Package ‘sampSurf’

March 5, 2021

Type Package

Title Sampling Surface Simulation for Areal Sampling Methods

Version 0.7-6

Date 2021-03-04

Author Jeffrey H. Gove

Maintainer Jeffrey H. Gove <jgove@fs.fed.us>

Depends R (>= 3.0.0), methods, sp(>= 1.3-0), raster(>= 2.9-5), boot (>= 1.3-3)

Imports lattice(>= 0.20-35), latticeExtra (>= 0.6-28), rasterVis(>= 0.45)

Suggests rgl (>= 0.10-19), rgeos (>= 0.4-3)

Description
Sampling surface simulation is useful in the comparison of different areal sampling methods in forestry, ecology and natural resources. The sampSurf package allows the simulation of numerous sampling methods for standing trees and downed woody debris in a spatial context. It also provides an S4 class and method structure that facilitates the addition of new sampling methods.

License GPL (>= 3)

LazyLoad yes

LazyData yes

Collate zzz.R transparentColorBase.R initRandomSeed.R boltDimensions.R bboxCheck.R bboxToPoly.R bboxSum.R checkStemDimensions.R defStemEnv.R transfMatrix.R spCircle.R sampleLogs.R sampleTrees.R sampSurfClassUnions.R chainsawSliver.R StemClass.R StemContainerClass.R TractClass.R TractConstructors.R downLog.R downLogs.R standingTree.R standingTrees.R ArealSamplingClass.R ArealSamplingConstructors.R InclusionZoneClass.R InclusionZoneConstructors.R izContainer.R izContainerConstructors.R InclusionZoneGridClass.R InclusionZoneGridConstructors.R izGridConstruct.R mirage.R sampSurfClass.R sampSurfConstructors.R hist.R plotStem.R

plotStemContainer.R plotArealSampling.R plotInclusionZone.R
plotInclusionZoneGrid.R plotTract.R plotizContainer.R
plotSampSurf.R showArealSampling.R showInclusionZone.R
showInclusionZoneGrid.R showStem.R showTract.R showSampSurf.R
summaryStem.R summaryStemContainer.R summaryArealSampling.R
summaryInclusionZone.R summaryIZContainer.R
summaryInclusionZoneGrid.R summaryTract.R summarySampSurf.R
setAsStem.R setAsizContainer.R bbox.R perimeter.R area.R
heapIZ.R gridCellEnhance.R plot3D.R clipStemsToTract.R
monteClass.R monteConstructors.R summaryMonte.R showMonte.R
histMonte.R horizontalLineSampling.R criticalHeightSampling.R
antitheticCHS.R taperInterpolate.R getID.R segmentVolume.R
MonteCarloContainer.R MonteCarloClass.R
MonteCarloConstructors.R MonteCarloProxy.R showMonteCarlo.R
summaryMonteCarlo.R plotMonteCarlo.R
horizontalPointMCSampling.R smithPlot.R

URL http://sampsurf.r-forge.r-project.org/

Repository CRAN

Repository/R-Forge/Project sampsurf

Repository/R-Forge/Revision 108

Repository/R-Forge/DateTimeStamp 2021-03-04 16:04:49

Date/Publication 2021-03-05 15:50:02 UTC

NeedsCompilation no

R topics documented:

`sampSurf-package` *Sampling Surface Simulation*

Description

The **sampSurf** package can be used to construct sampling surface simulations for different areal sampling methods common to forestry and ecology. These would include fixed-area plot methods, line intersect sampling, and various variable-radius plot sampling methods, the latter more commonly used in forestry. Most of these fall under the general probability proportional to size (PPS) umbrella and have some form of inclusion zone associated with each individual in the population. The inclusion zone has well-defined area (sometimes called the inclusion area) and may be thought of simply as that zone within which a sample point could fall and select the associated individual. The sample point can be the center of a circular fixed-area plot, an actual point as in horizontal point sampling, or say, the midpoint of a line in line intersect sampling.

In general we are interested in determining the properties of various sampling methods mentioned above when applied to fixed objects such as standing trees or down logs. The **sampSurf** package will allow generation of log or tree populations within a fixed tract area. The surface generated from the intersection of inclusion zones applied to the individuals in the population for a given attribute (e.g., cubic volume, number of individuals, etc.) are represented by the "`sampSurf`" class and can be displayed graphically. Estimator variance is directly associated with the sampling surface roughness, and so methods can also be compared visually.

sampSurf has support for several methods/protocols for sampling down logs; e.g.,. . .

- Fixed area plots, which includes the so-called ‘stand-up,’ ‘sausage,’ and ‘chainsaw’ protocols.
- Point relascope sampling (PRS).
- Perpendicular distance sampling (PDS) in several forms: Canonical PDS, omnibus PDS, canonical distance limited PDS (DLPDS), omnibus DLPDS, and hybrid DLPDS. Each of these PDS variants can be used with any one of three selection attributes: volume, coverage or surface areas.
- Distance limited sampling in two forms: Simple (DLS), and Monte Carlo (DLMCS).

Support for standing trees includes the following methods. . .

- Circular fixed area plots.
- Horizontal point (angle gauge) sampling (and tradiational Monte Carlo variants).
- Critical height sampling (also a Monte Carlo HPS variant) and importance sampling variants of CHS.
- Horizontal line (angle gauge) sampling.

Monte Carlo subsampling on down logs and standing trees is also available outside of the areal sampling paradigm. In addition, some areal sampling methods have special Monte Carlo variants, but the tradiational Monte Carlo subsampling framework now in place on stem objects means that these methods can be applied in conjunction with virtually any areal method. Horizontal point sampling is an example where both special and traditional Monte Carlo methods are available in the package.

Boundary (slopover) correction is available either through buffering, or the mirage method at present. The the kind of “Tract” that is used in the simulations will define which correction is desired.

The class structure is designed to support additions with relative ease, so more methods will appear in future releases.

Details

Package:	sampSurf
Type:	Package
Version:	0.7-6
Date:	4-Mar-2021
License:	GPL
LazyLoad:	yes

The package is written in S4 and has a number of classes and constructors, as well as some helper functions. In general, the capabilities include the following…

- Generate synthetic populations of logs or standing trees as "Stem" objects. Note that data from field measurements can also be used if available.
- Construct inclusion zones around each individual in the population and assign attributes to the zone for volume, etc.
- Generate "Tract" objects of given dimension (x,y) and resolution (grid cell size) to hold the objects. The grid cell resolution is related directly to sampling intensity: each grid cell center can be thought of as a sample point for the method under consideration.
- Combine the above into a sampling surface and calculate summary statistics, or display visually.
- Simulate multiple realizations of the same method with differing populations of objects to get more general results over many "plots" (i.e., "Tract" objects). (This is not implemented yet.)

Please note that there are several vignettes included within the package that explain in detail the classes and methods listed below using many examples. Please refer to the "Overview" vignette to get started. Then perhaps refer to the other vignettes in the order presented in the overview for detailed examples and descriptions. To find the vignettes on your help system, go to the index for the **sampSurf** package (see the link at the bottom of this page). The help files linked to below explain the various classes and methods in more detail, including slot and argument definitions.

In addition, the project homepage at R-Forge has some simple examples that might be helpful: `http://sampsurf.r-forge.r-project.org/`.

A companion package, **ssWavelets**, that supports wavelet filtering of sampling surfaces is found at: `http://sswavelets.r-forge.r-project.org/`.

Classes For Use In sampSurf…

The following S4 classes are defined within the **sampSurf** package to facilitate sampling surface generation. Note that it is not necessary to work with the class structures directly if all you want to do is generate a sampling surface for a population of synthetic logs or trees. However, some familiarization with the basic class structure is helpful in order to understand how the pieces fit together into a final sampling surface. Again, the vignettes are a source of much information.

The "Stem" class…:

Objects of the non-virtual classes below can be created using constructor functions of the same name, see the *Object Constructors* section for details.

Stem	Virtual base class for the following

downLog	A class for down logs
standingTree	A class for standing trees

The “StemContainer” class. . . :

Container classes hold collections of individual “Stem” subclass objects of the corresponding name. . .

StemContainer	Virtual base class for the following
downLogs	Container class for a population of down log objects
standingTrees	Container class for a population of standing tree objects

The “Tract” class. . . :

Tract	Land area holding the population of individuals
bufferedTract	A "Tract" subclass with a buffer
mirageTract	A "Tract" subclass with for the mirage method

The above classes are derived from the `Raster` class in the **raster** package. They conform to a rectangular area of ground such as a plot or any tract of land, tessellated into grid cells.

The “MonteCarloSampling” class. . . :

MonteCarloSampling	Virtual base class for Monte Carlo sampling
crudeMonteCarlo	Crude Monte Carlo sampling
importanceSampling	Importance subsampling
controlVariate	Control variate sampling
antitheticSampling	Antithetic sampling applied to any of the above

The above classes allow one to use the Monte Carlo methods mentioned to sample from individual "`Stem`" subclass objects. With these stand-alone classes, there is no areal context, sampling is done within individual stems.

Associated Container classes. . . :

mcsContainer	A collection of "`MonteCarloSampling`" objects
antitheticContainer	A collection of "`antitheticSampling`" objects

The “ArealSampling” class. . . :

These classes define the different sampling methods available and provide the method-based information for the subsequent construction of inclusion zones. . .

ArealSampling	A class structure for describing areal sampling methods
circularPlot	A subclass of "ArealSampling" for circular plot construction
pointRelascope	A subclass of "ArealSampling" for point relascope sampling

perpendicularDistance	A subclass of "ArealSampling" for variants of perpendicular distance sampling
distanceLimited	A subclass of "ArealSampling" for distance limited sampling
angleGauge	A subclass of "ArealSampling" for angle gauge (e.g., prism) sampling
lineSegment	A subclass of "ArealSampling" for line-oriented sampling methods

The "InclusionZone" class. . . :

InclusionZone	Virtual base class for combining the sampling method with the individual down log or tree obj
Classes for down logs. . .	
downLogIZ	Virtual subclass for all down log `InclusionZone` objects.
standUpIZ	Class for the "standup" method of sampling down logs.
sausageIZ	Class for the "sausage" method of sampling down logs.
chainSawIZ	Class for the "chainsaw" method of sampling down logs.
fullChainSawIZ	Class for the *full* "chainsaw" method of sampling down logs.
pointRelascopeIZ	Class for the point relascope method of sampling down logs.
perpendicularDistanceIZ	Class for the perpendicular distance method of sampling down logs.
omnibusPDSIZ	Class for the omnibus perpendicular distance method of sampling down logs.
distanceLimitedPDSIZ	Class for the distance limited perpendicular distance method of sampling down logs.
omnibusDLPDSIZ	Class for the omnibus distance limited perpendicular distance method of sampling down logs.
hybridDLPDSIZ	Class for the "hybrid" distance limited perpendicular distance method of sampling down logs.
distanceLimitedIZ	Class for the distance limited method of sampling down logs.
distanceLimitedMCIZ	Class for the distance limited Monte Carlo method of sampling down logs.
Classes for standing trees. . .	
standingTreeIZ	Virtual subclass for all standing tree `InclusionZone` objects.
circularPlotIZ	Class for the sampling standing trees using circulat plots.
horizontalPointIZ	Class for horizontal point sampling of standing trees.
horizontalPointCMCIZ	Class for crude Monte Carlo subsampling within horizontal point sampling of standing trees.
horizontalPointISIZ	Class for importance subsampling within horizontal point sampling of standing trees.
horizontalPointCVIZ	Class for control variate subsampling within horizontal point sampling of standing trees.
criticalHeightIZ	Class for critical height sampling of standing trees.
importanceCHSIZ	Class for importance critical height sampling of standing trees.
antitheticICHSIZ	Class for antithetic importance critical height sampling of standing trees.
pairedAICHSIZ	Class for paired antithetic importance critical height sampling of standing trees.
horizontalLineIZ	Class for horizontal line sampling of standing trees.
other. . .	
MonteCarloSamplingIZ	This class allows combining Monte Carlo with areal methods.

The "izContainer" class. . . :

The following are container classes that hold collections of objects of the corresponding type for inclusion zones generated based on down logs or standing trees and the desired sampling methods. Each holds a collection of objects that are a subclass of "downLogIZ" or "standingTreeIZ", respectively.

izContainer	Virtual base class for the following
downLogIZs	Holds inclusion zone objects associated with down logs
standingTreeIZs	Holds inclusion zone objects associated with standing trees

The “InclusionZoneGrid” class. . . :

InclusionZoneGrid	Not for general use, see documentation for details
csFullInclusionZoneGrid	Not for general use, see documentation for details
mirageInclusionZoneGrid	Not for general use, see documentation for details

The “sampSurf” class. . . :

sampSurf	The class structure for the final sampling surface simulation

Again, many of these classes will only be of interest to someone wanting to do special simulations or for someone who wants to add extensions–for example, new sampling methods.

Object Constructors

For each of the classes defined in the table above, we must be able to create objects that can be used in R. This is done using class-specific "constructor" methods that take the drudgery away from creating what can often be somewhat complicated (with all the graphical components) new object instances. There may be more than one constructor for a given class of object, and these are differentiated by the method signature; see the links provided below for more details. . .

“Stem” class constructors. . . :

downLog	Constructor for individual "downLog" objects
standingTree	Constructor for individual "standingTree" objects

The “StemContainer” class. . . :

downLogs	Constructs a container object holding multiple "downLog" objects
standingTrees	Constructs a container object holding multiple "standingTree" objects

“Tract” class constructors. . . :

Tract	Constructor for "Tract" objects
bufferedTract	Constructs a "bufferedTract" object
mirageTract	Constructs a "mirageTract" object

The “MonteCarloSampling” class constructors. . . :

crudeMonteCarlo	Apply Crude Monte Carlo sampling
importanceSampling	Apply importance subsampling
controlVariate	Apply control variate sampling
antitheticSampling	Apply antithetic sampling to any of the above

Associated Container class constructors...:

`mcsContainer`	A collection of "`MonteCarloSampling`" objects
`antitheticContainer`	A collection of "`antitheticSampling`" objects

"ArealSampling" class constructors...:

`circularPlot`	Constructs a circular plot object
`pointRelascope`	Constructs a point relascope object
`perpendicularDistance`	Constructs a perpendicular distance object
`distanceLimited`	Constructs a distance limited object
`angleGauge`	Constructs an angle gauge object
`lineSegment`	Constructs a line segment object

"InclusionZone" class constructors...:

Classes for down logs...	
`standUpIZ`	Creates an "InclusionZone" object for the 'stand-up' sampling method
`sausageIZ`	Creates an "InclusionZone" object for the 'sausage' sampling method
`chainSawIZ`	Creates an "InclusionZone" object for the 'chainsaw' sampling method
`fullChainSawIZ`	Creates an "InclusionZone" object for the *full* 'chainsaw' sampling method
`pointRelascopeIZ`	Creates an "InclusionZone" object for the 'point relascope' sampling method
`perpendicularDistanceIZ`	Creates an "InclusionZone" object for the 'perpendicular distance' sampling method
`omnibusPDSIZ`	Creates an "InclusionZone" object for the 'omnibus perpendicular distance' sampling method
`distanceLimitedPDSIZ`	Creates an "InclusionZone" object for the 'distance limited perpendicular distance' sampling m
`omnibusDLPDSIZ`	Creates an "InclusionZone" object for the 'omnibus distance limited perpendicular distance' sa
`hybridDLPDSIZ`	Creates an "InclusionZone" object for the 'hybrid distance limited perpendicular distance' sam
`distanceLimitedIZ`	Creates an "InclusionZone" object for the 'distance limited' sampling method
`distanceLimitedMCIZ`	Creates an "InclusionZone" object for the 'distance limited Monte Carlo' sampling method
Classes for standing trees...	
`circularPlotIZ`	Creates an "InclusionZone" object for the fixed-area circular plot sampling method
`horizontalPointIZ`	Creates an "InclusionZone" object for the horizontal point (prism) sampling method
`horizontalPointCMCIZ`	Creates an "InclusionZone" object for HPS with crude Monte Carlo subsampling
`horizontalPointISIZ`	Creates an "InclusionZone" object for HPS with importance subsampling
`horizontalPointCVIZ`	Creates an "InclusionZone" object for HPS with control variate subsampling
`criticalHeightIZ`	Creates an "InclusionZone" object for the critical height (CH) sampling method
`importanceCHSIZ`	Creates an "InclusionZone" object for the importance critical height sampling method
`antitheticICHSIZ`	Creates an "InclusionZone" object for the antithetic importance CH sampling method
`pairedAICHSIZ`	Creates an "InclusionZone" object for the paired antithetic importance CH sampling method
`horizontalLineIZ`	Creates an "InclusionZone" object for the horizontal line sampling method

The "izContainer" class constructors...:

`downLogIZs`	Creates a collection of "downLogIZ" objects for down logs
`standingTreeIZs`	Creates a collection of "standingTreeIZ" objects for standing trees

“InclusionZoneGrid” class constructors. . . :

izGrid	Creates an "InclusionZoneGrid" or subclass object (i.e., "csFullInclusionZoneGrid")
izGridMirage	Creates an "mirageInclusionZoneGrid" object.

“sampSurf” class constructors. . . :

sampSurf	Constructor for "sampSurf" objects

Summary and plotting methods

Almost without exception, the objects created above have graphical content made possible by using classes from the sp or raster packages. Therefore, methods have been added to the plot generics to allow for graphical display of objects. In addition, summary and show methods are also available (sometimes producing the same result) for printing information within the object. . .

hist	Create a histogram of object attributes
plot	To plot one of the package objects
plot3D	To plot "sampSurf" objects using package ’rgl’
show	To succinctly print one of the package objects
summary	To summarize one of the package objects
smithPlot	Generate H. F. Smith plots for simulation comparisions.

Helper methods

Slots in S4 objects can be accessed directly using slot method or the @ operator. However, some "helper" functions have been provided for certain objects, along with other potentially useful routines. . .

area	Returns the area of a spatial object
bbox	Return the bounding box for a spatial object
bboxCheck	Check for a valid bounding box for a spatial object
bboxToPoly	Convert a bbox matrix to a "SpatialPolygons" object
bboxSum	Calculate an overall bbox from and array of bbox matrices
getID	Returns the IDs associated with objects
getProxy	Return the desired proxy function for MC sampling
perimeter	Returns the perimeter of a spatial object, which can then be plotted
spCircle	Returns a circular "SpatialPolygons" object

Coercion methods

Object coercion is accomplished in S4 by a call to as with the appropriate object and class to which one wants to convert. . .

as	Convert from a "downLogs" or "standingTrees" object to a data frame
as	Convert "downLogIZs" or "standingTreeIZs" object to respective "StemContainer"

The resulting data frame in the first method will be compatible with those generated from calls to either “sampleLogs” or “sampleTrees”. The second takes the drudgery out of getting a log collection from, say, the `izContainer` slot of a `sampSurf` object.

