The "Stem" 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

Tuesday 21st December, 2010
11:00am

Contents

1 Introduction

The "Stem" class is a virtual class that is meant to form a basis for more descriptive subclasses. The idea is to contain all the common information for, e.g., down logs and standing trees, within this base class, then add more information as needed by generating new subclasses. This, of course, could go on for any number of inheritance levels. Time will tell whether this ends up being a reasonable approach, and it is possible that some of the information left for the subclasses might be better moved to the base class, but this can be remedied later.

Currently, just the dual branches for downed logs ("downLog" class) and standing trees are envisioned, but perhaps more will present themselves. In addition, the base "Stem" class, and thus the subclasses, all rely on the `sp` package in R. It is used to provide a convenient platform for

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

graphical display that will allow not only user-defined coordinates, but also will support any coordinate system defined in `proj.4`—with the exception of lat-long, since spatial coordinates must be commensurate with log dimension units. The drawback to this approach, of course, is that some familiarization with `sp` is required for extending classes. However, for the average user, the spatial components are encapsulated within the objects and plot naturally, as we shall see in the examples below.

The "Stem" class was created to be used within the sampling surface system of simulation. This is detailed elsewhere in the package documentation (see, e.g., *"The sampSurf Package Overview"*), and the class structure is contained within the **R** package `sampSurf`.

An overview of the "Stem" class structure is presented in Figure fig:Stem 1.

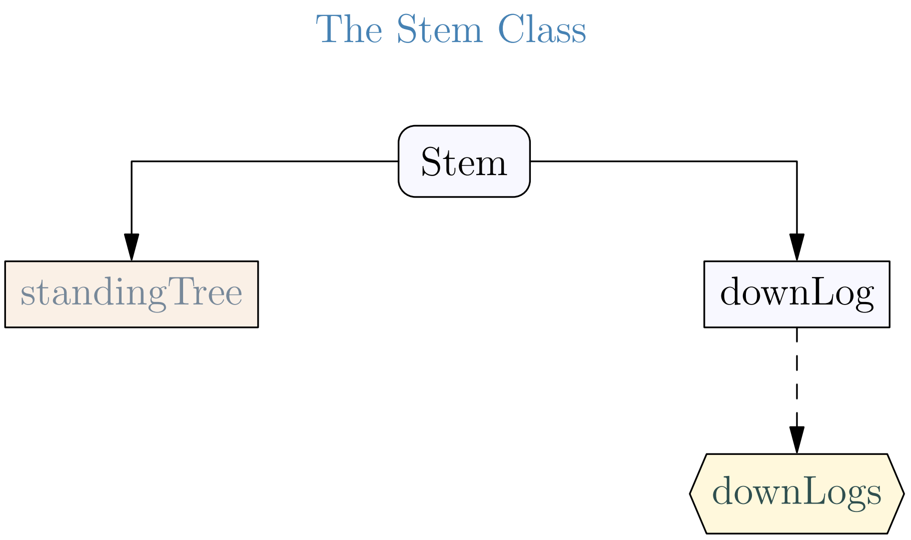

Figure 1: An overview of the "Stem" class. The "standingTree" class has not been implemented yet and the "downLogs" class is not really a subclass, but is instead a container class for a population of "downLog" objects. fig:Stem

2 The "Stem" Class

Again, this is a virtual class, so no objects can be directly created from it. However, some methods for this class are defined to give basic functionality to subclasses if desired. The class definition is given as...

```
R> getClass('Stem')

Virtual Class "Stem" [package "sampSurf"]

Slots:

Name:        species         units      location       spUnits   description
Class:     character     character SpatialPoints           CRS     character

Name:      userExtra
Class:           ANY

Known Subclasses: "downLog"
```

We see from the above listing that there are several slots defined for this class. These are detailed below.

