" PDS presented in the previous section and given by Ducey et al. (2008), allows one to estimate any attribute on the log and is sometimes referred to as "omnibus" PDS. Because this method uses stem measurements perpendicular to the sample point to form the

estimates, it has varying height surface in each case, except that for the variable we use for the PPS selection of the log[1]...

```
R> iz.opds = omnibusPDSIZ(dlogs@logs$log.1, pdsmet)
R> (izgOPDS = izGrid(iz.opds, btr))
```

```
Object of class: InclusionZoneGrid
------------------------------------------------------------
omnibusPDSIZ inclusion zone grid object
------------------------------------------------------------

InclusionZone class: omnibusPDSIZ
  units of measurement:  metric

Grid class: RasterLayer
Number of grid cells = 342
Cell dimensions: (nrows=19, ncol=18)
Grid cell values**...
  gridValues Freq
1          0  161
2       <NA>  181
**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
Min.       100   170.6   1181        1218        387.8    2010   1005
1st Qu.    100   189.7   1313        1284        408.8    2010   1005
Median     100   220.0   1522        1383        440.3    2010   1005
Mean       100   248.1   1717        1452        462.1    2010   1005
3rd Qu.    100   276.5   1913        1551        493.5    2010   1005
Max.       100   663.9   4594        2403        764.8    2010   1005


  Encapulating bounding box...
  min  max
x  68 77.0
y  30 39.5
```

```
R> plot(izgOPDS, estimate='coverageArea')
```

[1]Note that if log selection is with PP to volume, then since both biomass and carbon are simply scaled versions of volume, their surfaces will also be constant.

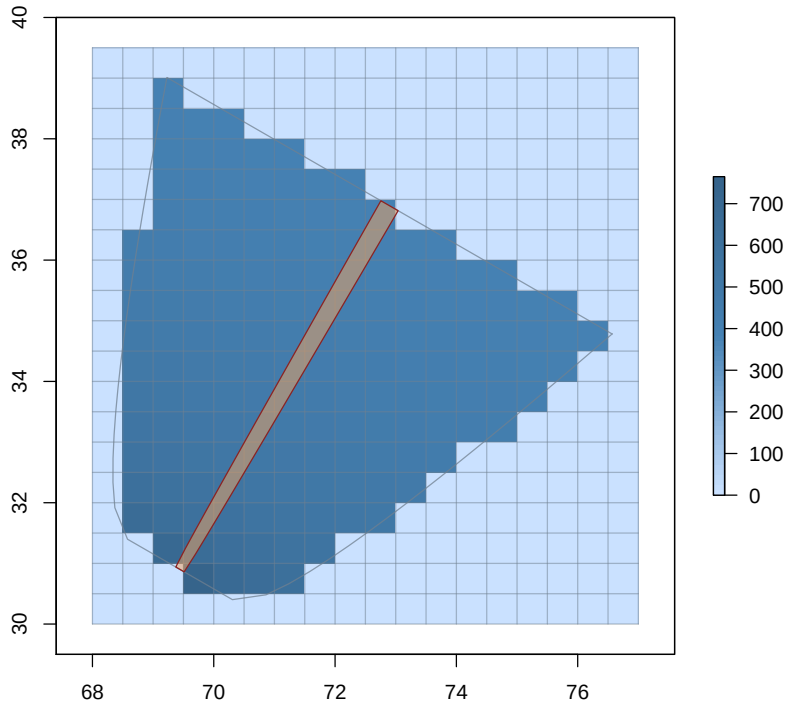

Figure 10: A "InclusionZoneGrid" object based on a "omnibusPDSIZ" object showing the variable height sampling surface for coverage area.