Miscellaneous methods

Some methods that might be useful as stand-alone routines. . .

`boltDimensions`	Calculates segment volumes, surface area, etc.
`checkStemDimensions`	Checks for inconsistencies in attribute slots.
`clipStemsToTract`	Clips or removes stems outside the tract
`gridCellEnhance`	Draws grid lines and centers.
`heapIZ`	Used to heap inclusion zone grids into a sampling surface.
`initRandomSeed`	Initialize `.Random.seed`.
`sampleLogs`	Draws a simulated sample of down logs and returns a data frame.
`sampleTrees`	Draws a simulated sample of standing trees and returns a data frame.
`segmentVolume`	Calculates segment volumes for down logs or standing trees.
`taperInterpolate`	Interpolates diameters or lengths within down logs or standing trees.
`transparentColorBase`	Transparancy in base graphics.

‘sampSurf’ environments

This package uses a “hidden” environment to keep things that are not necessary for the user out of mind. There is little reason to peak into it, except that some function argument defaults are specified directly via this environment. Thus, to enable you to figure out better what is going on, the environment is exported and its members can be accessed if desired.

`.StemEnv`	This environment holds parameters for much of the package.

‘sampSurf’ class unions

There are a few class unions that are documented here only for the sake of completeness. It is unlikely that the casual user would ever need to be concerned about these.

`dlsIZNull`	Accepts either class "distanceLimitedIZ" (or subclass) or "NULL".
`dlsNumeric`	Accepts either class "distanceLimited" or "numeric".
`monteBSSampleOrNULL`	Accepts either class "monteBSSample" or "NULL".
`monteNTSampleOrNULL`	Accepts either class "monteNTSample" or "NULL".
`numericNULL`	Accepts either class "numeric" or "NULL".
`pdsIZNull`	Accepts either class "perpendicularDistanceIZ" (or subclass) or "NULL".
`horizontalPointMonteCarloSamplingIZ`	Accepts any of the MC and horizontal point sampling combinations.
`SPNULL`	Accepts either class "SpatialPolygons" or "NULL".
`RLNULL`	Accepts either class "RasterLayer" or "NULL".
`izgNULL`	Accepts either class "InclusionZoneGrid" or "NULL".

"monte": When is *n* Sufficiently Large?

This section has both classes and methods defined for doing Monte Carlo analysis of the convergence of confidence interval catch rates as sample size increases.

`montePop`	Population class with constructor `montePop`.
`monteSample`	Virtual repeated sampling class with the following two subclasses.
`monteNTSample`	Normal theory repeated sampling class with constructor `monteNTSample`.
`monteBSSample`	Bootstrap repeated sampling class with constructor `monteBSSample`.
`monte`	Main Monte Carlo class with constructor `monte`.

Please see the "'monte": When is *n* Sufficiently Large?' vignette for more details and examples.

Author(s)

Jeffrey H. Gove

References

The first two references describe the sampling surface method in detail. The rest provide some examples of its use, or document some of the methods in the package, and are not exhaustive.

Williams, M. S. 2001. New approach to areal sampling in ecological surveys. *Forest Ecology and Management* **154**:11–22.

Williams, M. S. 2001. Nonuniform random sampling: an alternative method of variance reduction for forest surveys. *Canadian Journal of Forest Research* **31**:2080–2088.

Williams, M. S. and Gove, J. H. 2003. Perpendicular distance sampling: an alternative method for sampling downed coarse woody debris. *Canadian Journal of Forest Research* **33**:1564–1579.

Gove, J. H., Williams, M. S., Stahl, G., and Ducey, M. J. 2005. Critical point relascope sampling for unbiased volume estimation of downed coarse woody debris. *Forestry* **78**:417–431.

Stahl, G., Gove, J. H., Williams, M. S., and Ducey, M. J. 2010. Critical length sampling: a method to estimate the volume of downed coarse woody debris. *European Journal of Forest Research* **129**:993–1000.

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

Gove, J. H., Ducey, M. J. and Valentine, H. T. 2012. A distance limited method for sampling downed coarse woody debris. *Forest Ecology and Management* **282**:53–62.

Ducey, M. J., Williams, M. S., Gove, J. H., Roberge, S. and Kenning, R. S. 2013. Distance limited perpendicular distance sampling for coarse woody material: Theory and field results. *Forestry* **86**:119–128

Gove, J. H., Ducey, M. J., Valentine, H. T. and Williams, M. S. 2013. A comprehensive comparison of perpendicular distance sampling methods for sampling down coarse woody debris. *Forestry* **86**:129–143.

Lynch,T. B. and Gove, J. H. 2013. An antithetic variate to facilitate upper-stem height measurements for critical height sampling with importance sampling. *Canadian Journal of Forest Research* **43**:1151–1161.

Lynch,T. B. and Gove, J. H. 2014. The unbiasedness of a generalized mirage boundary correction method for Monte Carlo integration estimators of volume. *Canadian Journal of Forest Research* **44**:810–819.

Gove, J. H. 2017. Some Refinements on the Comparison of Areal Sampling Methods via Simulation. *Forests* **8,393**:1–24.

See Also

The package makes extensive use of existing GIS packages within R. Please see the `sp` and `raster` packages for details on the underlying grid and polygon classes and methods.

`.StemEnv` *'Hidden' Environment for Constants, etc.*

Description

This environment is used throughout the **sampSurf** package. Even though it is hidden, it is simply an environment to hold 'global' variables/parameters and functions in a central, but "out of the way" place. It is exported from the **sampSurf** namespace, so you can access any objects within it, though most of the contents will not be of interest to the majority of users. Also note that it has been locked for safety, so one can not add or change its contents.

Please see the code itself for information on the contents of this environment and the purpose of each object it contains.

Author(s)

Jeffrey H. Gove

Examples

```
require(sampSurf)
ls(.StemEnv)
.StemEnv$sampleLogsNames
get('wbTaper', envir=.StemEnv)
```

angleGauge *Generate Objects of Class "angleGauge"*

Description

This generic function has only one method used as a constructor function for objects that are of class "angleGauge". This method should be used in preference to new to insure a valid object.

Usage

```
angleGauge(baf, ...)
```

Arguments

baf	The basal area factor of the angle gauge (prism) in the appropriate units.
...	See the methods for other arguments.

Details

Only one method currently exists for object generation. Its arguments are documented in angleGauge-methods.

Value

A valid object of class "angleGauge"

Author(s)

Jeffrey H. Gove

See Also

angleGauge-methods and the "ArealSampling" class.

Examples

```
#
#  create an object with an English 10 baf angle gauge...
#
ag = angleGauge(10, units='English')
ag
```

angleGauge-class *Class* "angleGauge"*: For Angle Gauge Sampling Methods*

Description

A subclass of "ArealSampling" that can be used to create objects that encapsulate all the parameters necessary for, e.g., horizontal point (prism) sampling of standing trees (or downed logs).

Objects from the Class

Objects can be created by calls of the form `new("angleGauge",...)`, and while this is reasonable with this class because there are no graphical slots, it is still not recommended. The preferred method for creating new objects is via the `angleGauge` constructor.

Slots

In addition to those slots in class "ArealSampling," the following are defined...

`angleDegrees`: Object of class `"numeric"`: The gauge angle in degrees, normally in the range 0 < `angleDegrees` <= 6.5 degrees.

`angleRadians`: Object of class `"numeric"`: The gauge angle in radians.

`diopters`: Object of class `"numeric"`: The prism angle in diopters (Delta).

`k`: Object of class `"numeric"`: Angle gauge constant (dimensionless); i.e., $2 \times \sin(\frac{\eta}{2})$, where η is `angleRadians` above.

`prf`: Object of class `"numeric"`: The plot radius factor; English=ft/in, metric=m/cm.

`PRF`: Object of class `"numeric"` The plot radius factor; English=ft/ft, metric=m/m.

`alpha`: Object of class `"numeric"` The plot radius proportionality constant; English=ft/ft, metric=m/m.

`baf`: Object of class `"numeric"`: The corresponding basal area factor in the appropriate units; English=ft^2/acre, metric=m^2/ha.

`df`: Object of class `"numeric"`: Diameter factor for horizontal line sampling in inches or cm.

`DF`: Object of class `"numeric"`: Diameter factor for horizontal line sampling in feet or meters.

Extends

Class "ArealSampling", directly.

Methods

horizontalPointIZ `signature(standingTree = "standingTree",angleGauge = "angleGauge")`: Object constructor for horizontal point sampling.

summary `signature(object = "angleGauge")`: Show a summary of the `object`.

Author(s)

Jeffrey H. Gove

References

T. G. Gregroire and H. T. Valentine. 2009. *Sampling strategies for natural resources and the environment*. Chapman and Hall/CRC, 496p.

See Also

The "`ArealSampling`" class.

Examples

```
showClass("angleGauge")
```

`angleGauge-methods` *Methods* `"angleGauge"` *Object Construction in Package* **sampSurf**

Description

There is currently only one method based on the `angleGauge` generic that is used for object construction. It is detailed below.

Methods

`signature(baf = "numeric")` This method takes the basal area factor as the key argument. . .

usage. . .

```
angleGauge(baf,
           units = 'metric',
           description = 'angle gauge method',
           ...)
```

- `baf`: The basal area factor.
- `units`: Either “English” or “metric”.
- `description`: A character vector description of the object.

`antitheticContainer` *Generate Objects of Class "`antitheticContainer`"*

Description

The methods for this function will allow the creation of valid "`antitheticContainer`" class objects. Please note that it is preferable to use the `antitheticSampling` constructor method directly with an "`mcsContainer`" object to create a valid collection. Please see the vignette below for detailed examples. See also `antitheticContainer-methods` for method details.

Usage

```
antitheticContainer(object, ...)
```

Arguments

`object`	The signature object for the generic.
`...`	Just gobbled presently.

Details

The vignette below gives examples on creating "antitheticContainer" objects for each of the Monte Carlo subsampling methods available in **sampSurf** by passing a "mcsContainer" object to the `antitheticSampling` constructor. Please use this rather than calling the current constructor directly.

Value

A valid object of class "`antitheticContainer`."

Note

Please note that this is not a completely functional container class in the traditional sense at present as it does not have replacement, deletion, or addition functions. If you need to do any of these operations, perform them on the `list` object (in the `mcsObjs` slot) and then recreate the container. If the object is not re-built after, e.g., deletion, the summary statistics will be incorrect.

Author(s)

Jeffrey H. Gove

References

Gove, J. H. 2013. Monte Carlo sampling methods in **sampSurf**. Package vignette.

See Also

`antitheticSampling`

Examples

```
sTrees = standingTrees(5, startSeed = 12)
sTrees.cmc = crudeMonteCarlo(sTrees, n.s = 10)
sTrees.anti = antitheticSampling(sTrees.cmc)
sTrees.anti
print(sTrees.anti@stats, digits = 4) #compare with cmc below
print(sTrees.cmc@stats, digits = 4)
```

---

antitheticContainer-class

Class "antitheticContainer"

Description

This class allows one to store a collection of objects that are all of class "antitheticSampling."

Objects from the Class

Objects can be created by calls of the form `new("antitheticContainer",...)`. However, the actual constructor method (antitheticContainer) need never be called directly; instead, use the antitheticSampling constructor to create container objects of this class.

Slots

`mcsObjs`: Object of class `"list"`: A list of objects that must be of class "antitheticSampling."

`stats`: Object of class `"matrix"`: Summary statistics for the objects in slot `mcsObjs`.

`description`: Object of class `"character"` A character description of the contents if desired.

Extends

Class "mcsContainer", directly.

Methods

No methods defined with class "antitheticContainer" in the signature. However, `plot`, `show`, `summary` and `hist` methods will work via inheritance.

Note

Please note that this is not a completely functional container class in the traditional sense at present as it does not have replacement, deletion, or addition functions. If you need to do any of these operations, perform them on the `list` object (in the `mcsObjs` slot) and then recreate the container. If the object is not re-built after, e.g., deletion, the summary statistics will be incorrect.

Author(s)

Jeffrey H. Gove

References

Gove, J. H. 2013. Monte Carlo sampling methods in **sampSurf**. Package vignette.

See Also

`antitheticContainer`, `antitheticSampling`, `mcsContainer`.

Examples

```
showClass("antitheticContainer")
```

`antitheticContainer-methods` *Methods for* `"antitheticContainer"` *object creation in Package* **sampSurf**

Description

There is only one method for the `antitheticContainer` generic function. Please note that this method should not be called directly. Instead, one should use the `antitheticSampling` constructor by passing it an object of class "`mcsContainer`" containing a collection of objects where importance, crude Monte Carlo or control variate sampling have been applied.

Methods

`signature(object = "list")`

usage...

```
antitheticContainer(object,
                    description = 'Antithetic Sampling container object',
                    ... )
```

- `object`: An object of class "`list`".

`antitheticICHSIZ` *Generate Objects of Class* "`antitheticICHSIZ`"

Description

This is the generic function for class “antitheticICHSIZ”. Please see the associated method in `antitheticICHSIZ-methods` for more details.

Usage

```
antitheticICHSIZ(standingTree, angleGauge, ...)
```

Arguments

`standingTree`	Signature object of class "`standingTree`".
`angleGauge`	Signature object of class "`angleGauge`".
`...`	See methods.

Details

Since only one method exists for this generic, its signature arguments coincide with the above. Please see `antitheticICHSIZ-methods` for more details.

Value

A valid object of class "`antitheticICHSIZ`."

Author(s)

Jeffrey H. Gove

See Also

Class "`antitheticICHSIZ`", and `antitheticICHSIZ-methods`.

Examples

```
st = standingTree(dbh=50, solidType=4, height=25)
ag = angleGauge(baf=4)
iz.aichs = antitheticICHSIZ(st, ag)
summary(iz.aichs)
plot(iz.aichs, axes=TRUE, cex=2)
```

`antitheticICHSIZ-class`

Class "antitheticICHSIZ"

Description

This class holds the inclusion zone definition for the antithetic importance sampling variant/protocol of critical height sampling for standing trees.

Objects from the Class

Objects can be created by calls of the form `new("antitheticICHSIZ",...)`. However, this is not recommended because the objects are fairly complex. Instead, one can use the object constructor "`antitheticICHSIZ`" to create new objects.

Slots

This class is a subclass of "importanceCHSIZ" (see below). It adds no new slots to the definition of that class, so please refer to it for the slot definitions.

Extends

Class "`importanceCHSIZ`", directly.
Class "`criticalHeightIZ`", by class "importanceCHSIZ", distance 2.
Class "`horizontalPointIZ`", by class "importanceCHSIZ", distance 3.
Class "`circularPlotIZ`", by class "importanceCHSIZ", distance 4.
Class "`standingTreeIZ`", by class "importanceCHSIZ", distance 5.
Class "`InclusionZone`", by class "importanceCHSIZ", distance 6.

Methods

izGrid `signature(izObject = "antitheticICHSIZ",tract = "Tract")`: "InclusionZoneGrid" constructor

Author(s)

Jeffrey H. Gove

References

T. B. Lynch and J. H. Gove. 2013. An antithetic variate to facilitate upper-stem height measurements for critical height sampling and fixed-radius plot sampling with importance sampling. *Canadian Journal of Forest Research* (forthcoming).

"*The InclusionZone Class*" vignette.

See Also

See also the "`circularPlotIZ`", "`horizontalPointIZ`", "`criticalHeightIZ`", "`importanceCHSIZ`" and "`pairedAICHSIZ`" classes, and the "`standingTreeIZs`" container class

Examples

```
showClass("antitheticICHSIZ")
```

`antitheticICHSIZ-methods` *Methods for* `"antitheticICHSIZ"` *object construction in Package* **sampSurf**

Description

This documents the one method for generic function `antitheticICHSIZ` in Package **sampSurf** that allows for creation of objects of class "`antitheticICHSIZ`."

Methods

`signature(standingTree = "standingTree", angleGauge = "angleGauge")`

usage. . .

```
antitheticICHSIZ(standingTree,
                 angleGauge,
                 referenceHeight = .StemEnv$referenceCHSIZ,
           description = 'inclusion zone for antithetic ICH sampling method',
                 spID = paste('aichs',.StemEnv$randomID(),sep=':'),
                 spUnits = CRS(projargs=as.character(NA)),
                 ...)
```

- `standingTree`: An object of class "`standingTree`" for which the inclusion zone is to be determined.
- `angleGauge`: An object of class "`angleGauge`".
- `referenceHeight`: The height on the stem at which the inclusion zone is to be determined. Currently the choices are “butt” (default) or “dbh”. These are found in the argument assignment `.StemEnv$referenceCHSIZ` above.
- `description`: A character vector description of the object.
- `spID`: A unique identifier that will be used in displaying the spatial polygon for the inclusion zone component of the object.
- `spUnits`: A valid `CRS` object specifying the Coordinate Reference System. This defaults to `NA`, which means you want to use your own user-defined system, say for a sample plot located in the field.
- `dots`: Arguments to be passed on.

`antitheticSampling` *Generate Objects of Class "*`antitheticSampling`*"*

Description

This generic has two methods, they are used to apply antithetic sampling to an individual `"MonteCarloSampling"` subclass object, or collections of `"MonteCarloSampling"` objects. See `antitheticSampling-methods` for details.

Usage

```
antitheticSampling(object, ...)
```

Arguments

`object`	An object that is a subclass of "`MonteCarloSampling`".
`...`	Just gobbled at present.

Details

This method samples points on the bole antithetic to those found in the `object` argument. Note that one can *not* pass arguments on to the proxy. All of the information from the `object` passed is used to determine the proxy and any specific arguments used by the proxy in the creation of that object. The constructor functions do not allow any of this to be changed by the user.

Value

A valid object of class "`antitheticSampling`" or "`antitheticContainer`", depending on which method was used.

Author(s)

Jeffrey H. Gove

References

Gove, J. H. 2013. Monte Carlo sampling methods in **sampSurf**. Package vignette.

See Also

See `antitheticSampling-methods` for methods. Generics for Monte Carlo methods for the `object` are: `crudeMonteCarlo` `importanceSampling`, `controlVariate`.

Examples

```
#
# estimate volume between 10 and 15 m, using 5 random heights...
#
sTree = standingTree(dbh = 40, topDiam = 0, height = 20, solidType = 2.8)
sTree.cmc = crudeMonteCarlo(sTree, n.s = 5, segBnds = c(10,15), startSeed = 114)
sTree.anti = antitheticSampling(sTree.cmc)
sTree.anti
```

`antitheticSampling-class`
Class `"antitheticSampling"`

Description

This class definition allows for the application of antithetic sampling to any "`MonteCarloSampling`" subclass object. Please see the vignette reference below for more details.

