# The "InclusionZone" 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 December, 2010
11:19am

## Contents



## 1 Introduction

Inclusion zones in general are the area surrounding an object in which the sample point (e.g., the fixed-radius plot center) can land and sample the object. Therefore, they have a clearly defined area and perimeter. In addition, for PPS methods, they depend on some attribute of the object, such as length in the case of a down log. This class and its subclasses brings together objects from the "ArealSampling" class and the "Stem" class as we define the more useful subclasses corresponding to actual sampling methods for standing trees or down logs.

The "InclusionZone" object that will be defined through its subclasses will eventually instantiate an object that therefore contains both the inclusion zone and the stem itself. These components

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

will be stored in separate slots in the object, so they can be accessed individually as needed, but for plotting and summary purposes, the object will be treated as the union of the two component objects that define the inclusion zone and stem.

Figure 1 presents an overview of the class hierarchy arrangement. Round-cornered boxes represent virtual classes, while the "standingTreeIZ" class does not exist yet. A container class ("downLogIZs") for "downLogIZ" objects is also shown, it will handle collections of objects of any of the subclasses.

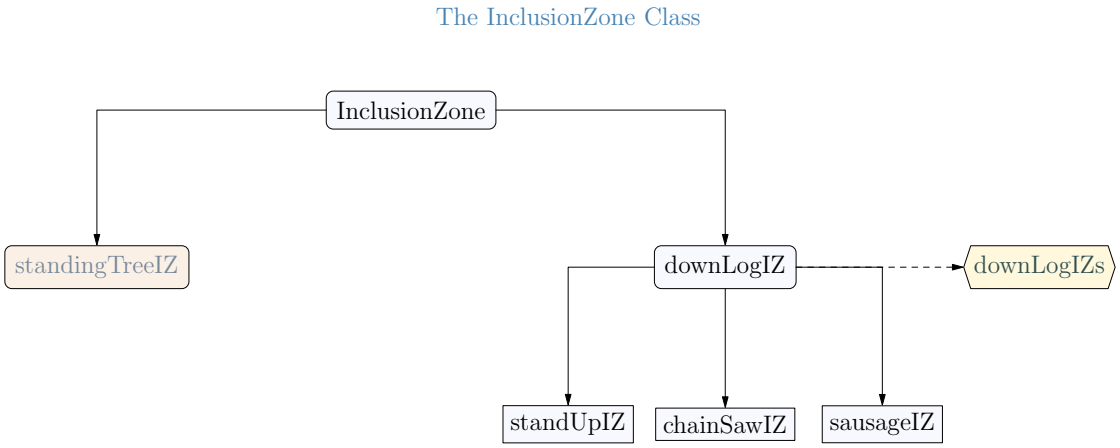


Figure 1: An overview of the "InclusionZone" class. fig:IZ

## 2 The "InclusionZone" Class

This is a virtual class which simply defines the main attributes of the subclasses. Note that this class is extremely basic, and it will have two direct subclasses, much like the "Stem" class, for standing tree- and down log-based sampling methods.

The base class is defined with the slots...

```
R> showClass('InclusionZone')
```

```
Virtual Class "InclusionZone" [package "sampSurf"]

Slots:

Name:   description        units         bbox      spUnits    puaBlowup
Class:    character    character       matrix          CRS      numeric
```

```
Name:  puaEstimates    userExtra
Class:         list          ANY

Known Subclasses:
Class "downLogIZ", directly
Class "standUpIZ", by class "downLogIZ", distance 2
Class "chainSawIZ", by class "downLogIZ", distance 2
Class "sausageIZ", by class "downLogIZ", distance 2
```

### 2.1 Class slots

- *description*: Some descriptive text about this class.
- *units*: A character string specifying the units of measure. Legal values are "English" and "metric."
- *bbox*: The bounding box enclosing the overall object and its inclusion zone. Since the inclusion zone may not include the entire object (as in the "standup" method for down logs), we require it here. Most of the time, it is actually redundant as the "SpatialPolygons" slot containing the inclusion zone in the subclasses will have this same information.
- *spUnits*: A valid string of class "CRS" denoting the spatial units coordinate system ("?CRS" for more information) as in package `sp`.
- *puaBlowup*: The per unit area blowup or expansion factor, based on an acre or hectare depending upon the units of measure.
- *puaEstimates*: A list of estimates for per unit area quantities associated with the "Stem" component of the object.
- *userExtra*: This can be anything else the user wants to associate with the object. Normally, it might be in the form of a `list` object, but can be anything. The user has complete control over this, it will not be used in any of the methods applied to the class, it is there for extra information storage as desired.