11 Example: The "distanceLimitedIZ" Class

Here we interject a method that is not a PDS variant, but we introduce it now because it is used in the distance limited PDS method presented below. There are two protocols for distance limited sampling: *(i)* a standard/canonical (DLS) with constant surface heights for all attributes, and *(ii)* a crude Monte Carlo based protocol (DLMCS) with varying surface height for most, but not all attributes. The first protocol is discussed here, while DLMCS is discussed in the next section. The protocols are discussed in detail in Gove et al. (2012). The inclusion zone is covered in *The "InclusionZone" Class* vignette.

```
R> dlsMet = distanceLimited(3, units='metric')
R> iz.dls = distanceLimitedIZ(dlogs@logs$log.1, dls=dlsMet)
R> (izgDLS = izGrid(iz.dls, btr))
```

```
Object of class: InclusionZoneGrid
------------------------------------------------------------
distanceLimitedIZ inclusion zone grid object
------------------------------------------------------------

InclusionZone class: distanceLimitedIZ
  units of measurement:  metric

Grid class: RasterLayer
Number of grid cells = 380
Cell dimensions: (nrows=20, ncol=19)
Grid cell values**...
  gridValues Freq
1          0  167
2       <NA>  213
**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
Min.     96.31   240.8   1667        1404        446.7    1936  967.9
1st Qu.  96.31   240.8   1667        1404        446.7    1936  967.9
Median   96.31   240.8   1667        1404        446.7    1936  967.9
Mean     96.31   240.8   1667        1404        446.7    1936  967.9
3rd Qu.  96.31   240.8   1667        1404        446.7    1936  967.9
Max.     96.31   240.8   1667        1404        446.7    1936  967.9


  Encapulating bounding box...
   min max
x 66.5  76
y 29.0  39
```

```
R> plot(izgDLS, estimate='biomass')
```

The surfaces for log length and log density will be exactly the same under this protocol as the Monte Carlo variant as can be verified with the results in the following section. All other attribute surfaces will differ between the two protocols, as is illustrated for biomass in Figure 11.

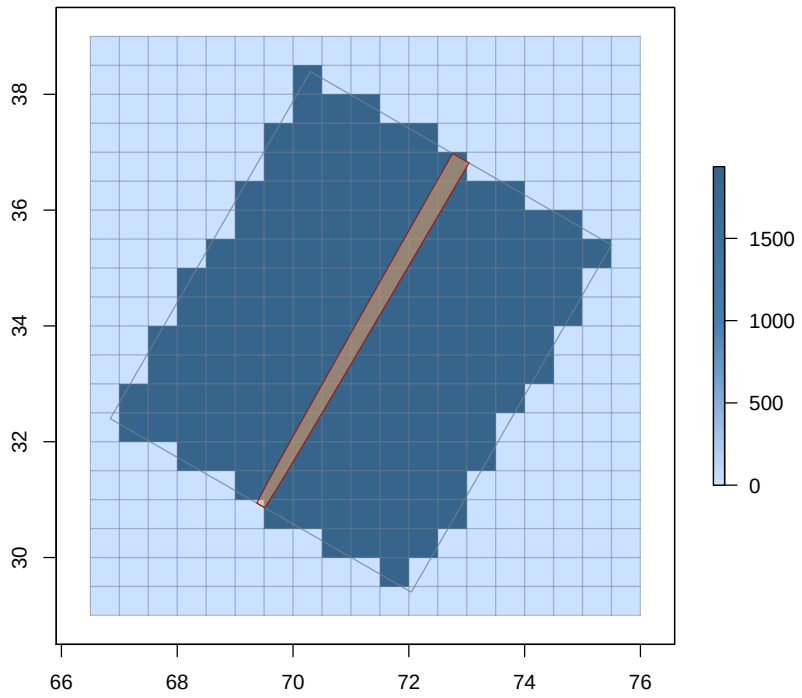

Figure 11: A "InclusionZoneGrid" object based on a "distanceLimitedIZ" object showing the variable height sampling surface for biomass.

12 Example: The "distanceLimitedMCIZ" Class

The Monte Carlo protocol for distance limited sampling is illustrated here. The inclusion zone is covered in *The "InclusionZone" Class* vignette, and the surface will be constant only for log length and density and be exactly the same as for DLS. An example follows...

```
R> dlsMet = distanceLimited(3, units='metric')
R> iz.dlmcs = distanceLimitedMCIZ(dlogs@logs$log.1, dls=dlsMet)
R> (izgDLMCS = izGrid(iz.dlmcs, btr))
```

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

```
------------------------------------------------------------

InclusionZone class: distanceLimitedMCIZ
  units of measurement:  metric

Grid class: RasterLayer
Number of grid cells = 380
Cell dimensions: (nrows=20, ncol=19)
Grid cell values**...
  gridValues Freq
1          0  167
2       <NA>  213
**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
Min.     36.28   240.8   1667       871.7        277.5   729.2  364.6
1st Qu.  73.92   240.8   1667      1244.0        396.1  1486.0  742.9
Median   98.93   240.8   1667      1439.0        458.2  1989.0  994.3
Mean     96.60   240.8   1667      1406.0        447.4  1942.0  970.8
3rd Qu. 121.00   240.8   1667      1592.0        506.7  2432.0 1216.0
Max.    141.00   240.8   1667      1718.0        547.0  2834.0 1417.0


  Encapulating bounding box...
   min max
x 66.5  76
y 29.0  39


R> plot(izgDLMCS, estimate='biomass')
```

Note from the summary output that the surface is indeed variable for all attributes other than density and Length. This is illustrated for biomass in Figure 12.