Objects from the Class

Objects can be created by calls of the form `new("antitheticSampling",...)`. However, an object constructor of the same name, `antitheticSampling`, has been provided and is the preferred method for creating objects that are ensured to be valid.

Slots

`mcsObj`**:** Object of class `"MonteCarloSampling"`: This is the original `"MonteCarloSampling"` subclass object based on the `u.s` random numbers found within the slot of this name within the object.

`mcsAnti`**:** Object of class `"MonteCarloSampling"`: This is the antithetic companion object to `mcsObj` that has been derived from the `1-u.s` random numbers.

`volEst`**:** Object of class `"numeric"`: The sample mean volume estimate for the bole segment.

`volVar`**:** Object of class `"numeric"`: The within bole variance estimate of `volEst`.

`ci.lo`**:** Object of class `"numeric"`: The lower 1-`alphaLevel` confidence interval on the bole volume estimate.

`ci.up`**:** Object of class `"numeric"`: The upper 1-`alphaLevel` confidence interval on the bole volume estimate.

`alphaLevel`**:** Object of class `"numeric"`: The two-tailed alpha-level for confidence interval construction.

`trueVol`**:** Object of class `"numeric"`: The true volume for the stem segment being estimated (see `segBnds` in the `"MonteCarloSampling"` class definition).

`relErrPct`**:** Object of class `"numeric"`: The relative error in volume in percent.

`description`**:** Object of class `"character"`: A description of the object if desired (defaults are given for each class).

Methods

show `signature(object = "antitheticSampling")`: print a summary of the object.

summary `signature(object = "antitheticSampling")`: print a summary of the object.

Author(s)

Jeffrey H. Gove

References

Gove, J. H. 2013. Monte Carlo sampling methods in **sampSurf**. Package vignette.

See Also

The following subclasses and related via the `mcsObj` slot: `MonteCarloSampling`, `crudeMonteCarlo`, `importanceSampling`, `controlVariate`,

Examples

```
showClass("antitheticSampling")
```

`antitheticSampling-methods`
Methods for `antitheticSampling` *object construction in Package* **sampSurf**

Description

The methods below allow for the construction of individual objects of class "`antitheticSampling`" when applied to individual "`MonteCarloSampling`" subclass objects. Alternatively, the second constructor can be applied to a collection of "`MonteCarloSampling`" subclass objects and will return an appropriate collection of "`antitheticSampling`" objects (see, e.g., the "`antitheticContainer`" class).

Methods

`signature(object = "MonteCarloSampling")`

usage...

```
antitheticSampling(object,
                   alphaLevel = 0.05,
                   description = 'Antithetic Sampling',
                   ... )
```

- `object`: An object that is a subclass of "`MonteCarloSampling`".
- `alphaLevel`: The two-tailed alpha-level for confidence interval construction.
- `description`: A character vector description of the object.

`signature(object = "mcsContainer")` See the above definitions for the first method for arguments not shown below.

usage...

```
antitheticSampling(object,
                   alphaLevel = 0.05,
                   description = 'Antithetic Sampling collection',
                   ... )
```

- `object`: A collection of "`MonteCarloSampling`" subclass objects in a valid "`mcsContainer`" object.

`area` *Area of objects in package* **sampSurf**

Description

A number of the different classes of objects in **sampSurf** have some kind of areal representation, often just part of the class definition as one of the slots. This function simply returns the area as stored within the object as given in `area-methods`.

Details

The area that is stored and returned is in the native units of the object. In most cases, this will be either square feet or square meters. The area-methods provide individual detail, as do the individual class definitions for the objects passed.

Value

Area in the correct units.

Author(s)

Jeffrey H. Gove

See Also

area-methods

Examples

```
tract = Tract(c(x=20,y=35),cellSize=0.25, units='metric')
area(tract)/.StemEnv$smpHectare  #in hectares
```

---

area-methods *Methods for Function* area *in Package* **sampSurf**

Description

The methods described here are for the area generic as used in the **sampSurf** package. Each of the methods shown below has different arguments that customize it for a different class of objects found in this package.

Methods

All of the methods have the same argument form, so details will only be given for the "InclusionZone" class version. The other available method signatures are simply listed.

`signature(x = "InclusionZone")`

usage...

```
area(x, ... )
```

- x: An object that is a subclass of "InclusionZone".
- ...: Currently not used—just gobbled by R.

`signature(x = "InclusionZoneGrid")`

`signature(x = "Tract")`

`ArealSampling-class` *Class "ArealSampling"*

Description

This virtual class facilitates the definition of subclasses that will encapsulate the basic information required to define individual areal sampling methods used in forestry.

For example, as detailed in “The ArealSampling Class” vignette, we may define subclasses based on whether the sampling methods are for standing trees or downed logs. Other classes, like fixed-area plots, could span these two main subclasses.

Objects from the Class

A virtual Class: No objects may be created from it.

Slots

`description`**:** Object of class `"character"`: Some descriptive text about this class.

`units`**:** Object of class `"character"`: A character string specifying the units of measure. Legal values are “English” and “metric.”

Methods

plot `signature(x = "ArealSampling",y = "missing")`: Graphical display of the object. Minimal display is the object `location`.

show `signature(object = "ArealSampling")`: Show main components or summary of an object.

summary `signature(object = "ArealSampling")`: Concise summary of the object.

Author(s)

Jeffrey H. Gove

References

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

See Also

"`circularPlot`" for a circular plot subclass.

"`pointRelascope`" for the point relascope sampling subclass.

"`perpendicularDistance`" for the perpendicular distance sampling subclass.

"`distanceLimited`" for the distance limited sampling subclass.

"`angleGauge`" for horizontal point sampling subclass.

"`lineSegment`" for the line-based subclass.

Examples

```
showClass("ArealSampling")
```

`bboxCheck` *Function to Check Spatial Bounding Boxes*

Description

A `bbox` is not a formal class, but is used often enough in package **sampSurf** that a little routine to check for a valid one is useful. It could have been made a class, but it was not in the more basic **sp** and **raster** packages, and so was not introduced here.

Usage

```
bboxCheck(bbox)
```

Arguments

`bbox`	A matrix to be checked.

Details

A valid `bbox` matrix is `2x2`, all numeric, and has row names `c('x','y')` and column names `c('min','max')` in package **sampSurf**. A check is also made for minima-maxima extents in the correct sense.

Value

The function returns true if the matrix passes all the checks, otherwise it stops with an error.

Author(s)

Jeffrey H. Gove

See Also

`bbox`

Examples

```
tract = Tract(c(x=20,y=20), cellSize=0.5)
bb = bbox(tract)
bboxCheck(bb)
```

---

bboxSum *Function to Compute an Overall Bounding Box (bbox)*

Description

Takes an array of bbox matrices and forms the overall encapsulating bounding box from these.

Usage

```
bboxSum(arr.bbox, ...)
```

Arguments

arr.bbox	This is a 3 dimensional array of bbox matrices with the rows and columns of each matrix as the first two dimensions, and each matrix indexed by the 3rd dimension of the array. Note that each bbox will be checked for validity such that row names must be c("x","y") and column names must be c("min","max"). Usually it is most convenient to use the bbox method for retrieving valid individual bbox matrices to put into the array.
...	Not presently used.

Details

The details are all presented above.

Value

A valid bbox matrix for the minimal bounding box of the set.

Author(s)

Jeffrey H. Gove

See Also

bbox, bboxToPoly, bboxCheck

Examples

```
dlogs = downLogs(3)
bboxArray = array(dim=c(2,2,3))
bboxArray[,,1] = bbox(perimeter(dlogs@logs$log.1))
bboxArray[,,2] = bbox(perimeter(dlogs@logs$log.2))
bboxArray[,,3] = bbox(perimeter(dlogs@logs$log.3))
dimnames(bboxArray) = dimnames(bbox(dlogs@logs$log.1)) #page dim doesn't matter
bbox = bboxSum(bboxArray)
## Not run:
plot(dlogs, axes=TRUE)
plot(bboxToPoly(bbox), add=TRUE)

## End(Not run)
```

---

bboxToPoly *Function to Convert a Valid "bbox" to a "SpatialPolygons" Object*

Description

This converts a valid bbox, or object with a bbox specification in the **sampSurf** package, if available, into a spatial polygon object of class "`SpatialPolygons`".

Usage

```
bboxToPoly(object, ...)
```

Arguments

`object`	Some object with a valid bbox slot from one of the classes in **sampSurf**, or a valid bbox matrix.
`...`	Black hole for everything else.

Details

One can plot the result of this function to quickly get an idea of the overall spatial extent of an object. In many cases the result from this object is the same as that returned by `perimeter`, but not always; e.g., see the example below.

Value

A "`SpatialPolygons`" object that can be plotted directly.

Author(s)

Jeffrey H. Gove

Examples

```
cp = circularPlot(10, center=c(x=20,y=20))
cp.perim = perimeter(cp)
plot(cp.perim, axes=TRUE, lty='dashed')             #the perimeter of the plot
plot(bboxToPoly(cp.perim), add=TRUE, border='blue')  #the bounding box
```

`boltDimensions` *Calculate Bolt Dimensions for “downLog” Objects*

Description

This function will calculated volume, length, surface area, etc. for each bolt in a “downLog” object, where the bolts are defined by the `taper` slot.

Usage

```
boltDimensions(dlog, runQuiet = FALSE)
```

Arguments

`dlog`	An object of class "`downLog`"
`runQuiet`	`TRUE`: no feedback; `FALSE`: print a summary report

Details

Nothing is stored in a “downLog” object with respect to the individual bolts except the taper measurements in the `taper` slot. This allows the easy computation of different attributes for each bolt as described above.

Value

A data frame with self-evident columns, and rows representing bolts. For example, cross-check the diameters and lengths with the log’s `taper` slot. The "biomass" and "carbon" columns will be `NA` if no conversions have been given for the log at creation.

Author(s)

Jeffrey H. Gove

See Also

"`downLog`"

Examples

```
dl = downLog(logLen=5, nSegs=10)
dl@taper
(boltDimensions(dl))
```

bufferedTract *Generate Objects of Class "bufferedTract"*

Description

This creates a "bufferedTract" object with an internal buffer. There is only one method for object construction at present.

Usage

```
bufferedTract(bufferWidth, tract, ...)
```

Arguments

bufferWidth	Signature argument: buffer width.
tract	Signature argument: a "Tract" object.
...	See methods for details.

Details

This generic is defined completely with respect to the signature of its method in bufferedTract-methods, which should be consulted for details.

Value

A valid object of class "bufferedTract"

Author(s)

Jeffrey H. Gove

See Also

bufferedTract-methods, Tract-methods, Tract

Examples

```
tract = Tract(c(x=20,y=20), cellSize=0.25)
buffTr = bufferedTract(4, tract)
summary(buffTr)
buffTr
plot(buffTr, axes=TRUE)
```

`bufferedTract-class` *Class "bufferedTract"*

Description

This class is a subclass of the `"Tract"` class. It provides, in addition to the functionality in "Tract", the ability to create an internal buffer within the tract. This might be important, for example, when drawning a population of logs for sampling surface generation. The buffer width can be set such that no log's inclusion zone extends outside the tract. Objects of this class are also descendants of class `"RasterLayer"`, and so share the full functionality of this class.

Objects from the Class

Objects can be created by calls of the form `new("bufferedTract",...)`. However, this is not recommended due to the complexity of the class. Instead, use the `bufferedTract` constructor to create objects.

Slots

In addition to those slots that are defined within the `"Tract"`, the following are available...

`bufferRect`**:** Object of class `"matrix"`: A matrix in the form of a bounding box with row names `c("x","y")` and column names `c("min","max")`.

`spBuffer`**:** Object of class `"SpatialPolygons"`: A `SpatialPolygons` representation of `bufferRect` via the `sp` package.

Extends

Class `"Tract"`, directly. Class `"RasterLayer"`, by class "Tract", distance 2. Class `"Raster"`, by class "Tract", distance 3. Class `"BasicRaster"`, by class "Tract", distance 4.

Methods

downLogs `signature(object = "numeric",container = "bufferedTract")`: Create downlogs

plot `signature(x = "bufferedTract",y = "missing")`: Plot the logs

show `signature(object = "bufferedTract")`: Show the object

Author(s)

Jeffrey H. Gove

See Also

`Tract`, `mirageTract`

Examples

```
showClass("bufferedTract")
```

bufferedTract-methods *Methods for "bufferedTract" Object Construction in package "sampSurf"*

Description

There is only one existing method available for the `bufferedTract` generic function, which will generate valid objects of class "`bufferedTract`".

Methods

`signature(bufferWidth = "numeric", tract = "Tract")` This method takes the form shown below. . .

usage. . .

```
bufferedTract(bufferWidth,
              tract,
              runQuiet = FALSE,
              ... )
```

- `bufferWidth`: The width of the buffer in the correct units. For "English" units, this would be feet, for "metric," meters. The buffer must be smaller than the tract extents.
- `tract`: A valid tract object to which we will add the internal buffer.

chainSawIZ *Generate Objects of Class "`chainSawIZ`"*

Description

This is the generic definition for generating objects of class "chainSawIZ." There is only one method corresponding to this generic: `chainSawIZ-methods`.

Usage

```
chainSawIZ(downLog, plotRadius, ...)
```

Arguments

`downLog`	Signature object of class "`downLog`".
`plotRadius`	Signature object for plot radius.
`...`	See methods.

Details

Since only one method exists for this generic, its signature arguments coincide with the above definitions. Please see `chainSawIZ-methods` for more details.

Value

A valid object of class "chainSawIZ."

Author(s)

Jeffrey H. Gove

References

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

See Also

Class "chainSawIZ", and chainSawIZ-methods.

Examples

```
## Not run:
dl = downLog(buttDiam=40, solidType=4, logAngle=4*pi/3, logLen=6)
iz.cs = chainSawIZ(dl, plotRadius=3, plotCenter=c(x=2,y=2), runQuiet=FALSE)
summary(iz.cs)
plot(iz.cs, axes=TRUE, showPlotCenter=TRUE, cex=2, showNeedle=TRUE)

## End(Not run)
```

`chainSawIZ-class` *Class "chainSawIZ"*

Description

This class holds the inclusion zone definition for the 'chainsaw' method (Gove and Van Deusen, 2011) for sampling down coarse woody debris.

Objects from the Class

Objects can be created by calls of the form `new("chainSawIZ",...)`. However, this is not recommended because the objects are fairly complex. Instead, one can use the object constructor `chainSawIZ` to create new objects.

Slots

Most of the slots are described in the superclasses (see below), please see their help pages for more information. This class adds only three slots to the "downLogIZ" class specification.

`circularPlot`**:** Object of class `"circularPlot"`: Object of class `"circularPlot"`: An object from the "`ArealSampling`" subclass "`circularPlot`".

`sliver`**:** Object of class `"SpatialPolygons"`: A `"SpatialPolygons"` object representing the intersection (sliver) between the circular plot and the log.

`bolt`**:** Object of class `"list"`: A list defining the sQuoteminimal bounding bolt within the log that fully encompases the sliver. The list is described in *The InclusionZone Class* vignette.

Extends

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

Methods

izGrid `signature(izObject = "chainSawIZ", tract = "Tract")`: “InclusionZoneGrid” generic constructor

perimeter `signature(object = "chainSawIZ")`: Return the object perimeter

plot `signature(x = "chainSawIZ", y = "missing")`: Plot the object

summary `signature(object = "chainSawIZ")`: Summary of the object

Author(s)

Jeffrey H. Gove

References

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

See Also

`standUpIZ`, `sausageIZ` `fullChainSawIZ` and the `downLogIZs` container class.

Examples

```
showClass("chainSawIZ")
```

`chainSawIZ-methods` *Method for "chainSawIZ" object construction in Package* **sampSurf**

Description

This is the one methods for generic function `chainSawIZ` in Package ‘sampSurf’ that allows for creation of objects of class `"chainSawIZ."`

Methods

`signature(downLog = "downLog", plotRadius = "numeric")`

usage...

```
chainSawIZ(downLog,
           plotRadius,
           plotCenter = c(x=0, y=0),
           description = 'inclusion zone for "chainsaw" method',
           spID = unlist(strsplit(tempfile('cs:',''),'\/'))[2],
           spUnits = CRS(projargs=as.character(NA)),
           ... )
```

- `downLog`: An object of class "`downLog`" which the inclusion zone is to be determined for under the chainsaw method.
- `plotRadius`: The radius of the circular fixed-area plot in the correct units: feet for "English" and meters for "metric."
- `plotCenter`: The (x,y) positions of the center of the circular plot that intersects the log forming the “sliver” that would be cut with a chainsaw at their intersection, and which will be sampled for volume. This should be a numeric vector of length 2 with names "x" and "y" (e.g., see `circularPlot`).
- `description`: A character vector description of the object.
- `spID`: A unique identifier that will be used in displaying the spatial polygon for the circular plot component of the object.
- `spUnits`: A valid `CRS` object specifying the Coordinate Reference System. This defaults to `NA`, which means you want to use your own user-defined system, say for a sample plot located in the field.
- `dots`: Arguments to be passed on.

`checkStemDimensions` *Check for consistency in attribute slots within “Stem” Objects*

Description

Stem class constructors allow one to specify both taper and total stem attributes such as volume and biomass. This function checks to make sure that any inconsistencies between the taper version of these attributes and those specified in the constructor are not too large. See below for details.

Usage

```
checkStemDimensions(stem, tolerancePercent = 1, ...)
```

Arguments

`stem`	An object that is a subclass of "`Stem`"
`tolerancePercent`	The tolerance limit in percent (the default is 1 percent). Differences larger than this will result in a warning.
`...`	Just gobbled for now.

Details

If one specifies a volume (optional) for a "Stem" object in the constructor in addition to the taper data (required), the two may not necessarily be in agreement. The same is true for other attibutes like surface area, etc. This is okay for the usual Horvitz-Thompson type estimators where the sampling surface is flat (constant) within a given inclusion zone. The routines use the entered volume for estimation and the taper data for graphical display. However, the Monte Carlo sampling methods (such as distance limited sampling for down logs, or critical height sampling for standing trees) use the taper data to determine heights or diameters along the bole as part of the estimate. Thus, if the two values for volume (that in the volume slot and that given by integration of the taper data) are not closely commensurate, it can result in an unintended apparent bias in the simulations. Note also that methods like perpendicular distance sampling rely on the taper data for the inclusion zone, so there could be a bias produced here as well.

This function can be used to check how closely the slot and taper values correspond for a given "Stem" subclass object, and will print a warning if the discrepancy is too large. One should not ignore this warning in the cases mentioned above. More information on this is given in the reference below.