## 3 The "downLogIZ" Class

This is another small definitional step. It simply starts the bifurcation for subclasses that are related to "ArealSampling" classes for down logs. One slot is added in the process, that for a "downLog" object. Please note that this class is also *virtual*, and so just helps form the fact that its subclasses are "InclusionZone" classes for downed logs.

```
R> getClass('downLogIZ')
```

```
Virtual Class "downLogIZ" [package "sampSurf"]

Slots:

Name:       downLog  description        units         bbox      spUnits
Class:      downLog    character    character       matrix          CRS

Name:     puaBlowup puaEstimates    userExtra
Class:      numeric         list          ANY

Extends: "InclusionZone"

Known Subclasses: "standUpIZ", "chainSawIZ", "sausageIZ"
```

## 4 The "standUpIZ" Class

gove&vanDeusen:2011
Gove and Van Deusen (2011) recognize three main protocols for sampling down logs with fixed-area plots. The so-called "standup" method is probably the most commonly used. Basically, the standup method includes the log if the center of the large-end of the log lands within the sample plot. Therefore, the inclusion zone can be envisioned as centered on that point of the log. The subclass is defined as...

```
R> getClass('standUpIZ')
```

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

Slots:

Name:  circularPlot      downLog  description        units         bbox
Class: circularPlot      downLog    character    character       matrix

Name:       spUnits    puaBlowup puaEstimates    userExtra
Class:          CRS      numeric         list          ANY

Extends:
Class "downLogIZ", directly
Class "InclusionZone", by class "downLogIZ", distance 2
```

It is seen to be a direct extension of the "downLogIZ" class, having added a slot for the "circularPlot" object.

## 4.1 Object creation

The constructor for the subclass takes a "downLog" object and a fixed-plot radius, the "circularPlot" object is created from the latter using the same units as those in the "downLog" object...

```
R> dl = downLog(species='red maple', logLen=10, buttDiam=10,
+               topDiam=2, center=c(x=3,y=2), logAngle=pi/4,
+               units='English')
R> su = standUpIZ(dl, 5)
R> summary(su)


Object of class: standUpIZ
------------------------------------------------------------
inclusion zone for "standup" method
------------------------------------------------------------

InclusionZone...
  units of measurement:  English
  Per unit area blowup factor: 554.62315 per acre

  Object bounding box...
         min       max
x -5.5330166 6.5944595
y -6.5349045 5.5944595


downLog component...
  Spatial ID: log:0.398508
  Volume in cubic feet: 1531.5143 per acre
  Number of logs: 554.62315 per acre

standUpIZ...
  use "summary" on the circularPlot slot for details


R> summary(su@circularPlot)


Object of class: circularPlot
------------------------------------------------------------
```

```
fixed area circular plot
------------------------------------------------------------

ArealSampling...
  units of measurement:  English
  spatial units:  NA
  spatial ID: cp:4cfb8d32
  location...
    x coord:  -0.53553391
    y coord:  -1.5355339
    (Above coordinates are for plot center)

circularPlot...
  radius = 5 feet
  area = 78.539816 square feet (0.001803 acres)
  Number of perimeter points: 101 (closed polygon)
```

In the above example, the summary shows the extents of the object. Note that summaries of the "downLog" and "circularPlot" slots can easily be generated as shown.

## 4.2 Plotting the object

The `plot` generic function has also been extended to be able to handle plotting of the objects of the "standUpIZ" class. The arguments are again detailed in the help page, but here is a simple example...