13 Example: The "distanceLimitedPDSIZ" Class

This class, as explained in *The "InclusionZone" Class* vignette, is a hybrid sampling method that restricts the maximum width of the PDS inclusion zone, effectively truncating the search distance for logs. The "hybrid" effect comes from the fact that the inclusion zone can have one of three variations: *(i)* the inclusion zone is all PDS, *(ii)* the inclusion zone is all DLS, or *(iii)* it is a

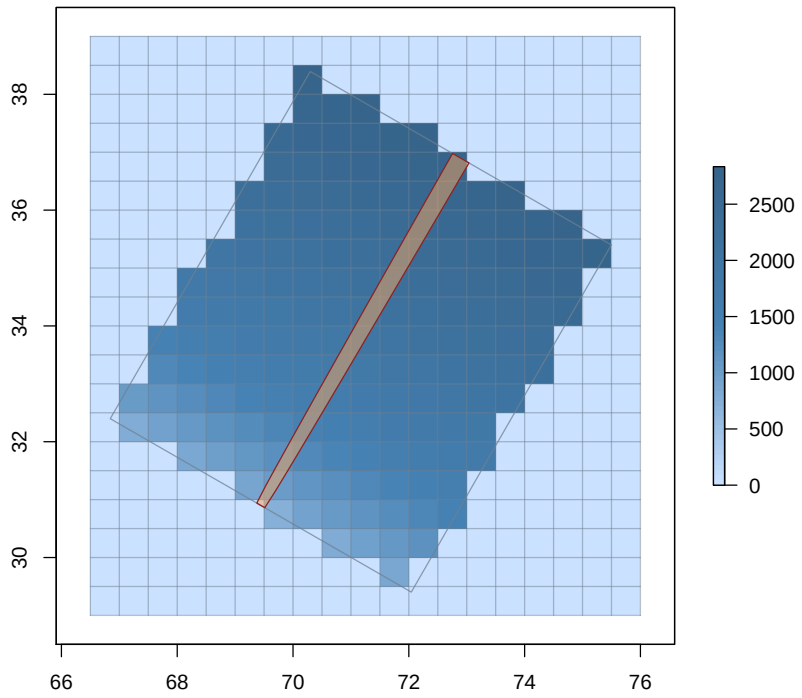

Figure 12: A "InclusionZoneGrid" object based on a "distanceLimitedMCIZ" object showing the variable height sampling surface for biomass.

combination of the two. This flexibility can make things a bit messy, however. We again define two different protocols within this method. First, "canonical" DLPDS uses DLS for the truncated portion of the inclusion zone and canonical PDS for the section that is treated as a normal PDS sample.[2] In the second protocol, we substitute omnibus PDS for any section that is to be sampled with PDS, while DLMCS is used for the distance limited portion, and refer to this as "omnibus" DLPDS. Note that the difference is entirely based on the protocols for both the PDS and distance limited components (if any) for each log. Both components of the inclusion zone can, therefore, be either constant or variable depending on the PPS selection strategy (volume, surface area or coverage area) and the particular attribute we are estimating (refer to the last few sections for more information).

[2]Of course, in field applications, this method is limited to sampling only for the PPS selection variable in the PDS component for the same reasons as described above.

```
R> iz.dlpds = distanceLimitedPDSIZ(dlogs@logs$log.1, pds=pdsmet, dls=dlsMet)
R> (izgDLPDS = izGrid(iz.dlpds, btr))


Object of class: InclusionZoneGrid
------------------------------------------------------------
a distance limited PDSIZ inclusion zone grid object
------------------------------------------------------------

InclusionZone class: distanceLimitedPDSIZ
  units of measurement:  metric

Grid class: RasterLayer
Number of grid cells = 288
Cell dimensions: (nrows=18, ncol=16)
Grid cell values**...
  gridValues Freq
1          0  143
2       <NA>  145
**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
Min.     100.0   248.7   1667        1590        506.1    2010   1005
1st Qu.  100.0   248.7   1667        1590        506.1    2010   1005
Median   121.0   248.7   1667        1590        506.1    2432   1216
Mean     111.9   279.7   1935        1630        519.0    2249   1124
3rd Qu.  121.0   320.2   2286        1683        535.8    2432   1216
Max.     121.0   320.2   2286        1683        535.8    2432   1216


  Encapulating bounding box...
  min max
x  68  76
y  30  39


R> plot(izgDLPDS, estimate='biomass', showPDSPart=TRUE)
```

