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

Friday 17th December, 2010
5:45pm

Contents

1 Introduction

When we think about building a sampling surface piece by piece from the inclusion zones of individual "Stem" objects, we assign the attribute value to all the grid cells 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 for each "Stem" object in the population. At this point there are no inherited methods, it just works for all subclasses of "InclusionZone". 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.

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

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, so we show several illustrations. In addition, we also treat the somewhat odd, but informative "full chainsaw" method where the entire sausage-based inclusion zone is filled with chainsaw estimates.

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.

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> cd = c(x=200, y=200)
R> tra = Tract(cellSize = 0.5,
+              cellDims = cd,
+              cellCenter = c(0.25, 0.25),
+              data = runif(prod(cd)),
+              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> tra = setValues(tra, rep(0, ncell(tra)))
R> btr = bufferedTract(10, tra)
R> dlogs = sampleLogs(1, buttDiam=c(30,40), sampleRect=btr@bufferRect,
+                     logLen=c(6,10), topDiams=c(0,0.5) )
```

```
R> dlogs = downLogs(dlogs)
R> sup = standUpIZ(dlogs@logs$log.1, 3)
R> izgSU = izGrid(sup, btr)
```

Here we have created a 200 by 200 cell raster grid 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 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)
```

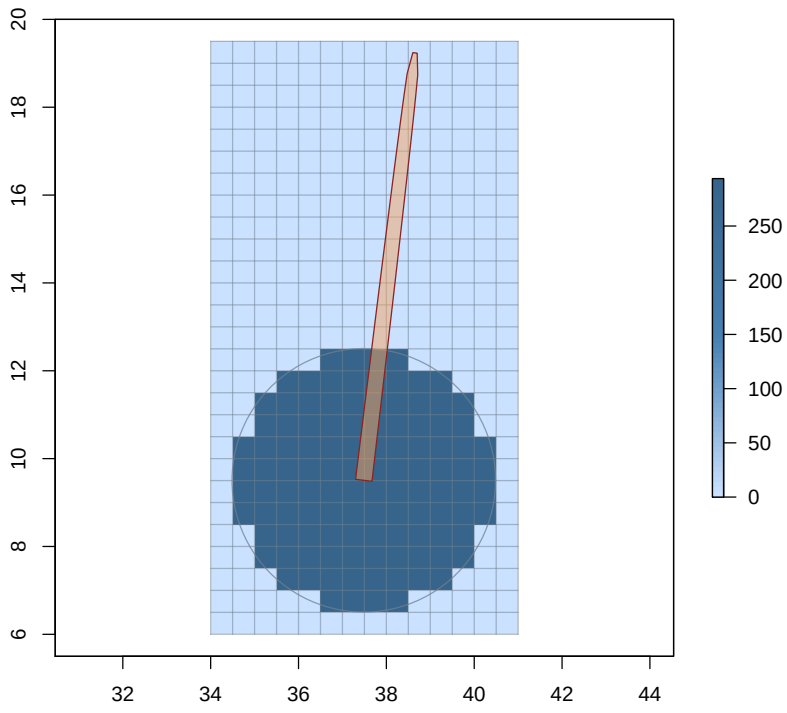

Figure 1: An "InclusionZoneGrid" object based on a "standUpIZ" object.

fig:suIZG

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)
```

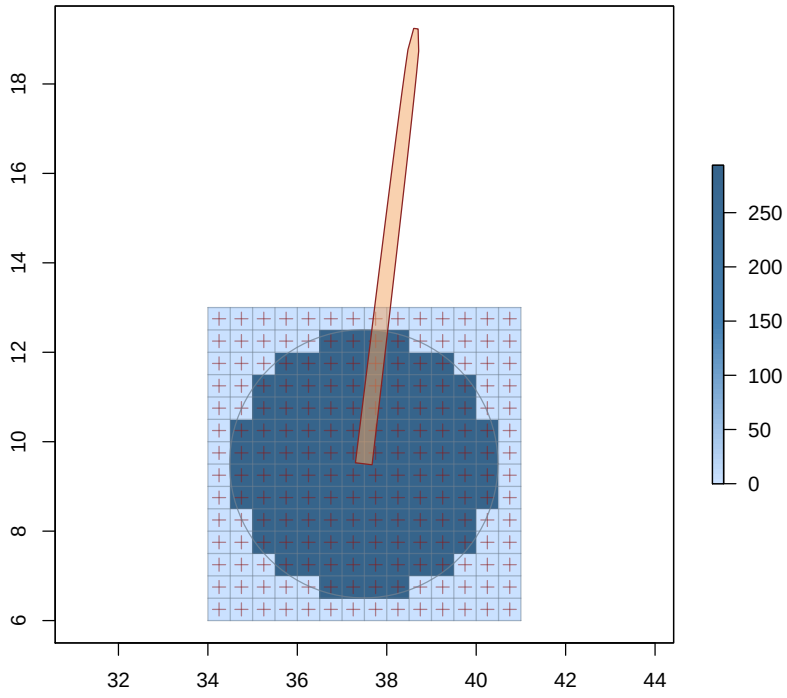

Figure 2: A "InclusionZoneGrid" object based on a "standUpIZ" object where only the actual inclusion zone is covered by the bounding grid, and grid cell centers are plotted as an option.

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...
   - Use the `polygonsToRaster` 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)
```