2.1 Class slots

- *species*: This is some description of the species. It is entirely left to the user whether it might be codes, common names, Latin names, all of the above, etc.
- *units*: A character string specifying the units of measure. Legal values are "English" and "metric."
- *location*: This is a "SpatialPoints" representation of the location of the object. For example, in the "downLog" class, this is the center of the log, both longitudinally and radially.
- *spUnits*: A valid string of class "CRS" denoting the spatial units coordinate system ("?CRS" for more information) as in package `sp`.
- *description*: A character vector with any comments about the stem if desired.
- *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 "downLog" Class

This is a direct subclass of "Stem" as shown above, and is a general class for down coarse woody debris. It should be general enough to contain data generated for simulation and those collected in a field study, where the log locations are measured (locations could also be generated for measured logs if missing).

One very important aspect of this class design came about mainly because of graphical considerations. It is very important that the diameters be in the *same* units as length as stored within the object. That means, if length is in meters, then diameters should be in meters, not centimeters. We will present ways to handle this in object creation later so that it is not burdensome.

3.1 "downLog" class slots

The object slots are defined as...

```
R> showClass('downLog')
```

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

Slots:

Name:        buttDiam         topDiam          logLen        logAngle
Class:        numeric         numeric         numeric         numeric

Name:       solidType          logVol           taper         profile
Class:    numericNULL         numeric      data.frame      data.frame

Name:          rotLog           spLog    slNeedleAxis         species
Class:         matrix SpatialPolygons     SpatialLines       character

Name:           units        location         spUnits     description
Class:      character   SpatialPoints             CRS       character

Name:       userExtra
Class:            ANY

Extends: "Stem"
```

Notice that the last six slots are inherited from "Stem." The others are new and are defined as follows...

- *buttDiam*: The large-end diameter, in the same units as length.

- *topDiam*: The small-end diameter, in the same units as length.
- *logLen*: The log length in meters or feet.
- *logAngle*: The angle of lie for the log. It should be relative to the log center, which is defined as the center of the 'needle' that defines the long axis, and also with respect to cross-section—the `location` slot in other words. Note that the canonical position is the log lying with tip due east with center at $(0, 0)$. Angles of rotation are counter-clockwise from this position. This is an important point to consider if, for example, log angles have been taken for north, they must first be converted to east as the origin.
- *solidType*: This is the taper and volume equation exponent parameter. If one measures taper and volume directly, it will not be used, but if either are computed, it should be an appropriate approximation to log form. See below for more details on the default taper equations.
- *logVol*: The log volume in cubic units of length.
- *taper*: The taper for the log. This can be measured and passed when creating the object. Normally, however, it will be generated using the `buttDiam`, `topDiam` and a taper equation with `nSegs` segments (see `downLog` constructor below). If the default equation is used, then it will also rely on `solidType`. The taper *must* be a data frame with columns labelled `diameter` and `length`.
- *profile*: The two-dimensional stem profile based on `taper`. This is always oriented as if the log were standing (north) for ease of interpretation. It is a data frame with columns `radius` and `length`. It is reflective, containing both sides of the log in a closed polygon in the `sp` "SpatialPolygons" sense. The central longitudinal axis is located at zero, with the butt at zero.
- *rotLog*: This is the `profile` rotated to its correct position in terms of `logAngle` and `location`. It is a matrix in homogeneous coordinate form with columns `x`, `y` and `hc`.
- *spLog*: Both `profile` and `rotLog` (as well as `taper`) are intermediate steps to the "SpatialPolygons" object of this name. It has the exact same data as `rotLog`, but in the "SpatialPolygons" form. This allows the log to be easily plotted in the correct juxtaposition using the `sp` package routines.
- *slNeedleAxis*: Holds the "SpatialLines" object defining the longitudinal 'needle' axis of the log, again in the correct sense of location and rotation.

3.2 Object creation

An object of class "downLog" can be created using the `S4` `new` operator. However, because the slot specifications are rather lengthy, this approach is not recommended; but one can get a 'dummy' downed log from simply...