```
R> plot(su, axes=TRUE, showPlotCenter=TRUE, cex=2, showNeedle=TRUE)
```

A couple things are worth noting in Figure 2. The overall bounding box includes the two objects, the inclusion zone (fixed-area plot) and the log, as is also shown in the above summary. The center of the plot is correctly aligned with the center of the large-end of the log. In addition, everything is relative to the location of the exact center of the log as defined in the "downLog" class. The correct juxtaposition of the inclusion zone is shown with regard to the log's rotation angle (the `logAngle` slot in the object). Again this information is all self-contained within the object. For example, in this object, the log angle can be found simply as...

```
R> su@downLog@logAngle
```

```
[1] 0.78539816
```

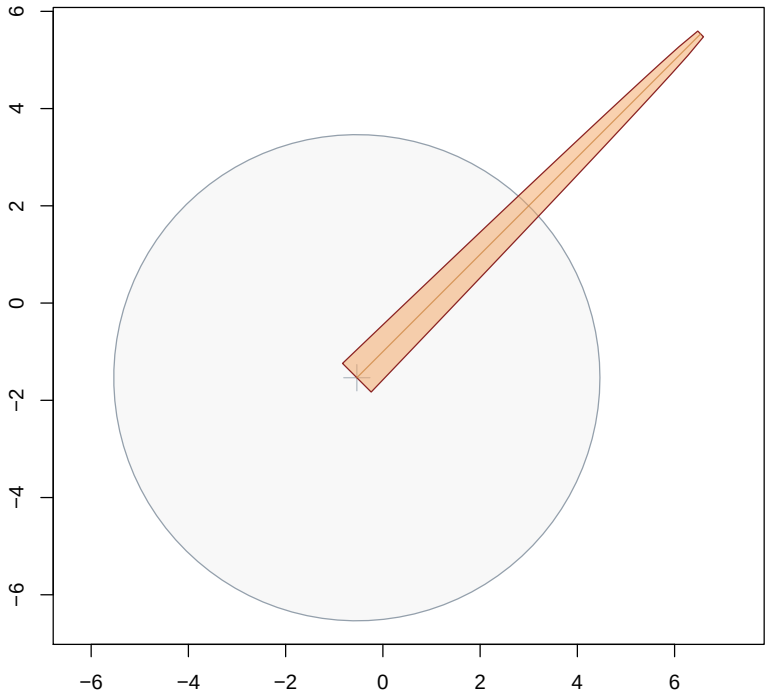


Figure 2: A "standupIZ" object. fig:suIZ

```
R> identical(dl@logAngle, su@downLog@logAngle)
```

```
[1] TRUE
```

## 5 The "chainSawIZ" Class

A second, but probably less commonly used protocol is what Gove and Van Deusen (2011) call the "chainsaw" method. With this method, the intersection of the plot with the log defines a 'sliver' of the log which is measured for volume. They discuss three main sub-protocols for the implementation of this method. This class essentially allows implementation of all three, control over which is determined by the program instantiating the objects of this class. The class extends "downLog" directly...

```
R> getClass('chainSawIZ')

Class "chainSawIZ" [package "sampSurf"]

Slots:

Name:     circularPlot          sliver            bolt         downLog
Class:    circularPlot SpatialPolygons            list         downLog

Name:      description           units            bbox         spUnits
Class:       character       character          matrix             CRS

Name:        puaBlowup    puaEstimates       userExtra
Class:         numeric            list             ANY

Extends:
Class "downLogIZ", directly
Class "InclusionZone", by class "downLogIZ", distance 2
```

wherein the following slots have been added...

## 5.1 Class slots

- *circularPlot*: An object of that class.
- *sliver*: A "SpatialPolygons" object representing the intersection (sliver) between the circular plot and the log.
- *bolt*: A `list` defining the 'minimal bounding bolt' within the log that fully encompasses the sliver. The slots in the `list` are defined as...
  - *rotBolt*: A matrix representation of the minimal bounding bolt.
  - *boltVol*: The bolt's volume.
  - *sectVol*: The volume of the sliver section.
  - *area*: A vector containing the proportion of the bolt that the sliver encompasses, the bolt polygon area, and the sliver polygon area.

## 5.2 Object creation

The constructor for the subclass takes a "downLog" object, a fixed-plot radius, and a center point for the fixed-radius plot...