Value

A `list` is returned invisibly with...

`stemID`	The stem ID value.
`tolerancePercent`	The value of the above argument specified by the user.
`volume`	A vector with elements: 1. volume from the slot, 2. volume from either the taper function or Smalian's, depending on solidType, 3. splined volume from the taper data, 4. the percent difference.
`surfaceArea`	A vector with elements: 1. surface area slot, 2. splined or taper SA depending on solidType, 3. percent difference.
`coverageArea`	As above in `surfaceArea` but for coverage area.
`biomass`	A vector with elements: 1. biomass slot value, 2. conversion from `volume[2]`, 3. percent difference. Note that if no biomass is specified in the respective object slot, all will be `NA`.
`carbon`	As above for `biomass` but for carbon.

Note

Note that as of **sampSurf** version 0.7-2 this function is run automatically within the constructors for down logs or standing trees. However, one can run it at any time as a check on the object, just like one can invoke `validObject` at any time.

Author(s)

Jeffrey H. Gove

References

Gove, J. H. 2014. Apparent bias in **sampSurf**. R **sampSurf** *package vignette series paper.*

See Also

"`Stem`", "`downLog`", "`standingTree`"

Examples

```
## Not run:
st = standingTree(dbh=20, height=20, solidType=3)
checkStemDimensions(st)
st2 = standingTree(dbh=20, height=20, solidType=3, treeVol=1.1*st@treeVol)
checkStemDimensions(st2)

## End(Not run)
```

`circularPlot` *Generate Objects of Class "*`circularPlot`*"*

Description

This generic function has only one method (`circularPlot-methods`) used as a constructor function for objects that are of class "`circularPlot`". This method should be used in preference to `new` to insure a valid object.

Usage

```
circularPlot(radius, ...)
```

Arguments

`radius`	This is the signature argument: The fixed-plot radius in the correct units
`...`	Arguments that are defined in `circularPlot-methods`

Details

Only one method currently exists for object generation. Its arguments are documented in `circularPlot-methods`.

Value

A valid object of class "`circularPlot`"

Author(s)

Jeffrey H. Gove

See Also

`circularPlot-methods`

Examples

```
cp = circularPlot(37.237, units='English', center=c(x=10,y=3))
summary(cp)
plot(cp, axes=TRUE, showPlotCenter=TRUE, cex=2.5)
```

`circularPlot-class` *Class "circularPlot": Fixed-area Circular Plots*

Description

A subclass of "`ArealSampling`" that can be used to create fixed-area circular plot objects for use in sampling surface simulations.

Objects from the Class

Objects can be created by calls of the form `new("circularPlot",...)`; however, this is not recommended due to the necessity to have the spatial representation correct. Preferably, one should use the `circularPlot` constructor for this class.

Slots

In addition to those slots in class "ArealSampling," the following are defined. . .

`radius`: Object of class `"numeric"`: The fixed-plot radius in the correct units. For "English" units, this would be feet, for "metric" units, meters.

`area`: Object of class `"numeric"`: The area of the plot in the correct units.

`perimeter`: Object of class `"SpatialPolygons"`: The `"SpatialPolygons"` object corresponding to the perimeter of the fixed-radius plot.

`location`: Object of class `"SpatialPoints"`: This is a “SpatialPoints” representation of the location of the object. In the "circularPlot" class, this is the fixed-radius plot center. It will often correspond to the location slot in the "`Stem`" object under sampling surface simulations. But there are exceptions: for example, under the ‘standup’ method, it will be at the large-end of the log, while under the ‘chainsaw’ method, it will be some point within the “sausage” shaped inclusion zone for protocol 1 in (Gove and Van Deusen, 2011).

`spID`: Object of class `"character"`: A unique identifier that is used in the “SpatialPolygons” representation of the object.

`spUnits`: Object of class `"CRS"`: A valid string of class “CRS” denoting the spatial units coordinate system (?CRS for more information) as in package **sp**.

Extends

Class "`ArealSampling`", directly.

Methods

bbox `signature(obj = "circularPlot")`: Return the bounding box.

perimeter `signature(object = "circularPlot")`: Return the perimeter object.

plot `signature(x = "circularPlot",y = "missing")`: Graphical display of the object.

summary `signature(object = "circularPlot")`: A summary of the object.

Author(s)

Jeffrey H. Gove

See Also

The "`ArealSampling`" class.

Examples

```
showClass("circularPlot")
```

`circularPlot-methods` *Methods for "circularPlot" Object Construction in package 'sampSurf'*

Description

There is currently only one method based on the `circularPlot` generic that is used for object construction. It is detailed below.

Methods

`signature(radius = "numeric")` This method takes the plot radius as the signature argument along with other optional aruments described as follows. . .

usage. . .

```
circularPlot(radius,
             units = 'metric',
             spUnits = CRS(projargs=as.character(NA)),
             centerPoint = c(x=0, y=0),
             description = 'fixed area circular plot',
             nptsPerimeter = 100,
             spID = unlist(strsplit(tempfile('cp:',''),'\/'))[2],
             ...)
```

- `radius`: The fixed-plot radius in the correct units. For "English" units, this would be feet, for "metric" units, meters.
- `units`: Either “English” or “metric”. These must be conformable with the projection in `spUnits`.

- spUnits: A valid CRS object specifying the Coordinate Reference System. This defaults to NA, which means you want to use your own user-defined system, say for a sample plot located in the field.
- centerPoint: The location of the center of the fixed-area plot in the appropriate spatial units. This should be a numeric vector of length 2 with names "x" and "y".
- description: A character vector description of the tract.
- nptsPerimeter: The number of points that will compose the spatial perimeter of the object. Note that the final result will always be one more point than requested in order to close the polygon.
- spID: Each object should have its own *unique* identifier that is used in constructing the Polygons object for the perimeter. This becomes very important when combining individual plots into a population. If nothing is supplied, a random ID is generated.

---

circularPlotIZ *Generate Objects of Class "circularPlotIZ"*

Description

This is the generic function for class “circularPlotIZ”. Please see the associated method in circularPlotIZ-methods for more details.

Usage

```
circularPlotIZ(standingTree, plotRadius, ...)
```

Arguments

standingTree	Signature object of class "standingTree".
plotRadius	Signature object for plot radius.
...	See methods.

Details

Since only one method exists for this generic, its signature arguments coincide with the above. Please see circularPlotIZ-methods for more details.

Value

A valid object of class "circularPlotIZ."

Author(s)

Jeffrey H. Gove

References

“*The InclusionZone Class*” vignette.

See Also

Class "`circularPlotIZ`", and `circularPlotIZ-methods`.

Examples

```
st = standingTree(dbh=50, solidType=4, height=25)
iz.cp = circularPlotIZ(st, 5)
summary(iz.cp)
plot(iz.cp, axes=TRUE, cex=2)
```

`circularPlotIZ-class` *Class* `"circularPlotIZ"`

Description

This class holds the inclusion zone definition for the fixed-area circular plot method for sampling standing trees.

Objects from the Class

Objects can be created by calls of the form `new("circularPlotIZ",...)`. However, this is not recommended because the objects are fairly complex. Instead, one can use the object constructor `circularPlotIZ` to create new objects.

Slots

This class adds one slot to the "standingTreeIZ" class specification, please see the superclasses (see below) for more definitions.

`circularPlot`**:** Object of class `"circularPlot"`: An object from the "`ArealSampling`" subclass "`circularPlot`". Please see the help for this class for more information on the slots associated with circular plot objects.

Extends

Class "`standingTreeIZ`", directly.
Class "`InclusionZone`", by class "standingTreeIZ", distance 2.

Methods

area `signature(x = "circularPlotIZ")`: Return the area of the circular plot inclusion zone.

izGrid `signature(izObject = "circularPlotIZ",tract = "Tract")`: Create the raster grid object associated with the inclusion zone.

perimeter `signature(object = "circularPlotIZ")`: Return the object perimeter as a SpatialPolygons object.

plot `signature(x = "circularPlotIZ",y = "missing")`: Plot the object.

summary `signature(object = "circularPlotIZ")`: Summary report for the object.

Author(s)

Jeffrey H. Gove

References

"*The InclusionZone Class*" vignette.

See Also

`standingTreeIZs` container class

Examples

```
showClass("circularPlotIZ")
```

---

circularPlotIZ-methods *Method for "circularPlotIZ" object construction in Package* **sampSurf**

Description

This is the one method for generic function `circularPlotIZ` in Package 'sampSurf' that allows for creation of objects of class "`circularPlotIZ`."

Methods

`signature(standingTree = "standingTree", plotRadius = "numeric")`

usage...

```
circularPlotIZ(standingTree,
               plotRadius,
               description = 'inclusion zone for circular plot method',
               spID = paste('cp',.StemEnv$randomID(),sep=':'),
               spUnits = CRS(projargs=as.character(NA)),
               ... )
```

- `standingTree`: An object of class "`standingTree`" for which the inclusion zone is to be determined.
- `plotRadius`: The radius of the circular fixed-area plot in the correct units: feet for "English" and meters for "metric."
- `description`: A character vector description of the object.
- `spID`: A unique identifier that will be used in displaying the spatial polygon for the circular plot component of the object.
- `spUnits`: A valid `CRS` object specifying the Coordinate Reference System. This defaults to `NA`, which means you want to use your own user-defined system, say for a sample plot located in the field.
- `dots`: Arguments to be passed on.

`classUnions` *Class Unions in* **sampSurf**

Description

Here are the class unions in the **sampSurf** package and their usage.

The Class Unions...

`dlsNumeric`**:** Used where a class of either “distanceLimited” or “numeric” is appropriate.

`pdsIZNull`**:** Used where a class of either “perpendicularDistanceIZ” (or subclass) or `NULL` is appropriate.

`dlsIZNull`**:** Used where a class of either “distanceLimitedIZ” (or subclass) or `NULL` is appropriate.

`numericNULL`**:** Used where a class of either “numeric” or `NULL` is appropriate. Currently, this is only used in the `solidType` slot of a “downLog” object.

`SPNULL`**:** Used where a class of either “SpatialPolygons” or `NULL` is appropriate.

`RLNULL`**:** Used where a class of either “RasterLayer” or `NULL` is appropriate.

`izgNULL`**:** Used where a class of either “InclusionZoneGrid” or `NULL` is appropriate.

`monteNTSampleOrNULL`**:** Used where a class of either “monteNTSample” (or subclass) or `NULL` is appropriate.

`monteBSSampleOrNULL`**:** Used where a class of either “monteBSSample” (or subclass) or `NULL` is appropriate.

`horizontalPointMonteCarloSamplingIZ`**:** This is used to collect all of the classes that allow Monte Carlo subsampling within a horizontal point first stage sample. The class is only used to make it simpler to have one `izGrid` function apply to all of the methods. Note that this does not include the critical height-based Monte Carlo methods.

Author(s)

Jeffrey H. Gove

See Also

`distanceLimitedPDSIZ`, `distanceLimitedPDSIZ-methods`, `downLog`, `monte`, `monteNTSample`, `monteBSSample`

`clipStemsToTract` *Clip stems to lie within tract*

Description

This function and associated methods will ultimately take a collection of "`Stem`" subclass objects and check that they all lie within the associated "`Tract`" subclass object. In addition, if some objects lie either paritally or completely outside the tract, the routine will return a new collection with only those portions of the collection that lie completely within the boundaries of the tract if desired. This function is especially helpful when using either the 'mirage' or 'walkthrough' methods to correct for boundary overlap.

Usage

```
clipStemsToTract(stems, tract, ...)
```

Arguments

`stems`	A collection of "`Stem`" subclass objects (i.e., "`downLogs`" or "`standingTrees`").
`tract`	A "`Tract`" subclass object.
`...`	See methods for other possible arguments.

Details

It is possible to have a collection of stems where some individuals do not lie completely within a desired tract or plot in **sampSurf**. When using buffered tracts, it is fine if some individuals intersect the internal buffer, as long as the respective inclusion zones do not overlap the external boundary. There are circumstances when it is advantageous to clip any tree or log that lies outside the tract out of the population. In addition, logs can intersect the tract boundary, and clipping the external part that is not within the tract is also sometimes useful. This is especially true when using either the mirage or walkthrough boundary correction methods.

One can invoke this function through the appropriate method and it will remove any stem that lies completely outside the tract, and also clip any logs to the tract boundary. If one desires only to check whether any stems are offending in this way, that option is also available. For "`standingTree`" objects, the center location must lie outside the tract. For "`downLog`" objects, the entire needle must lie outside the tract before the object is removed. If the needle intersects the tract, the external portion is clipped accordingly.

This generic function has several methods `clipStemsToTract-methods`, which should be consulted for more details on its use. Note that the method for buffered tracts clips to the internal buffer by default, but one can alternatively choose to clip to the tract boundary if desired.

Value

The data returned are in `list` form with components…

`df`	A data frame with as many rows as there are stems in the collection *originally passed* to the function. Columns identify which stems lie inside, outside and intersect, and if the latter, on which cardinal direction leg of the tract that occurs (can be more than one, e.g., in a corner).
`status`	A simple numeric summary of stems lying inside, outside and intersecting.
`stems`	The new collection of stems (or `NA` if all stems are inside) with the offending stems either removed or clipped.

Note

Please note that only two of the `clipStemsToTract-methods` methods should actually be called by the casual user. The others simply set up the control flow for doing all the dirty work.

Author(s)

Jeffrey H. Gove

See Also

"`StemContainer`", "`Tract`"

Examples

```
#
# make a smaller "plot" inside the large one and clip...
#
## Not run:
btr = bufferedTract(.1, Tract(c(x=50,y=50), cellSize=1))
bb = bbox(btr)
bb[,1] = 20
bb[,2] = 40
smtr = Tract(bb, cellSize=1)
dlogs = downLogs(100, btr)
dlogs2 = clipStemsToTract(dlogs, smtr, runQuiet=FALSE, showPlot=TRUE)
dlogs2$status

## End(Not run)
```

`clipStemsToTract-methods` *Methods for Generic Function* `clipStemsToTract` *in Package* **sampSurf**

Description

Please see the generic description `clipStemsToTract` for the main details. Essentially, using these methods one can check for stems that are wholly or paritally outside the tract, and can clip them to the tract boundary if desired so that only those portions of stems (e.g., logs) within the tract are preserved.

Methods

Only two of the following methods should be called by the user. The others are used to set up the checks and clipping details. The two methods for you to use are noted below, and make it simpler to use this function.

`signature(stems = "downLogs", tract = "SpatialPolygons")` Please do not use this method, it is the "guts" that does all of the detailed checking and clipping for “`downLogs`” collections.

usage...

```
clipStemsToTract(stems,
                 tract,
                 checkOnly = FALSE,
                 runQuiet = TRUE,
                 showPlot = FALSE,
                 ... )
```

- `stems`: A "`downLogs`" collection object.
- `tract`: A "`Tract`" subclass object.
- `checkOnly`: `TRUE`: just check the status of logs, don’t delete or clip any; `FALSE`: remove and clip as needed.
- `runQuiet`: `TRUE`: No feedback report; `FALSE`: a short summary report is printed.
- `showPlot`: `TRUE`: display a plot showing how the logs were clipped. East-west intersections are in green, N-S in red. It will show (by colors) how logs at corners that intersect more than once get clipped.
- ... : Other arguments, just gobbled for now.

`signature(stems = "standingTrees", tract = "SpatialPolygons")` Please do not use this method, it is the "guts" that does all of the detailed checking and removal for “`standingTrees`” collections.

usage...

```
clipStemsToTract(stems,
                 tract,
                 checkOnly = FALSE,
                 runQuiet = TRUE,
                 ... )
```

- `stems`: A "`standingTrees`" collection object.
- `tract`: A "`Tract`" subclass object.
  All other arguments are as previously defined.

`signature(stems = "StemContainer", tract = "bufferedTract")` This is one of the two methods that should be called by the user. A collection of logs or trees can be passed along with a buffered tract object. Please note that the internal buffer rectangle is used to clip against in this case by default, *not* the external tract boundary.

usage...

```
clipStemsToTract(stems,
                 tract,
                 checkOnly = FALSE,
                 runQuiet = TRUE,
                 clipToBuffer = TRUE,
                 ... )
```

- `stems`: A "`downLogs`" or "`standingTrees`" collection object.
- `tract`: A "`bufferedTract`" object.
- `clipToBuffer`: `TRUE`: Clip stems to the internal buffer (default); `FALSE`: clip to the tract boundary.
  All other arguments are as previously defined.

`signature(stems = "StemContainer", tract = "SpatialPolygons")` This method should not be called by the user. Please use the methods for “`Tract`” or “`bufferedTract`” objects. All this does is a general test of whether the bounding box for the collection is within the tract, it is called automatically from the other methods.

usage...

```
clipStemsToTract(stems,
                 tract,
                 checkOnly = TRUE,
                 runQuiet = TRUE,
                 ... )
```

All arguments are as previously defined in other methods. Please don't use this!

`signature(stems = "StemContainer", tract = "Tract")` This is one of the two methods that should be called by the user. A collection of logs or trees can be passed along with a tract subclass (though there is a special interface for buffered tracts, see above) object.

usage...

```
clipStemsToTract(stems,
                 tract,
                 checkOnly = FALSE,
                 runQuiet = TRUE,
                 ... )
```

- `stems`: A "`downLogs`" or "`standingTrees`" collection object.
- `tract`: A "`Tract`" object.
  All other arguments are as previously defined.

`controlVariate` *Generate Objects of Class "*`controlVariate`*"*

Description

This generic has two methods, they are used to apply control variate subsampling to an individual "`Stem`" object, or collections of "`Stem`" objects. See `controlVariate-methods` for details.

Usage

```
controlVariate(object, ...)
```

Arguments

`object`	This is the signature argument, see the `controlVariate-methods` for possible values.
`...`	Arguments that can be passed along to the proxy function.

Details

Control variate sampling uses the differences between measured and proxy model cross-sectional areas to estimate the volume of a bole section. Control variate sampling is sensitive to departures of the proxy from the true bole, and can actually produce negative volumes. See the examples in the vignette reference below for examples illustrating this phenomenon.

Value

A valid object of class "`controlVariate`" or "`mcsContainer`", depending on which method was used.

Author(s)

Jeffrey H. Gove

References

Gove, J. H. 2013. Monte Carlo sampling methods in **sampSurf**. Package vignette.

See Also

See `controlVariate-methods` for methods. Other similar generics for Monte Carlo methods include: `crudeMonteCarlo`, `importanceSampling`, `antitheticSampling`.