```
R> dl = new('downLog')
R> validObject(dl)
```

```
[1] TRUE
```

```
R> slot(dl, 'spLog')
```

```
An object of class "SpatialPolygons"
Slot "polygons":
list()

Slot "plotOrder":
integer(0)

Slot "bbox":
     min max
[1,]  NA  NA
[2,]  NA  NA

Slot "proj4string":
CRS arguments: NA
```

Note that a valid object is created, but it is not of much use. For example, the `taper` slot is trivial by default, being the entire log. The default object generated above, has random slots generated in metric as $0.1 \leq$ `buttDiam` ≤ 0.8, `topDiam` $=$ `runif(1,0,0.9)` $\times$ `buttDiam` with length lying between 1 and 10 meters; `logAngle` is random over zero to 2π. As shown, there is no information for plotting the log via the `sp` package. Clearly this is all just used to get a minimally valid object and the user should have much more control. Now, one can certainly specify any other slots when using `new`, but that becomes cumbersome. On the other hand, one can generate random dimensions and angles for logs this way, and then use them subsequently in a call to the constructor to get a complete "downLog" object if desired, though using `sampleLogs` (see below) is a much better choice for this.

The "downLog" class has a constructor function by the same name that will allow object creation (see documentation for a complete list of arguments and the function), and should be used in preference to `new`…

```
R> dl = downLog(species='eastern white pine', logLen=8,
+              buttDiam=50, topDiam=0, centerOffset=c(x=3,y=2),
+              logAngle=pi/4, description='Durham, NH',
+              userExtra=list(decayClass=4))
```

The key argument in this constructor is `object`, it defines the signature of this constructor. If it is missing, as in the case above, then it is assumed that no taper data exist, and that instead, taper is to be calculated for the log. In this case, the following taper and volume equations are applied vandeusen:1990 (Van Deusen, 1990)...

$$
\begin{aligned}
d(l) &= D_u + (D_b - D_u)\left(\frac{L-l}{L}\right)^{\frac{2}{r}} \\
v(l) &= \frac{\pi}{4}\left[D_u^2 l + L(D_b - D_u)^2 \frac{r}{r+4}\left(1 - \left(1 - \frac{l}{L}\right)^{\frac{r+4}{r}}\right)\right. \\
&\qquad \left. + 2LD_u(D_b - D_u)\frac{r}{r+2}\left(1 - \left(1 - \frac{l}{L}\right)^{\frac{r+2}{r}}\right)\right]
\end{aligned}
$$

where D_b is the large-end or butt diameter, with small-end diameter D_u, $0 \le l \le L$ is the intermediate log length for volume or diameter estimates, and r is a parameter such that $0 \le r < 2$ generates a neiloid, $r = 2$ generates a cone, and $r > 2$ generates a paraboloid. The r parameter is specified directly via the `solidType` argument, which defaults to $r = 3$. Note again that with these taper and volume equations, the units for diameters must be the same as for length.

The rest of what goes on in generating the object slots is pretty straightforward. Once the taper has been determined, a profile can be generated. Then the log is rotated and translated to its final position as specified by `logAngle` and `centerOffset`. Log volume can be passed to the constructor, in which case, it overrides the taper function version. Note that section volumes are not calculated or saved. If this is desired, one could generate a subclass of "downLog" with this and other information for instance. A summary of the newly created object is supplied via...