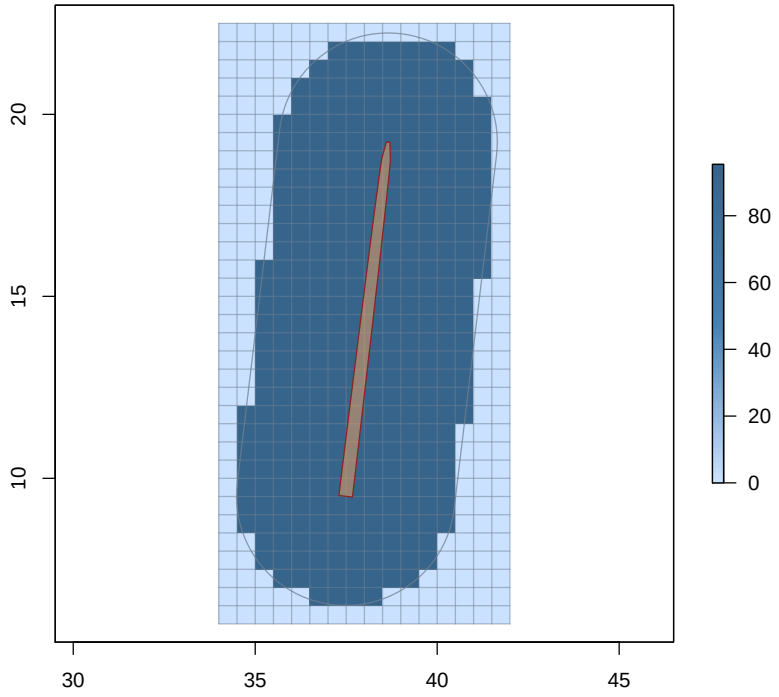

Figure 3: A "InclusionZoneGrid" object based on a "sausageIZ" object showing that the stem and inclusion zone are covered by the bounding grid.

fig:sausIZG

fig:sausIZG

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


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

InclusionZone class: chainSawIZ
  units of measurement:  metric

Tract class: RasterLayer
Grid cell values*...
  gridValues Freq
1          0    1

Number of grid cells = 1
Cell dimensions: (nrows=1, ncol=1)
Per unit area estimates in data slot...
        cubicVolume Density
Min.         167.85  353.68
1st Qu.      167.85  353.68
Median       167.85  353.68
Mean         167.85  353.68
3rd Qu.      167.85  353.68
Max.         167.85  353.68


  Encapulating bounding box...
         min       max
x 36.0704054 42.068895
y  9.4840923 19.242683
```

```
*Note: data slot values get swapped with zero-valued grid cells as necessary.
```

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

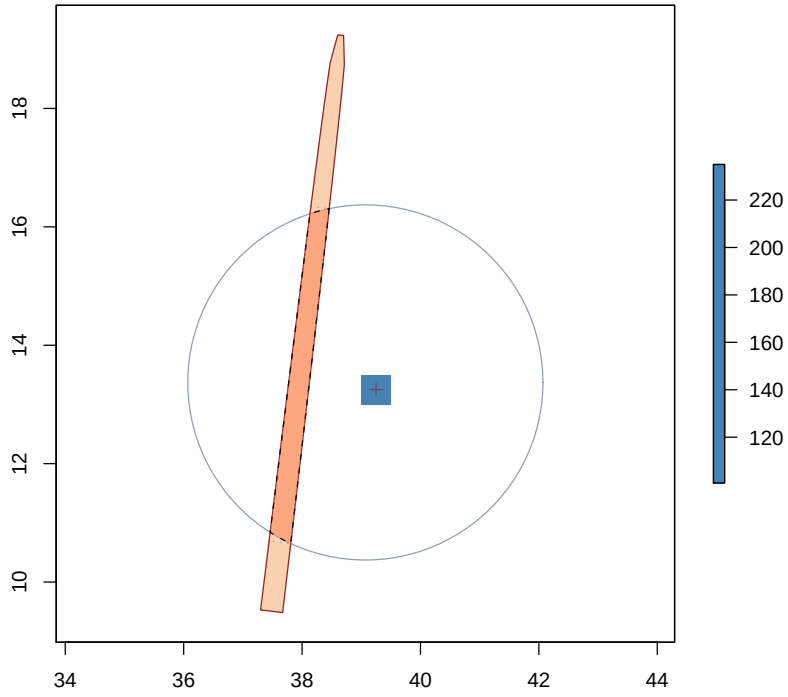

Figure 4: A "InclusionZoneGrid" object based on a "chainSawIZ" object showing that the inclusion zone is only a single point, and therefore is assigned to just one grid cell. fig:csawIZG

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". Again, this is the strange component with respect to the so-called chainsaw method.

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')
```