```
R> cs = chainSawIZ(dl, 2, c(x=7.8, y=3.9), runQuiet=FALSE)


Percentage sliver is of bolt area = 18.075158
Bolt volume (not expanded) = 0.019930252
Section/sliver volume (not expanded) = 0.0036024245


R> summary(cs)


Object of class: chainSawIZ
------------------------------------------------------------
inclusion zone for "chainsaw" method
------------------------------------------------------------

InclusionZone...
  units of measurement:  English
  Per unit area blowup factor: 3466.3947 per acre

  Object bounding box...
          min       max
x -0.83016173 9.8000000
y -1.83016173 5.8997483


downLog component...
  Spatial ID: log:0.398508
  Volume in cubic feet: 12.487425 per acre
  Number of logs: 3466.3947 per acre
  The above estimates are based on the expanded sliver portion.
  The following are unexpanded...
    Total log volume: 2.7613602 cubic feet
    Bounding bolt volume: 0.019930252 cubic feet
    Sliver volume: 0.0036024245 cubic feet
    Sliver area is 0.18075158 of bounding bolt

chainSawIZ...
  use "summary" on the circularPlot slot for sample plot details
```

The `summary` method shows a bit more information about the resulting object than with other protocols. Most importantly, it shows information about the sliver size and volume, and the minimal bounding bolt's size with respect to the entire log. We can see from this example, that we truly must have just cut a little sliver off, since it's volume is so small. However, in the next example, we take a good chunk of the log in the plot intersection...

```
R> cs2 = chainSawIZ(dl, plotRadius=3, c(x=2, y=2), runQuiet=FALSE)
```

```
Percentage sliver is of bolt area = 97.81168
Bolt volume (not expanded) = 1.5331178
Section/sliver volume (not expanded) = 1.4995682
```

The constructor was requested to print the necessary information for comparison, and it will be verified by the graphics below.

### 5.3 Plotting the object

The `plot` generic function has also been extended to be able to handle plotting of the objects of the "chainSawIZ" class. The arguments are again detailed in the help page, but here is a simple example...

```
R> plot(cs, axes=TRUE, showNeedle=TRUE, showPlotCenter=TRUE)
R> plot(cs2, add=TRUE, showPlotCenter=TRUE, showLog=FALSE, izColor=NA)
```

fig:csiz
Figure 3 presents a comparison of the two "chainSawIZ" objects created above, using the same log for reference. As mentioned above, the second object takes a much larger slice of the log for an estimate of volume than the first. It is also worth noting that the first example shows an intersection where the log's 'needle' is not encountered. This would therefore not qualify as falling into the sausage-shaped inclusion zone for the overall log, unless it also intersected the end of the log (see Gove and Van Deusen (2011) for details). In the Figure, the minimal bounding bolts are delineated by the dotted lines, and the slivers are clearly seen in the intersection, also shown as dashed.

## 6 The "sausageIZ" Class

Another method, which allows the log to be sampled if any part of it's "needle" falls inside the circular fixed-area plot is called the "sausage" method by Gove and Van Deusen (2011). It is a direct subclass of "downLogIZ"...