```
R> summary(dl)
```

```
Object of class: downLog
------------------------------------------------------------
Durham, NH
------------------------------------------------------------

Stem...
  Species:  eastern white pine
  units of measurement:  metric
  spatial units:  NA
  location...
    x coord:  3
    y coord:  2
    (Above coordinates are for log center)
  Spatial ID: log:0.223088
```

```
downLog...
  Butt diameter = 0.5 meters (50 cm)
  Top diameter = 0 meters (0 cm)
  Log length =  8 meters
  Log volume =  0.67319843 cubic meters
  Log angle of lie =  0.78539816 radians (45 degrees)
  Taper parameter =  3

Taper (in part)...
    diameter length
1 0.50000000    0.0
2 0.48319126    0.4
3 0.46608488    0.8
4 0.44865856    1.2
5 0.43088694    1.6
6 0.41274091    2.0

  "Note: userExtra" slot is non-NULL
```

Now, when the method signature argument is present, it must be a data frame specifying the log's taper values, presumably as measured in the field. We can see how this might work with the following...

```
R> lt = dl@taper
R> nt = length(lt)
R> fdx = ifelse(1:nt%%2, TRUE, FALSE)
R> (newTaper = lt[fdx,])
```

```
     diameter length
1  0.50000000    0.0
3  0.46608488    0.8
5  0.43088694    1.6
7  0.39418676    2.4
9  0.35568933    3.2
11 0.31498026    4.0
13 0.27144176    4.8
15 0.22407024    5.6
17 0.17099759    6.4
19 0.10772173    7.2
21 0.00000000    8.0
```

And create a new object (obviously with the same dimensions as the log it was taken from above) using this taper...

```
R> dl2 = downLog(newTaper, species='eastern white pine', logLen=8,
+                buttDiam=50, topDiam=0, centerOffset=c(x=2.5, y=2),
+                logAngle=3*pi/4, description='Durham, NH',
+                userExtra=list(decayClass=1) )
R> summary(dl2)
```

```
Object of class: downLog
------------------------------------------------------------
Durham, NH
------------------------------------------------------------

Stem...
  Species:  eastern white pine
  units of measurement:  metric
  spatial units:  NA
  location...
    x coord:  2.5
    y coord:  2
    (Above coordinates are for log center)
  Spatial ID: log:0.670622

downLog...
  Butt diameter = 0.5 meters (50 cm)
  Top diameter = 0 meters (0 cm)
  Log length =  8 meters
  Log volume =  0.67319843 cubic meters
  Log angle of lie =  2.3561945 radians (135 degrees)
  Taper parameter =  3

Taper (in part)...
     diameter length
1  0.50000000    0.0
3  0.46608488    0.8
5  0.43088694    1.6
7  0.39418676    2.4
9  0.35568933    3.2
11 0.31498026    4.0

  "Note: userExtra" slot is non-NULL
```

In the above example, we have essentially the same log, but with fewer sections derived from measurements passed to the constructor as a data frame in the signature argument. The log angle and offset are a little different than with the previous log.

If you are unfamiliar with S4 classes, this is a very simple illustration of how one can use the signature to dictate the results. However, this is not really the full power of S4 in this case because we have simply done all the computations for either a data frame or missing `object` argument within the same routine. S4 allows much more powerful capabilities for having different signatures and different routines (possibly inherited) operating on them. That was unnecessary in this case.

Note that in the above examples, the `summary` generic function has been extended with methods for the "downLog" class.

3.3 Plotting the object

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