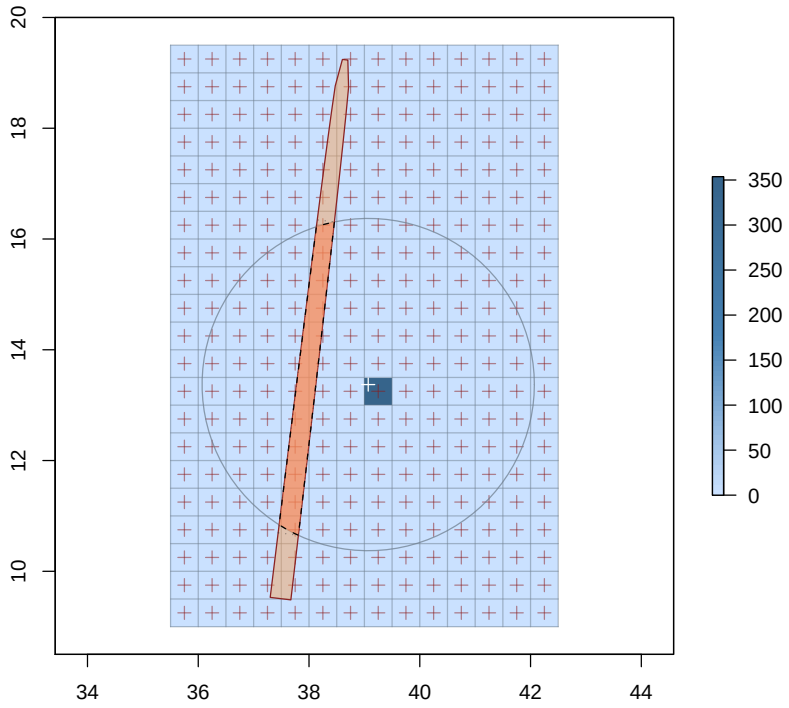

Figure 5: A "InclusionZoneGrid" object based on a "chainSawIZ" object showing the minimal bounding grid for the entire inclusion zone object; this also illustrates that the inclusion zone is only a single point, since only one grid cell is non-zero valued in terms of number of stems per unit area estimate.

fig:csawIZM

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.

fig:csawIZMBG

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...

fig:csawIZMBG

```
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,] 39.068895 13.371621
[2,] 39.250000 13.250000
```

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 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)
```

```
Grid Topology...
                      x     y
cellcentre.offset  0.25  0.25
cellsize           0.50  0.50
cells.dim         20.00 20.00
```

```
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...

```
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')
```