Figure 13 shows[3] the two inclusion zones in the hybrid region. Since biomass is a scaled version of volume (the PPS selection variable), the PDS component surface is constant as in Figure 9. The DLS component is also constant, just as the surface generated in Figure 11. Now we are able to see

[3]The log is randomly generated with each run of this document, and so differs with each creation, but the parameters are chosen such that it should show the two zones of case *(iii)*.

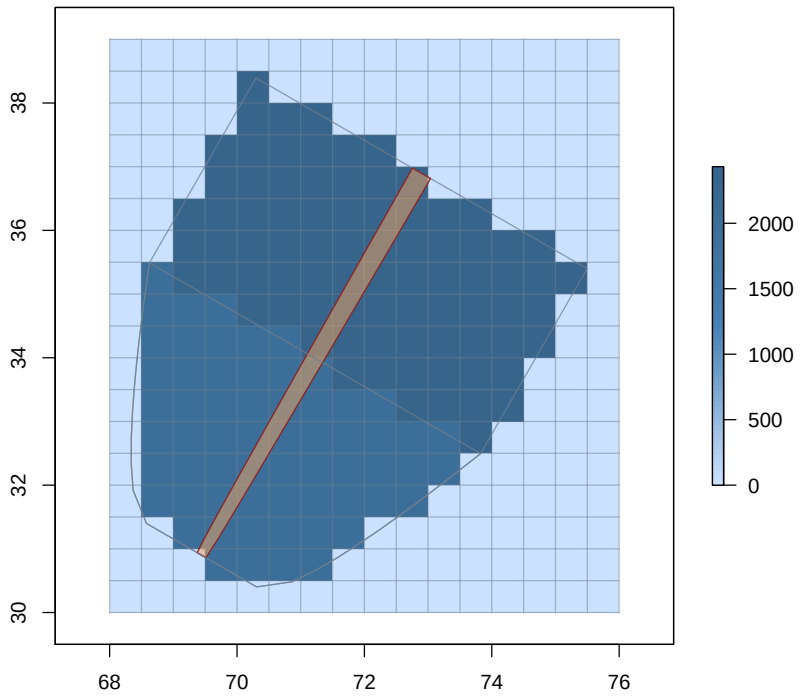

Figure 13: A "InclusionZoneGrid" object based on a "distanceLimitedPDSIZ" object showing the variable "stair step" height sampling surface for biomass.

why this is a hybrid scheme more clearly. Note from the object summary that the combination of the two types of inclusion zone objects makes all of the attributes variable in case *(iii)*, since both PDS and DLS will always produce constant height surfaces that will, in general, not be the same height, resulting in the entire surface resembling a "step" function.

14 Example: The "omnibusDLPDSIZ" Class

Omnibus PDS is the final PDS-based method. Again, this is similar to canonical DLPDS with the exception that omnibus PDS is employed within the PDS component of the inclusion zone, and DLMCS is used in the distance limited portion, if any. This method will be more appropriate for most field applications since one is able to estimate any attribute shown in Ducey et al. (2008). The resulting surface will be variable as described in the sections for the individual component methods above.

```
R> iz.odlpds = omnibusDLPDSIZ(dlogs@logs$log.1, pds=pdsmet, dls=dlsMet)
R> (izgODLPDS = izGrid(iz.odlpds, btr))


Object of class: InclusionZoneGrid
------------------------------------------------------------
a distance limited PDSIZ inclusion zone grid object
------------------------------------------------------------

InclusionZone class: omnibusDLPDSIZ
  units of measurement:  metric

Grid class: RasterLayer
Number of grid cells = 288
Cell dimensions: (nrows=18, ncol=16)
Grid cell values**...
  gridValues Freq
1          0  143
2       <NA>  145
**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
Min.     100.0   233.9   1667        1449        461.1    2010   1005
1st Qu.  100.0   248.7   1667        1522        484.6    2010   1005
Median   105.5   248.7   1667        1600        509.2    2120   1060
Mean     112.1   278.6   1927        1628        518.3    2253   1126
3rd Qu.  124.0   277.5   1981        1683        535.6    2493   1247
Max.     141.0   676.7   4831        2464        784.3    2834   1417


  Encapulating bounding box...
  min max
x  68  76
y  30  39


R> plot(izgODLPDS, estimate='surfaceArea', showPDSPart=TRUE)
```

It may be difficult to see in Figure 14, but the surface is actually slightly convex from butt to tip, because it varies in the reverse sense for each of the two components; i.e., larger near the butt for DLMCS, but larger near the tip for omnibus PDS.