```
R> plot(dl, axes=TRUE, showLogCenter=TRUE)
R> plot(dl2, add=TRUE, showNeedle=TRUE, showLogCenter=TRUE, cex=3)
```

Note in Figure fig:dl 2 that the logs are in their correct position and rotation. Here we have just used user-defined coordinates based on meters, but other CRS projections could be used.

3.3.1 Coordinate reference systems (CRS)

If other coordinate reference systems supported by "CRS" are used, one must be very careful to have everything about the log commensurate with the system or something will be incorrect when plotting. It is up to the user to monitor this when creating logs, especially when they are subsequently to be used in the later sampling surface interface. That is, if the 'CRS' units are in metric, then your log measurements better be as well. If you use, geographic (lat-long) for spatial coordinates, you should convert them first to some system based on meters or feet.

Please note that you can enter any character string into the `spUnits` slot via the constructor and have it be accepted if the `rgdal` package is not installed on your system. This package is what does all the spatial coordinate system checks.

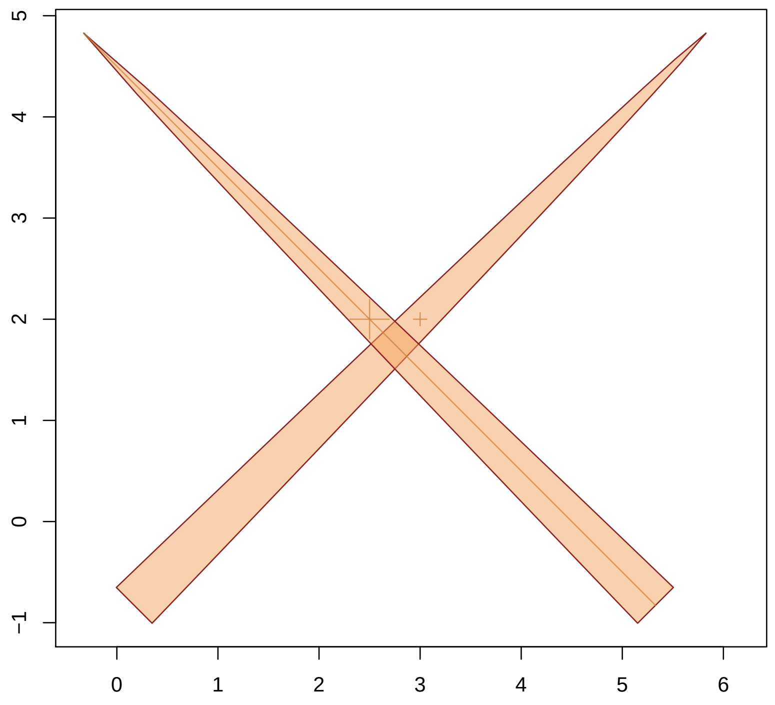

Figure 2: Two downed logs generated from the examples. fig:dl

4 Creating Synthetic Logs

The routine `sampleLogs` is a very simple function that can be used to create simulated logs for use in studying sampling methods for down coarse woody debris. Its use is detailed within the help pages, but its use in the following sections requires a little introduction here. First, the arguments a similar to what we have seen in the basic slots of the “downLog” class. . .

```
R> args(sampleLogs)
```

```
function (nLogs = 2, buttDiams = c(8, 40), topDiams = c(0, 0.9),
    logLens = c(1, 10), logAngles = c(0, 2 * pi), solidTypes = c(1,
        10), species = .StemEnv$species, sampleRect = NULL, startSeed = NA,
    runQuiet = FALSE, ...)
NULL
```

However, the obvious difference lies in the observation that the arguments all take a vector of lower and upper bounds from which the population is simulated. The exceptions are `species`, which can be any character vector of species names, codes, or other identifier, and `sampleRect`, which is a matrix in the form of a `sp` bounding box (`bbox`). This bounding box is used as the enclosing area from which to randomly draw the log center location coordinates in x and y. The `startSeed` parameter is used for adjusting the random number stream within **R**, through the function `initRandomSeed` (see the help file for this function, where more details are provided).

As an example, draw a population of logs from the default settings except for species identifiers...