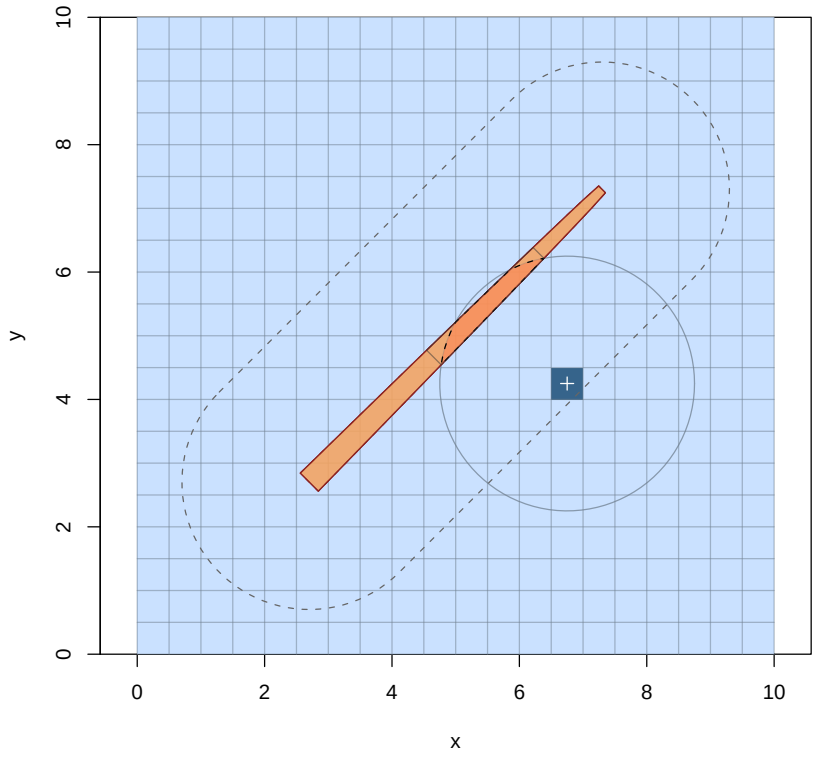

Figure 6: A built-up plot showing not only the "InclusionZoneGrid" object for one cell of the chainsaw method, but also the sausage object inclusion zone as would be determined by protocol 1 of Gove and Van Deusen (2011).

gove&vanDeusen:2011

fig:csawP1s

7 The "csFullInclusionZoneGrid" Class

gove&vanDeusen:2011
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')
```

```
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

Tract class: RasterLayer
Grid cell values*...
  gridValues Freq
1          0  246
```

```
2       <NA>  129

Number of grid cells = 375
Cell dimensions: (nrows=25, ncol=15)
Per unit area estimates in data slot...
        cubicVolume Density
Min.        0.23506  353.68
1st Qu.    25.36000  353.68
Median     76.07000  353.68
Mean       82.98000  353.68
3rd Qu.   141.06000  353.68
Max.      182.78000  353.68


  Encapulating bounding box...
   min  max
x 76.5 84.0
y 23.0 35.5

*Note: data slot values get swapped with zero-valued grid cells as necessary.
```

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 the other methods where the cells internal to the inclusion zone only take a single constant value (refer to the other methods above).

And plotting the object is as usual...

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

As mentioned above, because the `chiz` 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...

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

```
[1] 246
```

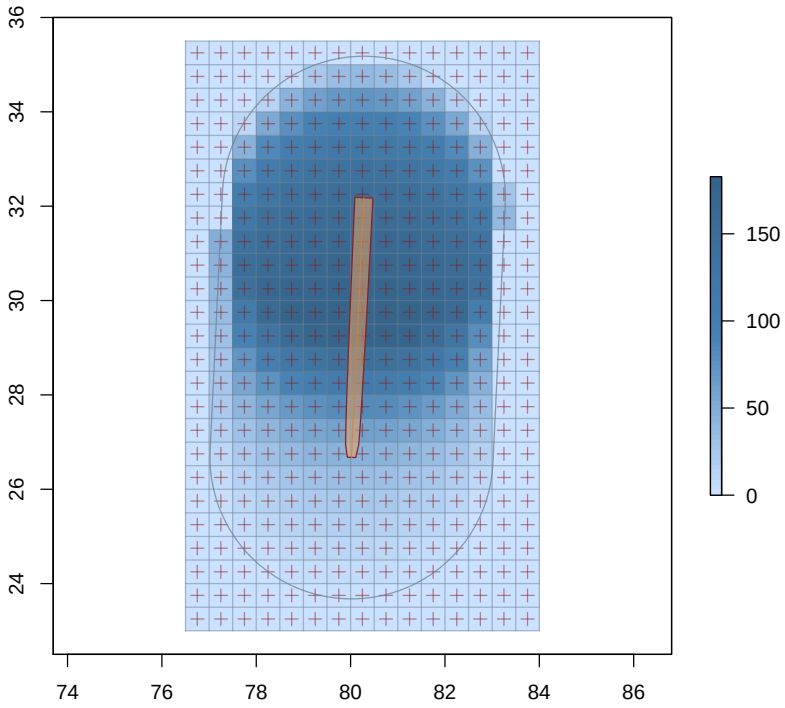

Figure 7: A "csFullInclusionZoneGrid" object based on a "sausageIZ" object. fig:csawFIZ

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

```
          izgCS.20           izgCS.21           izgCS.22           izgCS.23
"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.

References

aDeusen:2011 J. H. Gove and P. C. Van Deusen. On fixed-area plot sampling for downed coarse woody debris. *Forestry*, 2011. Submitted August 2010. 8, 12, 13