```
R> getClass('sausageIZ')
```

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

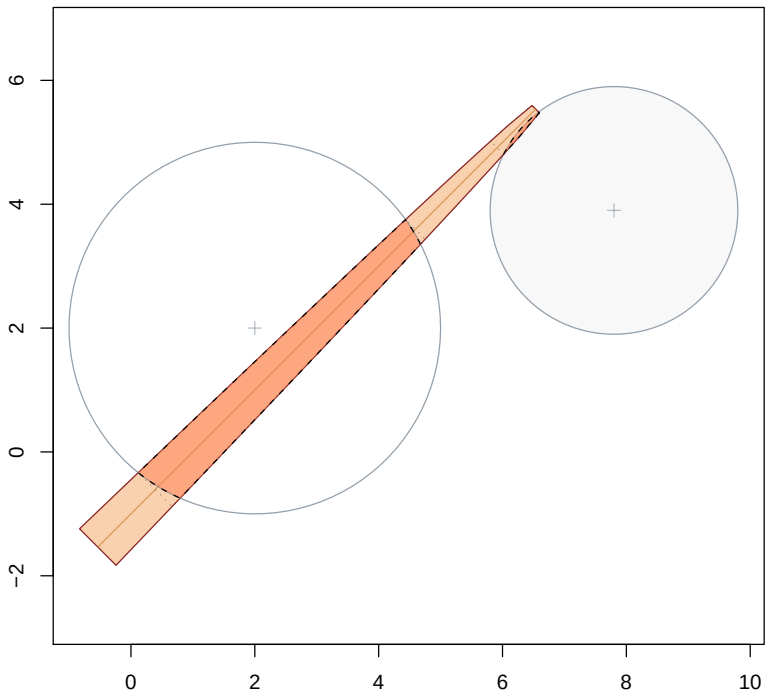


Figure 3: "chainsawIZ" objects comparison. fig:csiz

```
Slots:

Name:          sausage           radius             area        perimeter
Class:          matrix          numeric          numeric SpatialPolygons

Name:    pgSausageArea          downLog      description            units
Class:         numeric          downLog        character        character

Name:             bbox          spUnits        puaBlowup    puaEstimates
Class:          matrix              CRS          numeric             list

Name:        userExtra
Class:             ANY

Extends:
Class "downLogIZ", directly
```

```
Class "InclusionZone", by class "downLogIZ", distance 2
```

wherein the following slots have been added...

## 6.1 Class slots

- *sausage*: A matrix representation of the sausage inclusion zone in homogeneous coordinates. This can be manipulated and plotted as desired for easy access to the inclusion zone where needed.
- *radius*: The radius for the fixed-area plot that determines the sausage inclusion zone area.
- *area*: The exact area of the inclusion zone.
- *perimeter*: This is the inclusion zone perimeter as a "SpatialPolygons" object.
- *pgSausageArea*: This is the area of the sausage inclusion zone as calculated from the polygon in the `perimeter` slot using the "SpatialPolygons" object. As such, it is an approximation of the true area of the inclusion zone, which is given in the `area` slot. This just enables us to see how close the graphic representation is to the real area.

## 6.2 Object creation

The constructor for the subclass takes a "downLog" object and a fixed-plot radius, much like the "standUpIZ" class. From this the inclusion zone is determined...

```
R> sa = sausageIZ(dl, 5)
R> summary(sa)
```

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

InclusionZone...
  units of measurement:  English
  Per unit area blowup factor: 243.97919 per acre

  Object bounding box...
         min       max
```

```
x -5.5348916 11.535534
y -6.5348916 10.535534


downLog component...
  Spatial ID: log:0.398508
  Volume in cubic feet: 673.71443 per acre
  Number of logs: 243.97919 per acre

sausageIZ...
  Spatial ID: sausageIZ:5bfd4210
  radius = 5 feet
  area = 178.53982 square feet (0.004099 acres)
  Number of perimeter points: 101 (closed polygon)


R> sa@pgSausageArea


[1] 178.48489
```

We note from the above that there is a little bias in the area of the inclusion zone when computed from the "SpatialPolygons" object. This is small, but shows the approximate nature of the inclusion zone perimeter. Using more points to define the half-circles on each end of the zone allows this area estimate to converge to the actual area. Using too few points obviously results in larger bias. This should be kept in mind when constructing grids for sampling surface simulations eventually with these objects, because the perimeter will determine what grid cell centers are within the inclusion zone, and hence get assigned a surface attribute value.