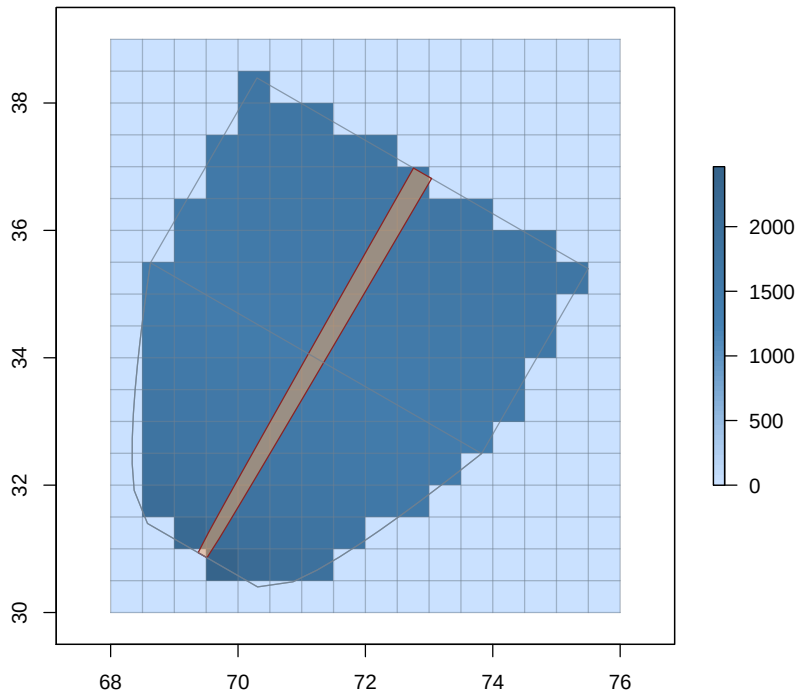

Figure 14: A "InclusionZoneGrid" object based on a "omnibusDLPDSIZ" object showing the variable height sampling surface for surface area.

15 Using `plot3D`

The `plot3D` generic was extended to handle objects of class "InclusionZoneGrid". Its use is simple, just remember to use the `estimate` argument to specify the desired surface attribute to be rendered...

```
R> plot3D(izgODLPDS, estimate='surfaceArea')
```

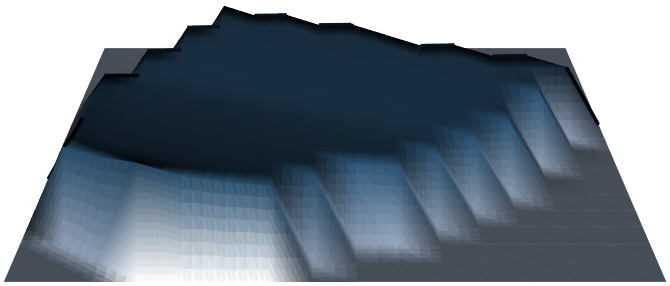

Figure 15: Representation of the sampling surface (via an "InclusionZoneGrid" object) for a single log sampled with "omnibusDLPDS".

References

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. 18, 20, 28

J. H. Gove and P. C. Van Deusen. On fixed-area plot sampling for downed coarse woody debris. *Forestry*, 84(2):109–117, 2011. 7, 12, 13

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. J. Ducey, A. Ringvall, and G. Ståhl. Point relascope sampling: a new way to assess down coarse woody debris. *Journal of Forestry*, 4:4–11, 2001. 17

J. H. Gove, M. J. Ducey, and H. T. Valentine. A distance limited method for sampling downed coarse woody debris. *Journal of Applied Ecology*, 2012. (In preparation). 22

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. 18

M. S. Williams, M. J. Ducey, and J. H. Gove. Assessing surface area of coarse woody debris with line intersect and perpendicular distance sampling. *Canadian Journal of Forest Research*, 35: 949–960, 2005. 18