```
R> format( sampleLogs(5, species=c('RM','ewp','red spruce')), digits=3)
```

```
Note: logs generated within [0,1] bbox!
     species logLen buttDiam topDiam solidType     x     y logAngle logAngle.D
1        ewp   5.79     35.2   16.60       8.2 0.896 0.598     5.47      313.4
2         RM   8.83     15.0   11.16       8.4 0.734 0.218     4.84      277.4
3        ewp   2.11     32.7   21.62       4.5 0.734 0.144     2.02      115.7
4 red spruce   1.90     16.8    4.77       5.8 0.180 0.817     1.44       82.6
5         RM   7.18     36.6   18.09       5.1 0.875 0.358     2.36      135.4
```

`sampleLogs` returns a data frame with many columns the same as the slots in "downLog". The columns `x` and `y` define the center point location of the logs, and `logAngle.D` is simply the log angle in degrees. Because no enclosing bounding box was specified through `sampleRect`, the log center locations all lie within a [0, 1] rectangle as reported (since `runQuiet` is `FALSE` by default).

One important thing to note in this routine is that the bounds one specifies for, e.g., diameters and length, should make sense in the units you are working in. In other words, if you want to generate diameters in cm, you can, and you can also generate diameters in m, it all depends on the bounds for `buttDiam`. Since the constructor for "downLog" objects expects to see diameters in either inches or cm and later converts to feet or meters, you should specify your ranges in inches or cm in general, if the intent is to later create "downLog" objects.

5 Container Classes

This idea essentially comes from C++ and Java. There needs to be a mechanism to have multiple versions of, e.g., "downLog" objects stored in a population or collection. This means there needs to be class definition and associated constructor methods, along with summary, plot, etc. methods to work with these container objects. One could, of course, store these objects within a `list` structure. However, this would not allow methods and generics to be written to act on the objects, since `lists` can contain anything. Thus we make this a class of its own so we can impart an inherent functionality to its objects.

At present, only the container for multiple down logs is established. We might want to make a virtual container class if we extend this to other "Stem" subclasses, or combinations of subclasses (i.e., stand and down material).

5.1 Class "downLogs"

This container holds a collection of "downlog" objects...

```
R> showClass('downLogs')
```

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

Slots:

Name:        logs       units        bbox      stats
Class:       list   character      matrix data.frame
```

This is a very simple class. At present, it holds a list of "downLog" objects and the overall bounding box for the collection or population, along with measurement units and simple statistics...

- *logs*: This is a normal **R** list holding the individual "downLog" objects. Please see the caveate below concerning deleting or adding to this list in your code.
- *units*: The units of measurement. Note that all logs in the collection *must* share the same units of measurement. This is checked at object creation.
- *bbox*: The overall bounding box for the collection, it is useful in plotting the entire collection.
- *stats*: Some simple statistics for the collection in a data frame.

Please note that at the present time this class only partially meets the requirements of a true "container class" in object oriented programming. This is because it does not as yet have methods for object deletion, editing, or addition to the list of down logs. Because the statistics and bounding box are tied to the collection, a caution is in order regarding changing in any way the objects within the list: if you add to or delete from the list, the `bbox` and `stats` slots will be incorrect unless also updated to reflect whatever changes have been made on the list slot. The best way to handle this is to simply extract the list from the object, do whatever editing has to be done to it, then use the constructor below to make a new object. Then everything will be correctly represented within the object. Eventually, routines for editing may be added.

5.1.1 Class construction

In keeping with the previous naming convention, the constructor function for this class is `downLogs`, matching the class name. A collection can be created in two main ways: synthetically, or from existing valid "downLog" objects. However, there are several actual constructor variants of the same name that differ in function based on their signatures as described in the following (with signature arguments in italics)...

1. `object`→`list`, `container`→`missing`: This is the base constructor, all of the following constructors just reformat their inputs into a list containing "downLog" objects and then call this constructor to make the object.

2. `object`→`numeric`, `container`→`matrix`: The here `object` specifies the number of logs that should be generated in the population and the `container` argument specifies a bounding box matrix in the sense of the `sp` package, with row names `c('x', 'y')` and column names `c('min','max')`, from which the log centers will be drawn at random.

3. `object`→`numeric`, `container`→`missing`: This is similar to the previous, but the bounding "box" from which log centers are drawn is specified by the `xlim` and `ylim` arguments. These specify the range in x and y and are internally converted to a matrix so that the second constructor can be called.