Examples

```
#
# estimate volume between 10 and 15 m, using 5 random heights...
#
sTree = standingTree(dbh = 40, topDiam = 0, height = 20, solidType = 2.8)
sTree.cv = controlVariate(sTree, n.s = 5, segBnds = c(10,15), startSeed = 114,
           proxy = 'wbProxy', solidTypeProxy = 2.5, truncateProxyStem = FALSE)
sTree.cv
```

`controlVariate-class` *Class* `"controlVariate"`

Description

This is the class definition that allows for the application of control variate sampling to `downLog` or `standingTree` objects. Examples of the class usage can be found in the Monte Carlo sampling vignette referenced below.

Objects from the Class

Objects can be created by calls of the form `new("controlVariate",...)`. However, an object constructor of the same name, `controlVariate`, has been provided and is the preferred method for creating objects that are ensured to be valid.

Slots

Please note that diameters below are presumed to be in the *same* units as length, i.e., meters for “metric”, and feet for “English” `units`. Cross-sectional areas are in compatible units.

This class has three superclasses that define a number of other slots in addition to the what follows. Please see these classes for the complete set of slot definitions for the "`controlVariate`" class. . .

`diff.s`: Object of class `"numeric"`: This vector holds the `n.s` differences between measured and proxy cross-sectional areas at the different sample points along the bole section.

Extends

Class "`importanceSampling`", directly.
Class "`crudeMonteCarlo`", by class "importanceSampling", distance 2.
Class "`MonteCarloSampling`", by class "importanceSampling", distance 3.

Methods

No new methods defined with class "controlVariate" in the signature. However, various methods such as `summary` and `plot` are available through inheritance.

Author(s)

Jeffrey H. Gove

References

Gove, J. H. 2013. Monte Carlo sampling methods in **sampSurf**. Package vignette.

See Also

`MonteCarloSampling`, `crudeMonteCarlo`, `importanceSampling`, `antitheticSampling`.

Examples

```
showClass("controlVariate")
```

`controlVariate-methods` *Methods for* `controlVariate` *object construction in Package* **sampSurf**

Description

The methods below allow for the construction of individual objects of class "`controlVariate`" when applied to individual "`Stem`" subclass objects. Alternatively, the second constructor can be applied to a collection of "`Stem`" subclass objects and will return an appropriate collection of "`controlVariate`" objects (see, e.g., the "`mcsContainer`" class).

Methods

`signature(object = "Stem")`

usage...

```
controlVariate(object,
               segBnds = NULL,
               n.s = 1,
               startSeed = NA,
               u.s = NA,
               proxy = 'gvProxy',
               alphaLevel = 0.05,
               description = 'Control Variate Sampling',
               ... )
```

- `object`: An object of class "`downLog`" or "`standingTree`".
- `segBnds`: A vector of length two giving the lower and upper height/length bounds for volume estimation within the bole. These bounds correspond to the limits of integration along the bole. If either of the bounds are `NULL` or `NA`, the entire bole is used (default).
- `n.s`: The number of sampled heights desired within `segBnds` for volume estimation.
- `startSeed`: The scalar seed for the random number generator used in the call to the class constructor. Please see the documentation in `initRandomSeed` for possible values and their meaning.
- `u.s`: The uniform random numbers used in selecting the sampling points along the bole. If this is either `NULL` or `NA`, then `n.s` and `startSeed` will be used to determine the random numbers. If this is a numeric vector, then `n.s` is set to its length, and `u.s` is used as the random number stream. No checking is done on the bounds of the numbers so *be careful* if using the latter option. It is most useful in antithetic sampling where the 1-`u.s` stream is used (automatically).
- `proxy`: A character name specifying the proxy function to be used in control variate sampling. See the vignette referenced in the generic for details.

- `alphaLevel`: The two-tailed alpha-level for confidence interval construction.
- `description`: A character vector description of the object.
- `...`: Arguments passed on to the proxy function.

`signature(object = "StemContainer")` See the above definitions for the first method for arguments not shown below.

usage...

```
controlVariate(object,
               segBnds = NULL,
               n.s = 1,
               startSeed = NA,
               u.s = NA,
               proxy = 'gvProxy',
               alphaLevel = 0.05,
               description = 'Control Variate Sampling',
               ... )
```

- `object`: A collection of `"Stem"` class objects in a valid "`StemContainer`" object.
- `segBnds`: The segment bounds, see the definition for the first method. *Note:* These bounds are used for all stems in the collection, so it is up to you to make sure they are legal for each stem.
- `n.s`: The number of sampled heights desired within `segBnds` for volume estimation on each stem in the collection.
- `startSeed`: By default, the stream is started using this seed (see above for the first method) and the current random number stream is continued for each stem in the collection. This results in a *different* set of random numbers for each stem (but all keyed off this starting value).
- `u.s`: If this is `NULL` or `NA`, then the `n.s` and `startSeed` combination are used as above. However, if this is a vector, then it is applied to each stem. Therefore, the *same* set of random numbers will be applied to *each* stem in the collection.
- `...`: Arguments passed on to the proxy function; note that these apply to each stem in the collection.

`criticalHeightIZ` *Generate Objects of Class "*`criticalHeightIZ`*"*

Description

This is the generic function for class “criticalHeightIZ”. Please see the associated method in `criticalHeightIZ-methods` for more details.

Usage

```
criticalHeightIZ(standingTree, angleGauge, ...)
```

Arguments

`standingTree`	Signature object of class "`standingTree`".
`angleGauge`	Signature object of class "`angleGauge`".
`...`	See methods.

Details

Since only one method exists for this generic, its signature arguments coincide with the above. Please see `criticalHeightIZ-methods` for more details.

Value

A valid object of class "`criticalHeightIZ`."

Author(s)

Jeffrey H. Gove

See Also

Class "`criticalHeightIZ`", and `criticalHeightIZ-methods`.

Examples

```
st = standingTree(dbh=50, solidType=4, height=25)
ag = angleGauge(baf=4)
iz.chs = criticalHeightIZ(st, ag)
summary(iz.chs)
plot(iz.chs, axes=TRUE, cex=2)
```

`criticalHeightIZ-class`

Class `"criticalHeightIZ"`

Description

This class holds the inclusion zone definition for critical height sampling for standing trees.

Objects from the Class

Objects can be created by calls of the form `new("criticalHeightIZ",...)`. However, this is not recommended because the objects are fairly complex. Instead, one can use the object constructor "`criticalHeightIZ`" to create new objects.

Slots

This class is a direct descendant of "`horizontalPointIZ`" and adds one new slot to the class definition; refer to that class description for other slots and their meanings. Please do note that the other slots listed below have slightly different meanings here...

`referenceHeight`: Object of class `"character"`: The reference height at which the inclusion zone is determined. It can be either "butt" for the base of the tree, or "dbh" for breast height as specified in `.StemEnv$referenceCHSIZ`.

`puaBlowup`: Object of class `"numeric"`: The exansion factor for every tree is simply the basal area factor (BAF).

`puaEstimates`: Object of class `"list"`: Only volume may be estimated under critical height sampling, all other attribute estimates are `NA`.

Extends

Class `"horizontalPointIZ"`, directly.
Class `"circularPlotIZ"`, by class "horizontalPointIZ", distance 2.
Class `"standingTreeIZ"`, by class "horizontalPointIZ", distance 3.
Class `"InclusionZone"`, by class "horizontalPointIZ", distance 4.

Methods

izGrid `signature(izObject = "criticalHeightIZ",tract = "Tract")`: "InclusionZoneGrid" constructor

summary `signature(object = "criticalHeightIZ")`: Object summary

Author(s)

Jeffrey H. Gove

References

There are numerous references to critical height sampling, the following are a sample...

T. G. Gregroire and H. T. Valentine. 2009. *Sampling strategies for natural resources and the environment*. Chapman and Hall/CRC, 496p.

"*The InclusionZone Class*" vignette.

See Also

See also the `"circularPlotIZ"` and `"horizontalPointIZ"` classes, and the "`standingTreeIZs`" container class

Examples

```
showClass("criticalHeightIZ")
```

criticalHeightIZ-methods *Methods for "criticalHeightIZ" object constuction in Package* **sampSurf**

Description

This documents the one method for generic function `criticalHeightIZ` in Package **sampSurf** that allows for creation of objects of class "`criticalHeightIZ`."

Methods

`signature(standingTree = "standingTree", angleGauge = "angleGauge")`

usage...

```
criticalHeightIZ(standingTree,
                 angleGauge,
                 referenceHeight = .StemEnv$referenceCHSIZ,
           description = 'inclusion zone for critical height sampling method',
                 spID = paste('chs',.StemEnv$randomID(),sep=':'),
                 spUnits = CRS(projargs=as.character(NA)),
                 ... )
```

- `standingTree`: An object of class "`standingTree`" for which the inclusion zone is to be determined.
- `angleGauge`: An object of class "`angleGauge`".
- `referenceHeight`: The height on the stem at which the inclusion zone is to be determined. Currently the choices are “butt” (default) or “dbh”. These are found in the argument assignment `.StemEnv$referenceCHSIZ` above.
- `description`: A character vector description of the object.
- `spID`: A unique identifier that will be used in displaying the spatial polygon for the inclusion zone component of the object.
- `spUnits`: A valid `CRS` object specifying the Coordinate Reference System. This defaults to `NA`, which means you want to use your own user-defined system, say for a sample plot located in the field.
- `dots`: Arguments to be passed on.

crudeMonteCarlo *Generate Objects of Class "`crudeMonteCarlo`"*

Description

This generic has two methods, they are used to apply crude Monte Carlo subsampling to an individual `"Stem"` object, or collections of `"Stem"` objects. See `crudeMonteCarlo-methods` for details.

Usage

```
crudeMonteCarlo(object, ...)
```

Arguments

`object`	This is the signature argument, see the `crudeMonteCarlo-methods` for possible values.
`...`	Arguments that can be passed along to the proxy function.

Details

Crude Monte Carlo is arguably the simplest method for estimating a volume integral within a segment of a stem bole. The equations used in estimation are given in the reference below, which also points to more detailed references. Essentially, the crude Monte Carlo method extracts one or more heights uniformly at random, at which cross-sectional areas are determined. From these, a volume estimate can be made for each subsampled height.

Value

A valid object of class "`crudeMonteCarlo`" or "`mcsContainer`", depending on which method was used.

Author(s)

Jeffrey H. Gove

References

Gove, J. H. 2013. Monte Carlo sampling methods in **sampSurf**. Package vignette.

See Also

See `crudeMonteCarlo-methods` for methods. Other similar generics for Monte Carlo methods include: `importanceSampling`, `controlVariate`, `antitheticSampling`.

Examples

```
#
# estimate volume between 10 and 15 m, using 5 random heights...
#
sTree = standingTree(dbh = 40, topDiam = 0, height = 20, solidType = 2.8)
sTree.cmc = crudeMonteCarlo(sTree, n.s = 5, segBnds = c(10,15), startSeed = 114)
sTree.cmc
```

crudeMonteCarlo-class *Class* "crudeMonteCarlo"

Description

This is the class definition that allows for the application of crude Monte Carlo sampling to downLog or standingTree objects. Examples of the class usage can be found in the Monte Carlo sampling vignette noted below.

Objects from the Class

Objects can be created by calls of the form new("crudeMonteCarlo",...). However, an object constructor of the same name, crudeMonteCarlo, has been provided and is the preferred method for creating objects that are ensured to be valid.

Slots

Please note that diameters below are presumed to be in the *same* units as length, i.e., meters for “metric”, and feet for “English” units. Cross-sectional areas are in compatible units.

In addition to the slots provided by the virtual superclass “MonteCarloSampling”, the following slots are represented…

proxy: Object of class "character": The name of the proxy function used. This should normally be the built-in “cmcProxy”, which is the default and the only proxy allowed by the constructor.

diam.s: Object of class "numeric": A vector of n.s diameters corresponding to the sampled heights hgt.s (see below).

rho.s: Object of class "numeric": A vector of cross-sectional areas corresponding to the diameters in diam.s.

hgt.s: Object of class "numeric": A vector of sampled heights at which the diam.s are taken.

vol.s: Object of class "numeric": A vector of volume estimates associated with the sampled hgt.s and associated diameters and cross-sectional areas.

volEst: Object of class "numeric": The sample mean volume estimate of the vol.s for the bole segment.

volVar: Object of class "numeric": The within bole variance estimate of volEst.

ci.lo: Object of class "numeric": The lower 1-alphaLevel confidence interval on the bole volume estimate.

ci.up: Object of class "numeric": The upper 1-alphaLevel confidence interval on the bole volume estimate.

alphaLevel: Object of class "numeric": The two-tailed alpha-level for confidence interval construction.

trueVol: Object of class "numeric": The true volume for the stem segment being estimated (see segBnds in the base class definition).

relErrPct: Object of class "numeric": The relative error in volume in percent.

Extends

Class "`MonteCarloSampling`", directly.

Methods

plot `signature(x = "crudeMonteCarlo",y = "missing")`: Displays a plot of the stem object and sampled points.

summary `signature(object = "crudeMonteCarlo")`: Prints a summary of the object.

Author(s)

Jeffrey H. Gove

References

Gove, J. H. 2013. Monte Carlo sampling methods in **sampSurf**. Package vignette.

See Also

`MonteCarloSampling`, `importanceSampling`, `controlVariate`, `antitheticSampling`.

Examples

```
showClass("crudeMonteCarlo")
```

---

crudeMonteCarlo-methods

Methods for `crudeMonteCarlo` *object construction in Package* **sampSurf**

Description

The methods below allow for the construction of individual objects of class "`crudeMonteCarlo`" when applied to individual "`Stem`" subclass objects. Alternatively, the second constructor can be applied to a collection of "`Stem`" subclass objects and will return an appropriate collection of "`crudeMonteCarlo`" objects (see, e.g., the "`mcsContainer`" class).

Methods

`signature(object = "Stem")`

usage...

```
crudeMonteCarlo(object,
                segBnds = NULL,
                n.s = 1,
                startSeed = NA,
                u.s = NA,
                alphaLevel = 0.05,
                description = 'crude Monte Carlo',
                ... )
```

- `object`: An object of class "`downLog`" or "`standingTree`".
- `segBnds`: A vector of length two giving the lower and upper height/length bounds for volume estimation within the bole. These bounds correspond to the limits of integration along the bole. If either of the bounds are `NULL` or `NA`, the entire bole is used (default).
- `n.s`: The number of sampled heights desired within `segBnds` for volume estimation.
- `startSeed`: The scalar seed for the random number generator used in the call to the class constructor. Please see the documentation in `initRandomSeed` for possible values and their meaning.
- `u.s`: The uniform random numbers used in selecting the sampling points along the bole. If this is either `NULL` or `NA`, then `n.s` and `startSeed` will be used to determine the random numbers. If this is a numeric vector, then `n.s` is set to its length, and `u.s` is used as the random number stream. No checking is done on the bounds of the numbers so *be careful* if using the latter option. It is most useful in antithetic sampling where the 1-`u.s` stream is used (automatically).
- `alphaLevel`: The two-tailed alpha-level for confidence interval construction.
- `description`: A character vector description of the object.

`signature(object = "StemContainer")` See the above definitions for the first method for arguments not shown below.

usage...

```
crudeMonteCarlo(object,
                segBnds = NULL,
                n.s = 1,
                startSeed = NA,
                u.s = NA,
                alphaLevel = 0.05,
                description = 'crude Monte Carlo',
                ...)
```

- `object`: A collection of "`Stem`" class objects in a valid "`StemContainer`" object.
- `segBnds`: The segment bounds, see the definition for the first method. *Note:* These bounds are used for all stems in the collection, so it is up to you to make sure they are legal for each stem.
- `n.s`: The number of sampled heights desired within `segBnds` for volume estimation on each stem in the collection.
- `startSeed`: By default, the stream is started using this seed (see above for the first method) and the current random number stream is continued for each stem in the collection. This results in a *different* set of random numbers for each stem (but all keyed off this starting value).
- `u.s`: If this is `NULL` or `NA`, then the `n.s` and `startSeed` combination are used as above. However, if this is a vector, then it is applied to each stem. Therefore, the *same* set of random numbers will be applied to *each* stem in the collection.

`csFullInclusionZoneGrid-class`
Class "csFullInclusionZoneGrid"

Description

This is a class definition for objects that enumerate the full set of chainsaw inclusion zones at each grid point within the sausage-shaped inclusion zone for the whole log under protocol 1 of Gove and Van Deusen (2011). It is a useful method to visualize how the chainsaw method generates different estimates at each sample point (grid cell center) within the log's zone, by "cutting" off a different sliver at each point, due to the differing intersection of the circluar plot with the log.

Objects from the Class

Objects can be created by calls of the form `new("csFullInclusionZoneGrid",...)`. However, this is not recommended because the objects are fairly complex. Instead, one can use the object constructor `izGrid` to create new objects. More details are found in *"The InclusionZoneGrid Class"* vignette.

Slots

As noted below, this class is a subclass of "`InclusionZoneGrid`"; it adds only one slot to the class, all others not defined below are as described in the superclass...

`chiz`: Object of class `"list"`: 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. The grid cell center is used as the center point of the circular plot that defines the chainsaw intersection of the plot with the log for each cell.

`iz`: Object of class `"InclusionZone"`: In this class, this slot contains the overall "`fullChainSawIZ`" (sausage) inclusion zone object which delineates the grid cells that intersect the log under protocol 1. Note that one can find the plot radius, etc. within the object contained in this slot.

Extends

Class "`InclusionZoneGrid`", directly.

Methods

No methods defined with class "csFullInclusionZoneGrid" in the signature. Because this is simple extension subclass of "InclusionZoneGrid," it's plotting and other routines will work on this class.

Author(s)

Jeffrey H. Gove

References

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

See Also

`InclusionZoneGrid`, `Stem`, `Tract`, `sampSurf`, `InclusionZone`

Examples

```
showClass("csFullInclusionZoneGrid")
```

`distanceLimited` *Generate Objects of Class "*`distanceLimited`*"*

Description

This generic function has only one method used as a constructor function for objects that are of class "`distanceLimited`". This method should be used in preference to `new` to insure a valid object.

Usage

```
distanceLimited(distanceLimit, ...)
```

Arguments

`distanceLimit`	Simply the distance limit in feet or meters, depending upon the `units` used.
`...`	Arguments that are defined in `distanceLimited-methods`

Details

Only one method currently exists for object generation. Its arguments are documented in `distanceLimited-methods`.

Value

A valid object of class "`distanceLimited`"

Author(s)

Jeffrey H. Gove

See Also

`distanceLimited-methods` and the "`ArealSampling`" class.

Examples

```
#
# this will produce a distance limit of 4 feet...
#
(dlsEng = distanceLimited(4, units="English"))
```

`distanceLimited-class` *Class* `"distanceLimited"`*: Distance Limited Sampling*

Description

A subclass of "`ArealSampling`" that can be used to create objects that encapsulate all the parameters necessary for any variant of distance limited sampling of down woody debris.