## 6.3 Plotting the object

The `plot` generic function has also been extended to be able to handle plotting of the objects of the "sausageIZ" class. The arguments are again detailed in the help page, but here is a simple example...

```
R> plot(sa, axes=TRUE, showLogCenter=TRUE, cex=2, showNeedle=TRUE)
R> dl2 = downLog(species='white pine', logLen=4, buttDiam=6,
+                topDiam=0, center=c(x=1,y=4), logAngle=3*pi/4, units='English'
+               )
R> sa2 = sausageIZ(dl2, 5)
R> plot(sa2, add=TRUE, izCol=NA)
R> plot(su@circularPlot@perimeter, lty='dashed', border='grey50', add=TRUE)
```

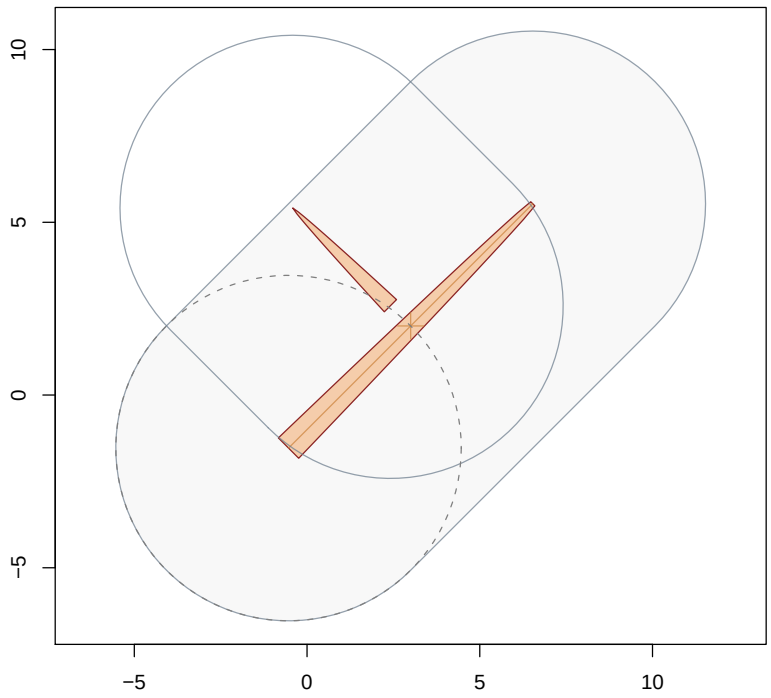


Figure 4: "sausageIZ" objects with "standUpIZ" perimeter for comparison. fig:saiz

fig:saiz
Figure 4 illustrates the sausage inclusion zones for two down logs. In the case of the longest log, the standup inclusion zone perimeter is shown for comparison.

One other thing to keep in mind under sausage sampling is that the log and inclusion zone centers coincide. Therefore, plotting the log center will identify them both using `showLogCenter=TRUE`; using `showPlotCenter=TRUE` does nothing at present. If you really want to also display the plot center, it can always be access via the `downLog` slot in the object itself: the `location` slot in this object is an object of class "SpatialPoints" and therefore can be accessed as the inclusion zone plot center.

## 7 Container Classes

As we've mentioned in other vignettes, there needs to be a mechanism to have multiple versions of, e.g., "downLogIZ" objects or subclasses thereof stored in a population of sorts. This means there

needs to be class and constructor methods, along with summary, plot, etc. One could just use a `list` structure, and this would work fine, but then there would be no ability to associate methods. So it is better to adopt an S4 class structure, albeit simple, that can encapsulate the behavior desired.

## 7.1 Class "downLogIZs"

This container holds a collection of "downlogIZ" objects along with some spatial information...

```
R> showClass('downLogIZs')
```

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

Slots:

Name:       iZones       units        bbox description
Class:        list   character      matrix   character
```

Like the "downLogs" class, this is also a very simple class. At present, it holds a list of "downLogIZ" objects in the `iZones` slot, and the overall bounding box for the collection or population, along with a description and units for all objects. The bounding box is useful in plotting and will also be used in the sampling surface routines.

### 7.1.1 Class construction

In keeping with the previous naming conventions, the constructor function for this class is `downLogIZs`. A collection can be created via the following steps, though there are alternatives in each case...

1. *Measured population*:
   (a) Use the field measurements to make individual "downLog" objects, and then place them into a `list` container.
   (b) Create valid "downLogIZ" subclass objects for each log—they all must be of the same class.
   (c) Use the "downLogIZs" constructor on this list of logs, to create the collection.
2. *Synthetic population*:

(a) Draw a sample of "downLog"s randomly within some sample area, using the "downLogs" constructor.

(b) Then create a `list` of "downLogIZ" subclass objects.

(c) Finally, group these into the container class with "downLogIZs".

The above are essentially the same in practice, as similar steps are followed whether a synthetic or real population of logs is used. At present, one must specify objects of exactly the same subclass in the `list` constructor. That is, you can not mix sampling methods as demonstrated below. Please see the "downLogs" constructor and *"The Stem Class"* vignette for more options in each of the first steps above.

In the first example, we will pretend that `dl` and `dl2` are logs from a field sample, and we then have made sausage inclusion zones for these in the example above. Therefore, we follow the rest of the steps...

```
R> sal = list(sa,sa2)
R> sapply(sal, class)
```

```
[1] "sausageIZ" "sausageIZ"
```

```
R> izs = downLogIZs(sal, description = 'two inclusion zones...\nboth sausage')
R> sapply(izs@iZones, class)
```

```
[1] "sausageIZ" "sausageIZ"
```

```
R> summary(izs)
```

```
Object of class: downLogIZs
------------------------------------------------------------
two inclusion zones...
both sausage
------------------------------------------------------------
Container class object...
  There are 2 inclusion zones in the population
  Inclusion zones are of class: sausageIZ
  Units of measurement:  English

  Encapulating bounding box...
```

```
         min       max
x -5.5348916 11.535534
y -6.5348916 10.535534
```

The above example is fine, because both of the member objects in the list used to construct the "downLogIZs" object were of class "sausageIZ". However, in the following example, we are mixing sampling methods, which is not allowed...

```
R> sal2 = list(su,sa)
R> iizs = tryCatch(downLogIZs(sal2),  error = function(e) e)
R> if(is(iizs, 'simpleError')) cat(iizs[[1]],'\n')
```

```
invalid class "downLogIZs" object: You can not mix inclusion classes in the population!
```

```
R> sapply(sal2,class)
```

```
[1] "standUpIZ" "sausageIZ"
```

It is very simple to draw a simulated population of logs plus their inclusion zone. As shown below, this can be done for a log population of any size. Since the logs are stored within the objects as a list structure, we can apply the list-based object constructors via `lapply` to return a new list that can directly be converted to a container object...

```
R> dls = downLogs(6, units='English', buttDiam=c(6,20))
R> dls.su = lapply(dls@logs, 'standUpIZ', plotRadius=5)
R> izs.su = downLogIZs(dls.su)
R> dls.sa = lapply(dls@logs, 'sausageIZ', plotRadius=5)
R> izs.sa = downLogIZs(dls.sa)
```

In this case we have created two more container objects for a set of "downLog"s within a "downLogs" object, the first for the sausage method, and the second for the standup method.

## 7.2 Plotting the object

There is a plot method for "downLogIZs" objects that works as usual.

In the case of the first example, where the "downLogIZs" object was created using 'field data,' we have generate Figure 5...