4. `object`→`numeric`, `container`→`bufferedTract`: Similar to the last two, where "bufferedTract" is a subclass of class "Tract" (see: *The Tract Class* vignette for details). The log centers are drawn from within the buffer of the object passed in this argument.

5. `object`→`data.frame`, `container`→`missing`: This will accept a previous collection of logs complete with locational information in the form of a data frame returned from `sampleLogs`. Note that these do not have to be synthetic logs, one can make such a data frame out of observed measurements. This constructor calls the "downLog" constructor for each log generated. Therefore, one can pass on arguments to this constructor such as `nSegs`.

In each case, there are other arguments to the constructors that may be passed. These are all detailed in the help files.

In the following, we demonstrate a couple of these constructors. The rest are simple enough to try once the general idea is demonstrated. First, generate a population of random logs from within a default set of x and y limits specified in `xlim` and `ylim` arguments—this corresponds to the third constructor in the previous list (i.e., `object` is `numeric` and `container` is `missing`)...

```
R> dlp = downLogs(15, xlim=c(0,20), ylim=c(10,40), buttDiams=c(10,35))
R> summary(dlp)
```

```
Object of class: downLogs
------------------------------------------------------------
Container class object...
  There are 15 logs in the population
  Units of measurement:  metric
  Population log volume =  2.6459414 cubic meters
  Average volume/log=  0.17639609 cubic meters
  Average length/log=  5.3466667 meters

  Encapulating bounding box...
        min       max
x 1.2786791 22.637344
y 8.3740215 40.087528
```

```
R> plot(dlp, axes=TRUE, showNeedle=TRUE)
R> plot(perimeter(dlp), add=TRUE, border='grey60', lty='dashed')
```

There are several things going on here. First, the limits for the area from which the log centers are drawn are specified in meters (default) via `xlim` and `ylim`. Second, and most importantly, one can pass several of the arguments to this constructor that are used to specify log characteristics in `sampleLogs`[1], because these constructors actually use `sampleLogs` to generate the logs. Figure 3 fig:dlp illustrates the population of logs that were drawn using the constructor in the code above.

As a second example, we use the data frame constructor to generate a collection from a set of logs. In this case, they were generated using `sampleLogs`, but as long as we use the same column names as appear in the resulting data frame from `sampleLogs`, these could just as easily be logs from a field inventory. The use of this method is shown in the following example...

```
R> buff = matrix(c(0,100,0,100), nrow=2, byrow=TRUE,
+                dimnames=list(c('x','y'),c('min','max')))
R> sl = sampleLogs(10, buttDiam = c(1,25), sampleRect = buff)
R> dlp2 = downLogs(sl)
R> summary(dlp2)
```

```
Object of class: downLogs
------------------------------------------------------------
Container class object...
  There are 10 logs in the population
  Units of measurement:  metric
```

[1] `buttDiams`, `topDiams`, `logLens`, `logAngles`, `solidTypes` and `species`—`startSeed` can also be passed in the ... argument list.