Objects from the Class

Objects can be created by calls of the form `new("distanceLimited",...)`, and while this is reasonable with this class because there are no graphical slots, it is still not recommended. The preferred method for creating new objects is via the `distanceLimited` constructor.

Slots

In addition to those slots in class "ArealSampling," the following are defined...

`distanceLimit`**:** Object of class `"numeric"`: This is simply the distance limit in either feet or meters, depending upon the `units` chosen.

Extends

Class `"ArealSampling"`, directly.

Methods

summary `signature(object = "distanceLimited")`: prints a summary of the object

Author(s)

Jeffrey H. Gove

See Also

The "`ArealSampling`" class.

Examples

```
showClass("distanceLimited")
```

distanceLimited-methods *Methods for "distanceLimited" Object Construction in Package* **sampSurf**

Description

There is currently only one method based on the `distanceLimited` generic that is used for object construction. It is detailed below.

Methods

`signature(distanceLimit = "numeric")` This method takes distance limit in appropriate units as the signature argument along with other optional aruments described as follows…

usage…

```
distanceLimited(distanceLimit,
                units = 'metric',
                description = 'distance limited method',
                ...)
```

- `distanceLimit`: The distance limit in appropriate units (feet or meters).
- `units`: Either "English" or "metric".
- `description`: A character vector description of the object.

distanceLimitedIZ *Generate Objects of Class "`distanceLimitedIZ`"*

Description

This is the generic definition for generating objects of class "distanceLimitedIZ." There is only one constructor method corresponding to this generic: `distanceLimitedIZ-methods`

Usage

```
distanceLimitedIZ(downLog, dls, ...)
```

Arguments

`downLog`	Signature object of class "`downLog`".
`dls`	Signature object of class "`distanceLimited`" containing the pertinent distance limit.
`...`	See methods.

Details

Since only one method exists for this generic, its signature arguments coincide with the above definitions. Please see `distanceLimitedIZ-methods` for more details.

Value

A valid object of class "`distanceLimitedIZ`."

Author(s)

Jeffrey H. Gove

References

Notes in vignette form are available from the author.

See Also

Class "`distanceLimitedIZ`", and `distanceLimitedIZ-methods`.

Examples

```
#
# generate a simple inclusion zone object with distance limit
# of 6 feet...
#
dl = downLog(buttDiam=15, solidType=4, logAngle=pi/3, logLen=10, units='English')
dlsEng = distanceLimited(6, units='English')
iz.dls = distanceLimitedIZ(dl, dls=dlsEng)
iz.dls
```

`distanceLimitedIZ-class`
Class `"distanceLimitedIZ"`

Description

This class holds the inclusion zone definition for the simple ‘distance limited’ method for sampling down coarse woody debris.

Objects from the Class

Objects can be created by calls of the form `new("distanceLimitedIZ",...)`. However, this is not recommended because the objects are fairly complex. Instead, one can use the object constructor `distanceLimitedIZ` to create new objects.

Slots

Most of the slots are described in the superclasses (see below), please see their help pages for more information. This class adds a few slots to the "downLogIZ" class specification.

`dls`**:** Object of class `"distanceLimited"`: This supplies the distance limit for establishing the inclusion zone.

`izPerim`**:** Object of class `"matrix"`: A matrix representation of the inclusion zone in homogeneous coordinates. This can be manipulated and plotted as desired for easy access to the inclusion zone where needed.

`area`**:** Object of class `"numeric"`: The exact area of the inclusion zone.

`perimeter`**:** Object of class `"SpatialPolygons"`: This is the inclusion zone perimeter as a "SpatialPolygons" object.

`pgArea`**:** Object of class `"numeric"`: This is the area of the 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, and adjust if necessary the number of points defining the inclusion area perimeter.

Extends

Class `"downLogIZ"`, directly.
Class `"dlsIZNull"`, directly.
Class `"InclusionZone"`, by class "downLogIZ", distance 2.

Methods

izGrid `signature(izObject = "distanceLimitedIZ",tract = "Tract")`: "InclusionZoneGrid" constructor

perimeter `signature(object = "distanceLimitedIZ")`: Return the object perimeter

plot `signature(x = "distanceLimitedIZ",y = "missing")`: Plot the object

summary `signature(object = "distanceLimitedIZ")`: Object summary

Author(s)

Jeffrey H. Gove

References

A vignette comparing the different variants is available from the author.

Gove, J. H., Ducey, M. J. and Valentine, H. T. 2012. A distance limited method for sampling downed coarse woody debris. *In Prep.*

See Also

`downLogIZs` container class

Examples

```
showClass("distanceLimitedIZ")
```

`distanceLimitedIZ-methods`   *Methods for* `"distanceLimitedIZ"` *Object Construction in Package* **sampSurf**

Description

This is the one method for generic function `distanceLimitedIZ` in Package ‘sampSurf’ that allows for creation of objects of class "`distanceLimitedIZ`."

Methods

`signature(downLog = "downLog", dls = "distanceLimited")`

usage...

```
distanceLimitedIZ(downLog,
                  dls,
                  description = 'inclusion zone for down log DL sampling',
                  spID = paste('dl',.StemEnv$randomID(),sep=':'),
                  spUnits = CRS(projargs=as.character(NA)),
                  ... )
```

- `downLog`: An object of class "`downLog`" for which the inclusion zone is to be determined.
- `dls`: An object of class "`distanceLimited`" that supplies the information on the distance limited method for constructing the inclusion zone.
- `description`: A character vector description of the object.
- `spID`: A unique identifier that will be used in displaying the spatial polygon for the inclusion zone component of the object.
- `spUnits`: A valid `CRS` object specifying the Coordinate Reference System. This defaults to `NA`, which means you want to use your own user-defined system, say for a sample plot located in the field.
- `dots`: Arguments to be passed on.

`distanceLimitedMCIZ`   *Generate Objects of Class* `"distanceLimitedMCIZ"`

Description

This is the generic definition for generating objects of class "distanceLimitedMCIZ." There is only one constructor method corresponding to this generic: `distanceLimitedMCIZ-methods`

Usage

```
distanceLimitedMCIZ(downLog, dls, ...)
```

Arguments

downLog	Signature object of class "downLog".
dls	Signature object of class "distanceLimited" containing the pertinent distance limit.
...	See methods.

Details

Since only one method exists for this generic, its signature arguments coincide with the above definitions. Please see distanceLimitedMCIZ-methods for more details.

Value

A valid object of class "distanceLimitedMCIZ."

Author(s)

Jeffrey H. Gove

References

Notes in vignette form are available from the author.

See Also

Class "distanceLimitedMCIZ", and distanceLimitedMCIZ-methods.

Examples

```
#
# generate a simple inclusion zone object with distance limit
# of 6 feet...
#
dl = downLog(buttDiam=15, solidType=4, logAngle=pi/3, logLen=10, units='English')
dlsEng = distanceLimited(6, units='English')
iz.dlmcs = distanceLimitedMCIZ(dl, dls=dlsEng)
iz.dlmcs
```

distanceLimitedMCIZ-class *Class* `"distanceLimitedMCIZ"`

Description

This class holds the inclusion zone definition for the 'distance limited Monte Carlo sampling' method for sampling down coarse woody debris.

Objects from the Class

Objects can be created by calls of the form `new("distanceLimitedMCIZ",...)`. However, this is not recommended because the objects are fairly complex. Instead, one can use the object constructor `distanceLimitedMCIZ` to create new objects.

Slots

This class is a direct descendent (subclass) of `"distanceLimitedIZ"` and adds no new slots to that definition. Please see the `"distanceLimitedIZ"` class definition for details.

Extends

Class `"distanceLimitedIZ"`, directly.
Class `"downLogIZ"`, by class "distanceLimitedIZ", distance 2.
Class `"dlsIZNull"`, by class "distanceLimitedIZ", distance 2.
Class `"InclusionZone"`, by class "distanceLimitedIZ", distance 3.

Methods

See other methods as defined for the direct superclass.

izGrid `signature(izObject = "distanceLimitedMCIZ",tract = "Tract")`: "InclusionZoneGrid" constructor

Author(s)

Jeffrey H. Gove

References

A vignette comparing the different variants is available from the author.

Gove, J. H., Ducey, M. J. and Valentine, H. T. 2012. A distance limited method for sampling downed coarse woody debris. *In Prep.*

See Also

`downLogIZs` container class

Examples

```
showClass("distanceLimitedMCIZ")
```

---

distanceLimitedMCIZ-methods
Methods for "distanceLimitedMCIZ" *Object Construction in Package* **sampSurf**

Description

This is the one method for generic function `distanceLimitedMCIZ` in Package 'sampSurf' that allows for creation of objects of class "`distanceLimitedMCIZ`."

Methods

`signature(downLog = "downLog", dls = "distanceLimited")`

usage...

```
distanceLimitedMCIZ(downLog,
                         dls,
                       description = 'inclusion zone for down log DLMC sampling',
                         spID = paste('dlmc',.StemEnv$randomID(),sep=':'),
                         spUnits = CRS(projargs=as.character(NA)),
                         ... )
```

- `downLog`: An object of class "`downLog`" for which the inclusion zone is to be determined.
- `dls`: An object of class "`distanceLimited`" that supplies the information on the distance limited method for constructing the inclusion zone.
- `description`: A character vector description of the object.
- `spID`: A unique identifier that will be used in displaying the spatial polygon for the inclusion zone component of the object.
- `spUnits`: A valid `CRS` object specifying the Coordinate Reference System. This defaults to `NA`, which means you want to use your own user-defined system, say for a sample plot located in the field.
- `dots`: Arguments to be passed on.

---

distanceLimitedPDSIZ *Generate Objects of Class* "`distanceLimitedPDSIZ`"

Description

This is the generic definition for generating objects of class "distanceLimitedPDSIZ." There is only one constructor method corresponding to this generic: `distanceLimitedPDSIZ-methods`.

Usage

```
distanceLimitedPDSIZ(downLog, pds, dls, ...)
```

Arguments

`downLog`	Signature object of class "`downLog`".
`pds`	Signature object of class "`perpendicularDistance`" containing the pertinent perpendicular distance sampling information.
`dls`	Signature object of class "dlsNumeric" which will accept either an object of class "`distanceLimited`" containing the pertinent distance limited sampling information or a "numeric" object specifying the distance limit.
`...`	See methods.

Details

Since only one method exists for this generic, its signature arguments coincide with the above definitions. Please see `distanceLimitedPDSIZ-methods` for more details.

Value

A valid object of class "`distanceLimitedPDSIZ`."

Author(s)

Jeffrey H. Gove

References

Ducey, M. J., Williams, M. S., Roberge, S., Kenning, R. S., and Gove ,J. H. 2005. Distance limited per- pendicular distance sampling for coarse woody material: Theory and field results. Unpublished.

See Also

Class "`distanceLimitedPDSIZ`", and `distanceLimitedPDSIZ-methods`.

Examples

```
dl = downLog(buttDiam=15, solidType=4, logAngle=pi/3, logLen=10, units='English')
pdsEng = perpendicularDistance(kpds=6, units='English')
dlsEng = distanceLimited(2, units='English')
iz.dlpds = distanceLimitedPDSIZ(dl, pds=pdsEng, dls=dlsEng)
iz.dlpds
is.null(iz.dlpds@pdsPart)
```

distanceLimitedPDSIZ-class *Class* "distanceLimitedPDSIZ"

Description

This class holds the inclusion zone definition for the 'distance limited perpendicular distance sampling' method (Ducey et. al 2012) for sampling down coarse woody debris. This class is fairly complicated because there are three possibilities for the components of the inclusion zone. It is best to read *"The InclusionZone Class"* vignette, along with the paper cited above for more information—the why's and wherefore's are not presented here, only the class documentation.

Objects from the Class

Objects can be created by calls of the form `new("distanceLimitedPDSIZ",...)`. However, this is not recommended because the objects are fairly complex. Instead, one can use the object constructor `distanceLimitedPDSIZ` to create new objects.

Slots

Most of the slots are described in the superclasses (see below), please see their help pages for more information. This class adds a few slots to the "`perpendicularDistanceIZ`" class specification as described below, please especially see that class for more details on other slots.

`dls`: Object of class "`distanceLimited`" supplying the distance limit for the inclusion zone.

`dlsDiameter`: Object of class "`numeric`": The limiting diameter for the zone transition between the regular PDS portion and the DLMC portion (see below, either or both may be missing).

`pdsPart`: Object of class "`pdsIZNull`" holding the perpendicular distance sampling portion of the inclusion zone as a "`perpendicularDistanceIZ`" object (or subclass, i.e., "`omnibusPDSIZ`" object), or `NULL` if none exists.

`dlsPart`: Object of class "`dlsIZNull`" holding the distance limited sampling portion of the inclusion zone as a "`distanceLimitedIZ`" object (or subclass, i.e., "`distanceLimitedMCIZ`" object), or `NULL` if none exists.

`pdsFull`: Object of class "`pdsIZNull`" holding a fully valid "`perpendicularDistanceIZ`" (or subclass, i.e., "`omnibusPDSIZ`" object) delineating the full PDS zone as it would appear if the entire log were treated as a PDS sampling object.

Extends

Class "`perpendicularDistanceIZ`", directly.
Class "`downLogIZ`", by class "perpendicularDistanceIZ", distance 2.
Class "`pdsIZNull`", by class "perpendicularDistanceIZ", distance 2.
Class "`InclusionZone`", by class "perpendicularDistanceIZ", distance 3.

Methods

izGrid `signature(izObject = "distanceLimitedPDSIZ",tract = "Tract")`: "InclusionZone-Grid" constructor

plot `signature(x = "distanceLimitedPDSIZ",y = "missing")`: Plot the object

summary `signature(object = "distanceLimitedPDSIZ")`: Object summary

Author(s)

Jeffrey H. Gove

References

Ducey, M. J., Williams, M. S., Gove, J. H., Roberge, S. and Kenning, R. S., 2012. Distance limited perpendicular distance sampling for coarse woody material: Theory and field results. *Forestry* (in review).

See Also

`downLogIZs` container class

Examples

```
showClass("distanceLimitedPDSIZ")
```

`distanceLimitedPDSIZ-methods`

Methods for "`distanceLimitedPDSIZ`" *Object Construction in Package* **sampSurf**

Description

This is the one method for generic function `distanceLimitedPDSIZ` in Package 'sampSurf' that allows for creation of objects of class "`distanceLimitedPDSIZ`."

Methods

`signature(downLog = "downLog", pds = "perpendicularDistance", dls = "dlsNumeric")`

usage...

```
distanceLimitedPDSIZ(downLog,
                     pds,
                     dls,
              description = 'inclusion zone for down log distance limited PDS',
                     spID = paste('dlpds',.StemEnv$randomID(),sep=':'),
                     spUnits = CRS(projargs=as.character(NA)),
                     pdsType = .StemEnv$pdsTypes,
                     ... )
```

- `downLog`: An object of class "`downLog`" for which the inclusion zone is to be determined.
- `pds`: An object of class "`perpendicularDistance`" that supplies the information on the perpendicular distance method for constructing the inclusion zone.
- `dls`: An object of class "`distanceLimited`" that supplies the information on the distance limited method for constructing the inclusion zone. Alternatively a "numeric" distance limit that will be converted to a "distanceLimited" object.
- `description`: A character vector description of the object.
- `spID`: A unique identifier that will be used in displaying the spatial polygon for the inclusion zone component of the object.
- `spUnits`: A valid `CRS` object specifying the Coordinate Reference System. This defaults to `NA`, which means you want to use your own user-defined system, say for a sample plot located in the field.
- `pdsType`: A character string that specifies the type of perpendicular distance sampling used with regard to the selection (i.e., probability proportional to...) of the log. It can be one of "`volume`", "`surfaceArea`" or "`coverageArea`". (See also `.StemEnv$pdsTypes` for legal types.)
- `dots`: Arguments to be passed on.

`downLog` *Generate Objects of Class "`downLog`"*

Description

This generic function has methods based on the signature formal argument `object`. It is used as a constructor function for objects that are of class "`downLog`". There are two methods that should be used in preference to `new` to insure a valid object.

Usage

```
downLog(object, ...)
```

Arguments

`object`	This is the signature formal argument, see the methods for more details.
`...`	Formal arguments that are different for each method, see those for details.

Details

`downLog` is defined completely with respect to the signature of its methods `downLog-methods`. Methods are available for `data.frame`, and `missing` signatures. Other methods can obviously be added for other signatures as necessary.

Value

A valid object of class "`downLog`."

Author(s)

Jeffrey H. Gove

References

"The Stem Class" vignette in this package.

See Also

downLog-methods

Examples

```
#create a downLog object and show it
dl = downLog(buttDiam=20, solidType=2.1, logAngle=pi/3)
summary(dl)
```

---

downLog-class *Class "downLog": Representation of Downed Logs*

Description

A subclass of virtual class "Stem" that can be used to represent downed coarse woody debris in the form of something resembling a log. The class provides for creation and graphical display of "downLog" objects.

Detail examples and information concerning this class is found in "The Stem Class" vignette, which should be consulted for more details on creating and using objects from this class.

Objects from the Class

Objects can be created by calls of the form `new("downLog",...)`. However, this is *not* recommended. The object has a number of required slots that can be somewhat complex to calculate. Therefore, a constructor function downLog has been provided that will make the desired object with the correct values for the slots. It is strongly recommended that one use this generic's methods to create the log objects.

Slots

Please note that all diameters below are presumed to be in the *same* units as length, i.e., meters for "metric", and feet for "English" `units`.

In addition to the slots provided by the virtual superclass "Stem", the following slots are represented...

`buttDiam`: Object of class `"numeric"`: the log diameter at the large (butt) end in the proper units.

`topDiam`: Object of class `"numeric"`: The small-end diameter of the log.

`logLen`: Object of class `"numeric"`: The log length in meters or feet.

`logAngle`**:** Object of class `"numeric"`: An angle for the log's position established from the center of the log relative to its "needle." The center of the log is equidistant from both ends along the established needle. On a straight log, the needle would correspond to the pith if everything were perfect.

`solidType`**:** Object of class `"numericNULL"`: The form parameter in the simple taper and volume equation presented in Van Deusen (1990) and Gove and Van Deusen (2011).

`logVol`**:** Object of class `"numeric"`: The log's volume.

`surfaceArea`**:** Object of class `"numeric"`: Total log surface area in the appropriate units.

`coverageArea`**:** Object of class `"numeric"`: Total log coverage area in the appropriate units.

`biomass`**:** Object of class `"numeric"`: Total log biomass. This will be `NA` if no conversion was specified at object creation.

`carbon`**:** Object of class `"numeric"`: Total log carbon content. This will be `NA` if no conversion was specified at object creation.