```
R> plot(izs, axes=TRUE, showLogCenter=TRUE, cex=2, showNeedle=TRUE)
```

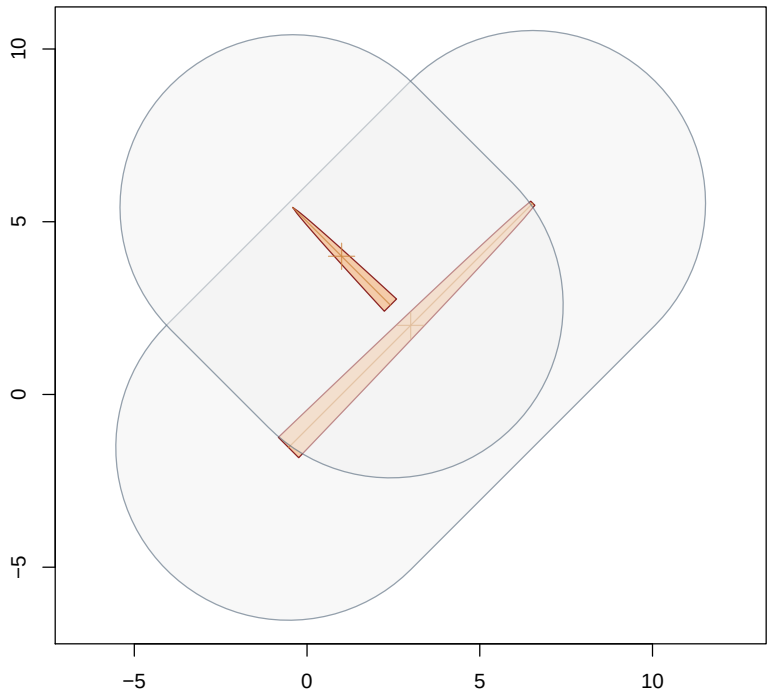


Figure 5: "downLogIZs" object with "sausageIZ" objects in the container. fig:dlIZs

In the second example, where we use synthetic logs, we generate Figure 6 showing the "standUpIZ" inclusion zones superimposed over the "sausageIZ" inclusion zones, purely for comparison... fig:dlIZs2

```
R> plot(izs.sa, axes=TRUE, showLogCenter=TRUE, cex=2, showNeedle=TRUE)
R> plot(izs.su, add=TRUE, izColor=NA)
```

Note that in generating Figure 6, we have plotted the sausage collection before the standup collection, because the former has a larger overall bounding box that will contain all of the collected objects, regardless of their "InclusionZone" classes. Note again, however, the restriction that within a collection, all of the objects must share the same "InclusionZone" class. fig:dlIZs2

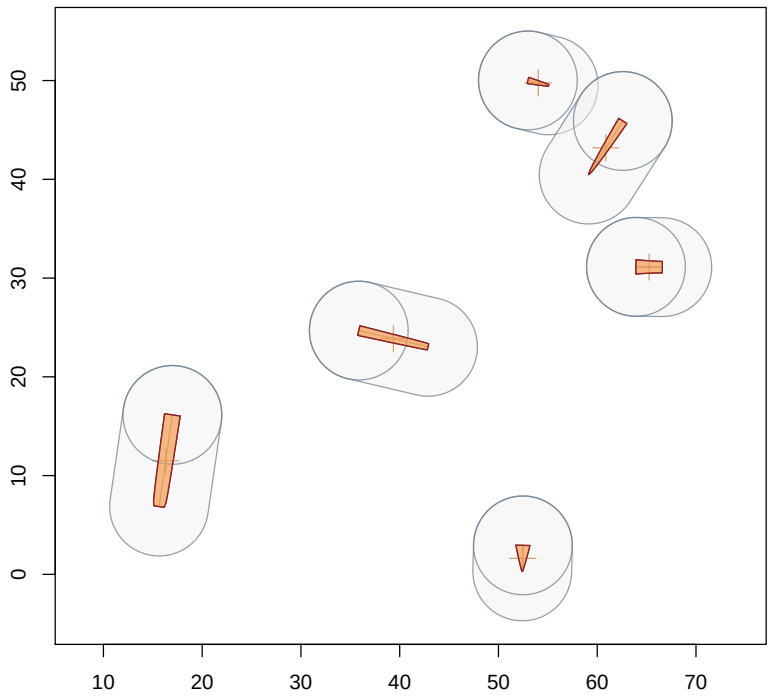


Figure 6: "downLogIZs" object of "sausageIZ" objects, with "standUpIZ" objects superimposed. fig:dlIZs2

## References

Deusen:2011 J. H. Gove and P. C. Van Deusen. On fixed-area plot sampling for downed coarse woody debris. *Forestry*, 2011. Submitted August 2010. 4, 7, 10