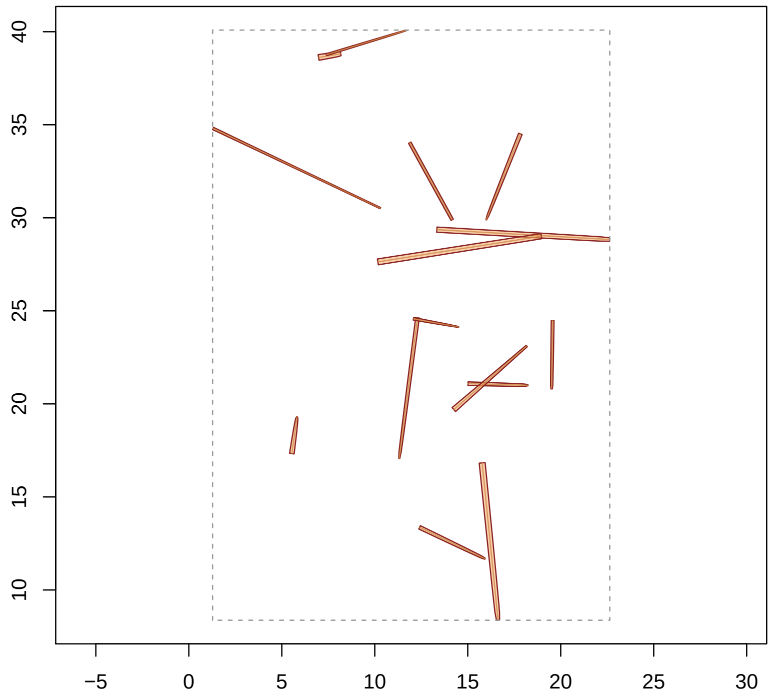

Figure 3: Synthetic population of downed logs generated from the examples along with the bounding box that encloses the entire collection.

fig:dlp

```
  Population log volume =  0.92943764 cubic meters
  Average volume/log=  0.092943764 cubic meters
  Average length/log=  5.748 meters

  Encapulating bounding box...
         min       max
x 18.1156255 84.284673
y  4.8781325 83.631615
```

Note that in this method, the encapsulating bounding box for the log population does not necessarily correspond to the extents passed in the `sampleRect` argument (in this last example, the tract buffer area). In fact, it can be significantly larger than this rectangle because it encompasses the entire log for every log in the population. The `sampleRect` argument, on the other hand, specifies the rectangle from which the log *centers* will be drawn.

Note that we can set the random number seed in a couple different ways, to get the repeatable results; this is explained in more detail in the help files…

```
R> slogs = sampleLogs(10, sampleRect=buff, startSeed=10)
R> dlgs1 = downLogs(slogs)
R> dlgs2 = downLogs(10, buff, startSeed=10)
R> identical(dlgs1, dlgs2)
```

```
[1] TRUE
```

As a final example, to make a population of down logs from already existing “downLog” objects, say from logs measured in the field, we could use the above method for `data.frame`s, or we could use the constructor with the `list` signature. For the latter method, first just create a list containing “downLog” objects, then create the collection…

```
R> ml = dlp@logs[1:2]      #a list of 2 logs as an example
R> dlp2 = downLogs(ml)
R> summary(dlp2)
```

```
Object of class: downLogs
------------------------------------------------------------
Container class object...
  There are 2 logs in the population
  Units of measurement:  metric
  Population log volume =  0.16522054 cubic meters
  Average volume/log=  0.082610272 cubic meters
  Average length/log=  3.495 meters

  Encapulating bounding box...
        min       max
x  5.4176637 17.921539
y 17.3110691 34.547921
```

The advantage of making a separate class for the container objects lies, of course, in the automatic validity checking, and the association of `plot`, `summary`, etc. methods to the object and its advantages over just collection “downLog” objects into a list is evident. For example, in validity checking, each log must be a valid “downLog” object or **R** will not construct the object.

Other constraints on the “downLogs” collection include that each log must be measured in the same `units`. If for some reason one had to mix units, one could just make two subpopulations, one for English and one for metric. Everything would work fine on those subpopulations.

5.1.2 Object coercion

It is sometimes useful to be able to convert backwards from a "downLogs" collection to a data frame in the form of that generated by `sampleLogs`. There is a simple facility for doing this using basic **R** coercion on the object; e.g.,

```
R> format( as(dlp2, 'data.frame'), digits=2)
```

```
           species logLen buttDiam topDiam solidType    x  y logAngle
1 shagbark hickory      5       20     3.5       6.0 16.9 32      4.3
2               rm      2       25     1.3       6.8  5.7 18      1.4
  logAngle.D
1        248
2         82
```

References

deusen:1990 P.C. Van Deusen. Critical height versus importance sampling for log volume: does critical height prevail? *Forest Science*, 36(4):930–938, 1990. 7