`conversions`**:** Object of class `"numeric"`: A vector with names `c('volumeToWeight','weightToCarbon')` specifying the conversion factors for woody biomass and carbon content.

`taper`**:** Object of class `"data.frame"`: The log's taper, either as specified from measurements, or as generated via `downLog`.

`profile`**:** Object of class `"data.frame"`: The log profile as generated from the taper. The log is assumed to be oriented with the small-end North, and the base at the origin.

`rotLog`**:** Object of class `"matrix"`: A rotated and translated version of `profile` using `logAngle` and `location` fields.

`spLog`**:** Object of class `"SpatialPolygons"`: A `SpatialPolygons` representation of `rotLog` via the `sp` package.

`slNeedleAxis`**:** Object of class `"SpatialLines"`: A fully transformed (rotated and translated) representation of the needle for the log as a `SpatialLines` object from package `sp`.

Extends

Class `"Stem"`, directly.

Methods

bbox `signature(obj = "downLog")`: Returns the bounding box for the object.

chainSawIZ `signature(downLog = "downLog",plotRadius = "numeric")`: Chainsaw method inclusion zone constructor.

plot `signature(x = "downLog",y = "missing")`: Graphically display a "downLog" object.

sausageIZ `signature(downLog = "downLog",plotRadius = "numeric")`: Sausage sampling inclusion zone constructor.

standUpIZ `signature(downLog = "downLog",plotRadius = "numeric")`: Stand-up method inclusion zone constructor.

summary `signature(object = "downLog")`: Summary of the object.

Author(s)

Jeffrey H. Gove

References

Gove, J. H., Williams, M. S., Stahl, G. and Ducey, M. J. 2005. Critical point relascope sampling for unbiased volume estimation of downed coarse woody debris. *Forestry* **78**:417–431.

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

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

See Also

`"Stem"`, `"standingTree"`, `"StemContainer"`

Examples

```
showClass("downLog")
```

`downLog-methods` *Methods for "downLog" Object Construction in package* **sampSurf**

Description

There are currently two available methods for the generic `downLog`. These methods generate objects of class "`downLog`" that are valid objects. This is the preferred method for generating such objects, rather than using `new`.

Methods

`signature(object = "missing")` This constructor will be used when the signature argument is "missing." When `object` is `missing`, the taper data is generated from the internal taper function between the `buttDiam` and `topDiam` diameters specified, in `nSegs` sections. The taper function used is documented in *"The Stem Class"* vignette and references for "`downLog`." The following arguments are part of the function call; all arguments with the same names as class slots are also defined in the class definition (and may be stored differently than the arguments).

usage...

```
downLog(
        buttDiam = 5,                #cm
        topDiam = 0,                 #cm
        logLen = 5,                  #meters
        nSegs = 20,
        solidType = 3,               #defaults to 3
        logAngle = 0,                #canonical position
        logVol = NULL,
        surfaceArea = NULL,
        coverageArea = NULL,
        biomass = NA,
        vol2wgt = NA,
        carbon = NA,
        wgt2carbon = NA,
        centerOffset = c(x=0, y=0),   #log center offset
        species = '',
        logID = paste('log',format(runif(1),digits=6),sep=':'),
        description = NULL,
        userExtra = NULL,
        units = 'metric',
        spUnits = CRS(projargs=as.character(NA)),
        runQuiet = TRUE,
        ...)
```

- `buttDiam`: The large-end diameter. For object creation, units are in either *inches* or *cm*. Internally, within the object, they are stored in the same units as length: *feet* or *meters*, depending on the value for `units`.
- `topDiam`: The small-end diameter with same units as large-end.
- `logLen`: The log length in meters or feet, depending on `units`.
- `nSegs`: The number of log segements to be generated from the taper function for the "missing" signature. Note that there will be `nSegs+1` diameter measurements for the log taper.
- `solidType`: The type of solid for the default taper equation; the range is from 1 to 10, with 1 being a neiloid, 2 a cone and 3 a paraboloid. `NULL` is the default, when the `object` argument is missing it defaults to a value of 3.
- `logAngle`: The log lie angle specified from the center of the log. It can take a value from 0 to `2*pi` and is releative to East being zero.
- `logVol`: The log volume if precomputed, otherwise, if `NULL`, the log volume will be computed from the taper volume equation or Smalian's formula if `solidType=NULL`.
- `surfaceArea`: The log surface area if precomputed, otherwise, if `NULL`, the log surface area will be computed from the taper equation or spline approximation.
- `coverageArea`: The log coverage area if precomputed, otherwise, if `NULL`, the log coverage area will be computed from the taper equation or spline approximation.
- `biomass`: The log woody biomass if precomputed, otherwise, if `NA`, the log biomass will be computed from the volume and `vol2wgt` conversion.
- `vol2wgt`: The volume to weight conversion factor. If `NA` and `biomass` is passed, then it will be computed.
- `carbon`: The log carbon content if precomputed, otherwise, if `NA`, the carbon content will be computed from the biomass and `wgt2carbon` conversion.
- `wgt2carbon`: The weight to carbon conversion factor. If `NA` and `carbon` is passed, then it will be computed.
- `centerOffset`: The log center position that will be used for the `location` slot. This is a vector of length two with names "x" and "y"; note that it can be length three with a "z" coordinate, but it is not used anywhere currently.
- `species`: Some species identifier as a character string.

- `logID`: Each log should have its own *unique* identifier that is used in constructing the `Polygons` object for the perimeter. This becomes very important when combining individual logs into a population or collection via the container class "`downLogs`." If nothing is supplied, a random ID is generated.
- `description`: A character vector description of the log.
- `userExtra`: Anything else that one wants to carry along with the log.
- `units`: Either "English" or "metric". These must be conformable with the projection in `spUnits`.
- `spUnits`: A valid `CRS` object specifying the Coordinate Reference System. This defaults to `NA`, which means you want to use your own user-defined system, say for a sample plot where the log has been located in the field.
- `runQuiet`: `TRUE`: no feedback during object creation; `FALSE`: prints some information along the way.
- ... : Other arguments to be passed along.

`signature(object = "data.frame")` When `object` is a "data.frame," then it is assumed that the data frame contains the taper data in the form of diameters and lengths (as columns with labels "diameter" and "length", respectively), with diameters in the *same* units as length. All arguments except those listed below are the same as in the previous constructor…

usage…

```
downLog(object,
        solidType = NULL,             #defaults to null for passed taper
        logAngle = 0,                 #canonical
        logVol = NULL,
        surfaceArea = NULL,
        coverageArea = NULL,
        biomass = NA,
        vol2wgt = NA,
        carbon = NA,
        wgt2carbon = NA,
        centerOffset = c(x=0, y=0),   #log center offset
        species = '',
        logID = paste('log',format(runif(1),digits=6),sep=':'),
        description = NULL,
        userExtra = NULL,
        units = 'metric',
        spUnits = CRS(projargs=as.character(NA)),
        runQuiet = TRUE,
        ...)
```

- `object`: A data frame (see note below).
- `solidType`: `NULL` is the default, because it is assumed that the taper data are from field measurements or have been generated from a different taper equation where this would not be a meaningful parameter. One can therefore have an educated guess about the genesis of taper data within an object by querying this slot in the completed object.
- `logVol`: The log volume if precomputed, otherwise, if `NULL`, the log volume will be computed from the taper data frame using Smalian's method.

Note

It may not be immediately apparent how the taper data in the data frame is to be structured if you have this data available either from measurements, or from a different taper equation. The best way to check this out is to simply create a dummy "downLog" object and then `show` or `print` it for a summary, which will show the first few records of the structure. If that is not enough, then you can look at the structure with the `@` operator applied to the object's `taper` slot. Please also see the vignette mentioned above. Remember, the diameters in the taper data frame are expected to be in the same units as length for a data frame.

Author(s)

Jeffrey H. Gove

See Also

The "`downLog`" class and the `downLog` generic.

Examples

```
#create a downLog object and show it
dl = downLog(buttDiam=20, solidType=2.1, logAngle=pi/3)
summary(dl)
```

---

downLogIZ-class *Class "downLogIZ"*

Description

This represents an incremental change from the virtual base "`InclusionZone`" class. It is meant to clearly facilitate the division between standing trees and down logs with regard to inclusion zones, and it is a virtual class like its direct parent.

Objects from the Class

A virtual Class: No objects may be created from it.

Slots

Only one slot has been added to this class, please see the definition of the "`InclusionZone`" class for other inherited slots...

`downLog`**:** Object of class `"downLog"`: Holds and object of class "`downLog`".

Extends

Class `"InclusionZone"`, directly.

Methods

summary `signature(object = "downLogIZ")`: Print a summary for any subclass object

Author(s)

Jeffrey H. Gove

References

“*The InclusionZone Class*” vignette.

See Also

For subclasses, see: `standUpIZ`, `chainSawIZ`, `sausageIZ`, `pointRelascopeIZ`, `perpendicularDistanceIZ`, `omnibusPDSIZ`, `distanceLimitedPDSIZ`, `omnibusDLPDSIZ`, `distanceLimitedIZ`, `distanceLimitedMCIZ`, and the `downLogIZs` container class.

Examples

```
showClass("downLogIZ")
```

`downLogIZs` *Generate Objects of Class "*`downLogIZs`*"*

Description

This generic function has one method based on the signature formal argument `object`. It is used as a constructor function for objects that are of class "`downLogIZs`". The associated method should be used in preference to `new` to insure a valid object.

Usage

```
downLogIZs(object, ...)
```

Arguments

`object`	Signature formal argument.
`...`	Formal arguments that are different for each method, see those for details.

Details

The methods that can be used to generate objects of class “`downLogIZs`” are detailed here: `downLogIZs-methods`.

Value

A valid object of class "`downLogIZs`."

Author(s)

Jeffrey H. Gove

References

"*The InclusionZone Class*" vignette.

See Also

For subclasses of `downLogIZ`, see: `standUpIZ`, `chainSawIZ`, `sausageIZ`, `pointRelascopeIZ`, `perpendicularDistanceIZ`, `omnibusPDSIZ`, `distanceLimitedPDSIZ`, `omnibusDLPDSIZ`, and `distanceLimitedIZ`. `distanceLimitedMCIZ`.

Examples

```
dls = downLogs(4, units='English', logLens=c(10,30))
dls.su = lapply(dls@logs, 'standUpIZ', plotRadius=5)
izs.su = downLogIZs(dls.su)
bbox(izs.su)
plot(izs.su, axes=TRUE)
```

---

downLogIZs-class *Class "downLogIZs"*

Description

Like the "`downLogs`" class, this a 'container' class that allows multiple objects of a specific subclass of `downLogIZ` to be grouped into a population or collection. Its specific intent it to hold a collection of "`InclusionZone`" objects for a given log population, that will ultimately be used to generate a sampling surface for a given sampling method. Please see "*The InclusionZone Class*" vignette for more information.

Objects from the Class

Objects can be created by calls of the form `new("downLogIZs",...)`; however, as in the other classes within this package, constructors have been written to simplify the process. A `downLogIZs` constructor should therefore be used in preference to `new`. The constructor, `downLogIZs`, has different forms that will be useful with different classes of objects.

Slots

The slots are the same as for the "`izContainer`" virtual superclass; please see it for details beyond what is below.

`iZones`: Object of class `"list"`: This slot holds a list of objects that all correspond to the same subclass of the "`downLogIZ`" class.

Extends

Class `"izContainer"`, directly.

Methods

No methods defined with class "downLogIZs" in the signature.

Author(s)

Jeffrey H. Gove

References

"*The InclusionZone Class*" vignette.

See Also

See the subclasses of `downLogIZ` for valid object types that can be stored in this container.

Examples

```
showClass("downLogIZs")
```

`downLogIZs-methods` *Method for "downLogIZs" object construction in Package 'sampSurf'*

Description

The following are the constructor methods for class "`downLogIZs`" in Package 'sampSurf'. Please see "*The InclusionZone Class*" vignette for more information and examples on the usage of these methods, as well as the `downLogIZs` generic and class `downLogIZs`.

Methods

`signature(object = "list")` This method will create a container object out of a list containing "`downLogIZ`" objects.

usage...

```
downLogIZs(object,
           description = '',
           ... )
```

- `object`: A list object containing the collection of inclusion zones for a given subclass of "`downLogIZ`". Note that the list must contain objects that are all of the same class, or sampling method.
- `description`: A description of the collection.
- ... : Other arguments to be passed along (not used currently).

`signature(object = "downLogs")` Create a container object out of a "`downLogs`" container object and an "`InclusionZone`" subclass specification.

usage...

```
downLogIZs(object,
           iZone,
           description = '',
           ... )
```

- `object`: A valid "downLogs" container object.
- `iZone`: A legal "InclusionZone" class specification (constructor name) that is relevant to "downLog" objects, as a character string.
- `description`: A description of the collection.
- ... : Other arguments to be passed along, e.g., to the `iZone` constructor.

`downLogs` *Generate Objects of Class "downLogs"*

Description

This generic function has methods based on the signature formal arguments `object` and `container`. It is used as a constructor function for objects that are of class "downLogs". This method should be used in preference to `new` to insure a valid object.

Usage

```
downLogs(object, container, ...)
```

Arguments

`object`	Signature formal argument to key on whether logs are passed or generated.
`container`	Relevant if the logs are to be contained within some physical boundary/area like a "Tract" object.
`...`	Formal arguments that are different for each method, see those for details.

Details

The methods that can be used to generate objects of class "downLogs" are detailed here: `downLogs-methods`. As mentioned, each has a different signature and supporting formal arguments allowing you to generate a collection in various ways. Please see the above link for more details.

Value

A valid object of class "downLogs"

Author(s)

Jeffrey H. Gove

References

"The Stem Class" vignette in this package.

See Also

downLog, "downLog", "StemContainer"

Examples

```
showMethods("downLogs")
dlogs = downLogs(15, xlim=c(0,20), ylim=c(10,40), buttDiams=c(10,35))
summary(dlogs)
plot(dlogs, axes=TRUE)
```

`downLogs-class` *Class "downLogs"*

Description

The “downLogs” class is a simplified container class that can hold multiple objects of class “downLog”. Its specific purpose is to hold a population of down logs that are either generated synthetically as part of a simulation, or a collection from field measurements. The constructor of the same name has several different forms corresponding to possible argument signatures.

Objects from the Class

Objects can be created by calls of the form `new("downLogs",...)`; however, as in the other classes within this package, constructors have been written to simplify the process. The `downLogs` constructor should therefore be used in preference to new.

Slots

Please see the virtual base class, “StemContainer”, for additional slots definitions.

`logs`**:** Object of class `"list"`: This holds the collection of “downLog” objects.

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 your R code. 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.

Extends

Class `"StemContainer"`, directly.

Methods

hist `signature(x = "downLogs")`: Displays a histogram of different variables in the collection.

plot `signature(x = "downLogs",y = "missing")`: Plot the collection.

summary `signature(object = "downLogs")`: Same as show currently.

Author(s)

Jeffrey H. Gove

See Also

sampleLogs, "downLog", "Stem", "StemContainer"

Examples

```
showClass("downLogs")
buff = matrix(c(0,100,0,100), nrow=2, byrow=TRUE,
              dimnames=list(c('x','y'),c('min','max')))
sl = sampleLogs(10, buttDiam = c(10,25), sampleRect = buff)
dls = downLogs(sl)
summary(dls)
```

---

downLogs-methods *Methods for "downLogs" Object Construction*

Description

The following are the constructor methods for class "downLogs" in Package **sampSurf**. Please see "*The Stem Class*" vignette for more information and examples on the usage of the methods.

Methods

`signature(object = "list", container = "missing")` Ultimately, all of the constructors end up calling this one to generate the object after they have done what they were meant to, and then converted all of the "downLog" objects into a list structure. So if one wishes, one can simply create one's own list of "downLog" objects and use this constructor.

usage...

```
downLogs(object,
         description = '',
         ... )
```

- `object`: A `list` as discussed above.
- `description`: A character description of the collection.
- ... : Other arguments to be passed along (not used currently).

`signature(object = "data.frame", container = "missing")` This particular method allows one to pass a data frame in the form generated from sampleLogs in `object` argument to construct an object of class "downLogs". Note, however, that the data frame does not have to be created by `sampleLogs`, and it does not have to contain synthetic/simulated logs. As long as all of the columns are present that are generated by `sampleLogs` it will be used; for example, the data frame can be constructed from a field sample of logs.

usage...

```
downLogs(object, units = 'metric', ... )
```

- `object`: A data frame as discussed above.
- `units`: The units of measurement.
- ... : Other arguments to be passed along to the `downLog` constructor—these apply uniformly to all logs in the collection.

`signature(object = "numeric", container = "bufferedTract")` This method will take as its first argument (`object`) the number of logs to be synthetically generated. The centers of the logs will all lie within the interior region of the "`bufferedTract`" object passed in the second (`container`) argument.

usage...

```
downLogs(object, container, units = 'metric', ... )
```

- `object`: A numeric (truncated to integer) object specifying the number of logs to generate as discussed above.
- `container`: A "`bufferedTract`" object specifying the internal portion of the tract within which the log centers will be generated.
- `units`: The units of measurement.
- ... : Other arguments to be passed along to the last constructor described below.

`signature(object = "numeric", container = "missing")` This constructor looks like it just generates logs randomly. But in reality, note the arguments below. It takes limits of a rectangular region as an alternative to specifying a bounding box matrix or a "bufferedTract" object as in two of the other constructors. Essentially, these limits get converted to a matrix bounding box and the appropriate constructor is used (see below).

usage...

```
downLogs(object,
         xlim = c(0,100),
         ylim = c(0,50),
         units = 'metric',
         ...)
```

- `object`: See explanation in the following constructor method.
- `units`: The units of measurement.
- `xlim`: Limits for the bounding rectangle in `x`.
- `ylim`: Limits for the bounding rectangle in `y`.
- ... : Other arguments to be passed along as described in the last constructor method.

`signature(object = "numeric", container = "matrix")` Like the previous constructor, the logs will be generated so that their centers lie within a rectangular area specified by the `matrix` object in the second argument. Note that ultimately, the synthetic population of logs is generated by a call to `sampleLogs`.

usage...

```
downLogs(object,
         container,
         units = 'metric',
         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,
         ... )
```

- `object`: A numeric (truncated to integer) object specifying the number of logs to generate as discussed above.
- `container`: A `matrix` object specifically in the form of a `bbox`, bounding box object (e.g., see the `sp` package). It must be a 2x2 matrix with row names `c("x","y")` and the column names must be `c("min","max")`. The object specifies the internal portion of the tract within which the log centers will be generated.
- `units`: The units of measurement.
- `buttDiams`: A length-two vector specifying the *range* of butt (large-end) diameters from which to uniformly draw the sample. Note that this range is assumed to be specified in cm for metric and inches for English units.
- `topDiams`: A length-two vector specifying the *range* of top (small-end) diameters in the form of a *proportion* of the `buttDiam` diameters, from which to uniformly draw the sample. See also `sampleLogs`.
- `logLens`: A length-two vector specifying the *range* of log lengths in feet (English) or meters (metric) from which to uniformly draw the sample.
- `logAngles`: A length-two vector specifying the *range* of log angles from which to draw the log lie angles; these are always counterclockwise relative to due East.
- `solidTypes`: A length-two vector specifying the range in the log shape parameter for the default taper and volume equations; where: 1-2 is a neiloid, 2 is a cone, and >2 is a paraboloid.
- `species`: A character vector of possible species from which to draw the sample uniformly. This can be any legal character string, and so can include codes, names, Latin names, etc.
- ... : Other arguments to be passed along to `sampleLogs`. Note specifically that one can control the random number stream by specifying a seed to be passed to `sampleLogs` with its `startSeed` argument. Additionally, this list can also contain arguments to be ultimately passed on to the `downLog` constructor to be applied to each individual log in the collection. For example, one can specify the number of segments desired in the taper function (`nSegs`) in this way.

`fullChainSawIZ` *Generate Objects of Class "*`fullChainSawIZ`*"*

Description

This is the generic function for class “fullChainSawIZ”. Please see the associated method in `fullChainSawIZ-methods` for more details on how to construct objects of this class.

Usage

```
fullChainSawIZ(downLog, plotRadius, ...)
```

Arguments

`downLog`	Signature object of class "`downLog`".
`plotRadius`	Signature object for plot radius.
`...`	See methods.

Details

The actual inclusion zone is the same as that for "`sausageIZ`." See the reference below for more details on how the actual sampling method works. Also *The "InclusionZone" Class* and *The "InclusionZoneGrid" Class* vignettes provide examples.

Value

A valid object of class "`fullChainSawIZ`."

Note

Note in particular that the `puaEstimates` slot is not valid in the sense that there is no one estimate for each attribute under this method as the surface will be variable from point to point; therefore, the attributes in this slot will all be assign `NA` by the constructor.

Author(s)

Jeffrey H. Gove

References

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

See Also

Class "`fullChainSawIZ`", and `fullChainSawIZ-methods`.

Examples

```
dl = downLog(buttDiam=40, solidType=4, logAngle=4*pi/3, logLen=6)
iz.fcs = fullChainSawIZ(dl, plotRadius=3)
summary(iz.fcs)
plot(iz.fcs, axes=TRUE, showPlotCenter=TRUE, cex=2, showNeedle=TRUE)
```

fullChainSawIZ-class *Class* "fullChainSawIZ"

Description

This class holds the inclusion zone definition for the full ‘chainsaw’ method (Gove and Van Deusen, 2011) for sampling down coarse woody debris.

Objects from the Class

Objects can be created by calls of the form `new("fullChainSawIZ",...)`. However, this is not recommended because the objects are fairly complex. Instead, one can use the object constructor `fullChainSawIZ` to create new objects.

Slots

This class is a direct subclass of “sausageIZ” and adds no new slots. Please see `"sausageIZ"` class definition for details.

Extends

Class `"sausageIZ"`, directly.
Class `"downLogIZ"`, by class "sausageIZ", distance 2.
Class `"InclusionZone"`, by class "sausageIZ", distance 3.

Methods

izGrid `signature(izObject = "fullChainSawIZ",tract = "Tract")`: Used in sampling surface construction.

Note

The inclusion zone for the full chainsaw method is the sausage inclusion zone, but it is given its own class here so that we are later able to use it as a signature argument for actually constructing the sampling surface. Each grid cell within the inclusion zone will have its estimate determined based on the `"chainSawIZ"` method for fixed-area plots, which produces a variable surface. All estimates for this method are unbiased with the exception of Density, which can have severe bias. The sausage method must be used to get unbiased estimates of this attribute. See the reference below for more details.

Author(s)

Jeffrey H. Gove

References

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

See Also

`standUpIZ`, `chainSawIZ`, `sausageIZ` and the `downLogIZs` container class.

Examples

```
showClass("fullChainSawIZ")
```

`fullChainSawIZ-methods` *Methods for “*`fullChainSawIZ`*” object construction in Package* **sampSurf**

Description

The one constructor method that is available for creating valid objects of class "`fullChainSawIZ`" is documented below. Note that because the inclusion zone for this method is exactly the same as that for the "`sausageIZ`" class (sausage sampling), the same arguments are used here as in that constructor.

Methods

`signature(downLog = "downLog", plotRadius = "numeric")`

usage...

```
fullChainSawIZ(downLog,
                    plotRadius,
                    nptsHalfCircle = 50,
               description = 'Inclusion zone for "fullChainSaw" sampling method',
                    spID = paste('fcs', .StemEnv$randomID(), sep=':'),
                    spUnits = CRS(projargs = as.character(NA)),
                    ... )
```

- `downLog`: An object of class "`downLog`" which the inclusion zone is to be determined for under this method.
- `plotRadius`: The radius of the circular fixed-area plot in the correct units: feet for "English" and meters for "metric."
- `nptsHalfCircle`: The number of points to use in the half-circle at each end of the sausage-shaped inclusion zone for the polygon representation of the object.
- `description`: A character vector description of the object.
- `spID`: A unique identifier that will be used in displaying the spatial polygon component of the object.
- `spUnits`: A valid `CRS` object specifying the Coordinate Reference System. This defaults to `NA`, which means you want to use your own user-defined system, say for a sample plot located in the field.
- `dots`: Arguments to be passed on.

getID *Retrieve Object IDs*

Description

This generic's methods will get the ID that is used to identify the object. Sometimes this can be embedded down within the object's spatial slot components, so this function just makes it simpler to find this information.

Usage

```
getID(object, ...)
```

Arguments

`object`	An object for a class supported by the methods.
`...`	Not used at present.

Details

Currently, methods exist for objects of classes described in `getID-methods`.

Value

The return is a vector of IDs for the object (e.g., container classes will normally have multiple "Stem" objects).

Author(s)

Jeffrey H. Gove

See Also

"`downLog`", "`downLogs`", "`standingTree`", and "`standingTrees`" class definitions.

Examples

```
st = standingTree()
getID(st)
```

`getID-methods` *Methods for Function* `getID` *in Package* **sampSurf**

Description

Please see the generic, this is a simple function for which each of the methods returns a vector (sometimes of length one) of identifiers for the objects passed. The IDs can be buried deeply within the object, especially when they are spatial IDs, so it is simpler to have these little helper methods.

Please note that identifiers for "`Stem`" class objects are the spatial IDs used in the polygon representing the individual log or tree.

Methods

`signature(object = "downLog")` This will return the spatial ID for the log as the log's identifier.

usage...

```
getID(object, ...)
```

- `object`: An object of class "downLog".

`signature(object = "downLogs")` Returns a vector of spatial IDs for the logs in the collection.

usage...

```
getID(object, ...)
```

- `object`: An object of class "downLogs".

`signature(object = "standingTree")` This will return the spatial ID for the tree as the tree's identifier.

usage...

```
getID(object, ...)
```

- `object`: An object of class "standingTree".

`signature(object = "standingTrees")` Returns a vector of spatial IDs for the trees in the collection.

usage...

```
getID(object, ...)
```

- `object`: An object of class "standingTrees".

getProxy *Proxy functions for Monte Carlo sampling methods in* **sampSurf**

Description

These functions encapsulate the proxy capabilities under Monte Carlo sampling within "Stem" subclass objects for the estimation of volume.

Usage

```
getProxy(proxy = c("cmcProxy", "gvProxy", "wbProxy"), ...)

cmcProxy(stem, u.s, segBnds, ...)

gvProxy(stem, u.s, segBnds, ...)

wbProxy(stem, u.s, segBnds, solidTypeProxy = 3, truncateProxyStem = TRUE,
        wbProxySolve = c('uniroot', 'nlminb'), warningsOn = FALSE, ...)
```

Arguments

`proxy`	A character argument specifying the name of the proxy function to retrieve.
`stem`	An object that is a subclass of "Stem", i.e., a "downLog" or "standingTree" object.
`u.s`	The uniform (0,1) random number vector for Monte Carlo sampling to determine the sampled heights for the respective method.
`segBnds`	A vector of length two giving the lower and upper height/length bounds for volume estimation within the bole. These bounds correspond to the limits of integration along the bole.
`solidTypeProxy`	The applicable values for this and what they do depend on whether the default **sampSurf** taper equation was used to construct the `stem` object passed. In general, specifying `NA` will use the exact taper for the `stem`, so there should be no estimation error. Otherwise, this specifies a value for the shape parameter to be used in the proxy function in the default taper system. Specifying a value between zero and one will yield a proxy stem with shape parameter that is this proportion of the true shape parameter for the `stem` passed (i.e., a value of 0.9 gives a proxy that is pretty close to the actual `stem`). If user-defined taper model was used to construct `stem`, then this this range is not applicable and the default vaule of 3 is used. This has been a very terse explanation of this argument, please consult the vignette refernece below for details on its use.
`truncateProxyStem`	`TRUE` enlarge the tip diameter if it is zero so inflation of the estimate is not a concern if cross-sectional areas close to zero are sampled in importance sampling. `FALSE` the proxy will taper to the tip if the `stem` passed does. Again, more details are given in the vignette.

`wbProxySolve`	The two options specify which method is used to numerically solve for the proxy heights. The default should normally be used, the `'nlminb'` option is slow.
`warningsOn`	`TRUE`: print warnings if necessary; `FALSE`: runs silently (default).
`...`	Used to ignore extraneous arguments passed at present.

Details

The first function, `getProxy`, returns one of the built-in proxy functions from the **sampSurf** namespace so that it can be used within the respective Monte Carlo sampling method. In the case of a user proxy, it will also look for the proxy within the user's workspace. The function does a number of checks for the minimal format of "proxy" functions that are described in the vignette reference below. Therefore, it will catch problems with arguments and return lists in user-defined proxies. This is necessary to make sure all user-written proxy functions accept and return the standard set of arguments and list components.

The built-in proxy functions are discussed in detail in the vignette. The built-in proxies range from very simple to fairly complex. They include...

`cmcProxy`: This is the default proxy used for crude Monte Carlo sampling in the `crudeMonteCarlo` object constructor. It is also used in control variate sampling in the `controlVariate` constructor for selecting the height/length sample points.

`gvProxy`: This is a simple proxy that is proportional to cross-sectional area at a given sampled height. It is the default proxy used in `importanceSampling` and `controlVariate` (as the proxy cross-sectional area model) methods for creation of their respective objects. Details on its uses and potential concerns are found in the vignette.

`wbProxy`: This proxy uses the default taper equation in **sampSurf**. This is a much more complicated proxy than the other two, and can take more arguments. It can be used in both `importanceSampling` and `controlVariate` methods for determination of the sample heights and cross-sectional areas. Please see the vignette below for details on this function and its use.

Value

`getProxy` returns the proxy function. The other functions return a list of the form...

`g`	The function (closure) that actually determines the cross-sectional area at a given height.
`G`	The volume integral for the stem segment.
`hgt.s`	The sampled heights along the bole segment at which `g` will be evaluated.

Please see the vignette for more details.

Author(s)

Jeffrey H. Gove

References

Gove, J. H. 2013. Monte Carlo sampling methods in **sampSurf**. Package vignette.

See Also

MonteCarloSampling, crudeMonteCarlo, importanceSampling, controlVariate, antitheticSampling.

Examples

```
#
# retrieve a built-in proxy and use it...
#
sTree = standingTree(dbh = 40, topDiam = 0, height = 20, solidType = 2.8)
cmcFun = getProxy('cmcProxy')
( cmcFun(sTree, runif(4), c(0, sTree@height)) )
```

---

gridCellEnhance *Add Grid Lines and Centers in Package "sampSurf"*

Description

This routine is found within several of the plotting methods for objects of class "Tract" or it subclasses. In these methods, the arguments detailed below are passed as part of the "..." list. It can, however, also be used as a stand-alone function if desired, and applied to an existing plot of "Tract" objects, or objects that contain a "Tract" object, such as those of class "sampSurf". Lastly, it is used directly in plotting objects of class "InclusionZoneGrid" of subclasses.

Usage

```
gridCellEnhance(tract,
                gridLines = FALSE,
                gridCenters = FALSE,
                gridLineColor = .StemEnv$gridLineColor,
                gridCenterColor = .StemEnv$gridCenterColor,
                lwdGrid = 1, ...)
```

Arguments

`tract`	An object of class "Tract" or subclass.
`gridLines`	`TRUE`: plot grid lines; `FALSE`: no grid lines.
`gridCenters`	`TRUE`: plot grid cell centers; `FALSE`: no grid cell centers.
`gridLineColor`	An R color for the grid lines.
`gridCenterColor`	An R color for the grid cell centers.
`lwdGrid`	The line width specification in the form of the `lwd` `par` parameter, for the grid lines.
`...`	Arguments to be ignored at present.

Details

A valid plot must have been constructed first for the “Tract” object as the embellishments will be added to it. This should probably only be used on small areas, since the lines or centers will overlap for large areas or high resolution objects.

Value

Invisibly.

Author(s)

Jeffrey H. Gove

See Also

plot-methods

Examples

```
#
# apply it after plotting a sampling surface...
#
tr = Tract(c(x=50,y=50), cellSize=0.5)
btr = bufferedTract(10, tr)
ssSA = sampSurf(2, btr, iZone = 'sausageIZ', plotRadius=3,
       buttDiam=c(20,40))
plot(ssSA, axes=TRUE)
gridCellEnhance(ssSA@tract, gridLines=TRUE)

#
# on a tract only...
#
plot(btr, axes=TRUE, gridCenters=TRUE)
```

---

heapIZ *‘Heap’ Up a Sampling Surface*

Description

This generic has only one method as detailed in heapIZ-methods. It is used to “heap” an object of class "InclusionZoneGrid" onto one of class "Tract" to build an object of class "sampSurf." Note that the heaping would be applied to several objects of the first class to ultimately build a surface.

Usage

```
heapIZ(izgObject, tract, ...)
```

Arguments

`izgObject`	An object of class "InclusionZoneGrid" or subclass.
`tract`	An object of class "Tract".
`...`	See methods for other arguments.

Details

This method is really not meant to be used stand-alone. It is called within a `sampSurf` constuctor in order to 'heap' or 'pile' the values with inclusion zones to build a sampling surface. See these routines for details.

Value

An object of class "Tract" or subclass.

Author(s)

Jeffrey H. Gove

See Also

`heapIZ-methods`

Examples

```
#
# see sampSurf constructor source code for details.
#
```

`heapIZ-methods` *Methods for Function heapIZ in Package* **sampSurf**

Description

There is just one method of this generic. Please see below for its signature.

Methods

`signature(izgObject = "InclusionZoneGrid", tract = "Tract")` Heap or pile attribute values within an "InclusionZoneGrid" object onto a "Tract" object. . .

usage. . .

```
heapIZ(izgObject,
       tract,
       estimate = unlist(c(.StemEnv$puaEstimates, .StemEnv$ppEstimates)),
       ... )
```

- `izgObject`: An object of class "InclusionZoneGrid" or one of its subclasses.

- `tract`: An object of class “tract” or one of its subclasses.
- `estimate`: A character variable corresponding to the desired estimate attribute as found in the argument list of possibilities above.
- ... : Other arguments to be passed along–not used at present.

`hist` *Histogram of Various Object Attributes in Package* **sampSurf**

Description

Several different classes have methods for generating histograms of one or more of their class attributes. The different methods are described in `hist-methods`. They all extend R’s graphics::`hist` generic, and so allow all of the arguments that it does. In addition, some of the methods have other new arguments as well.

Details

The different attributes of each class that are plotted are described in `hist-methods`. Not all classes have a `hist` method, so it is also advisable to check which ones do, and what extra arguments might be available. For example, one can render a distribution of any number of different components of `downLog` attributes in a `downLogs` container collection.

Value

The same as is returned in the graphics::`hist` generic.

Author(s)

Jeffrey H. Gove

See Also

See methods and graphics::`hist` for details.

Examples

```
dlogs = downLogs(20)
hist(dlogs, logAttr='logVol')
```

`hist-methods` *Methods for* **graphics** *Function* `hist` *in Package* **sampSurf**

Description

The methods described here are for the `hist` generic as used in the **sampSurf** package. Several of the methods shown below have different arguments that customize each for a different class of objects found in this package. Method dispatch is based on the signature `x` argument.

Methods

Some arguments that are in the graphics::`hist` method are given default values in the calls below. Please refer to the graphics documentation for details.

`signature(x = "downLogs")` Creates a histogram of any one of several down log attributes in the container collection as specified by the `logAttr` argument.

usage...

```
hist(x,
    logAttr = c('logLen','buttDiam','topDiam','logAngle','solidType','logVol',
                'surfaceArea','coverageArea','biomass','carbon'),
     xlab = logAttr,
     main = NA,
     col = 'gray90',
     ... )
```

- `x`: An object that is of class “downLogs”.
- `logAttr`: Any of these can be used for the histogram.
- `...`: Other graphics arguments to be passed on to `hist`.

`signature(x = "standingTrees")` Creates a histogram of any one of several standing tree attributes in the container collection as specified by the `treeAttr` argument.

usage...

```
hist(x,
     treeAttr = c('height','buttDiam','dbh','topDiam','solidType','treeVol',
                 'surfaceArea','biomass','carbon'),
     xlab = treeAttr,
     main = NA,
     col = 'gray90',
     ... )
```

- `x`: An object that is of class “standingTrees”.
- `treeAttr`: Any of these can be used for the histogram.
- `...`: Other graphics arguments to be passed on to `hist`.

`signature(x = "izContainer")` Creates a histogram of inclusion zone areas for the objects in the container—this will work for all subclasses of “izContainer”.

usage...

```
hist(x,
     xlab = 'Inclusion Zone Area',
     main = NA,
     col = 'gray90',
     ... )
```

- `x`: An object that is a subclass of “izContainer”, such as “downLogIZs”.
- `...`: Other graphics arguments to be passed on to `hist`.

`signature(x = "InclusionZoneGrid")` Creates a histogram of the desired attribute for the object. Please note that it is not uncommon for the surface to be flat, which means all grid cells have the same value, so that a histogram is really meaningless. Use this on classes of objects where the surface (attribute estimate) varies within an individual inclusion zone.

usage...

```
hist(x,
     zeroTrunc = TRUE,
     estimate = 'volume',
     main = NA,
     xlab = NA,
     col = 'gray90',
     ... )
```

- `x`: An object that is a subclass of “InclusionZoneGrid”.
- `zeroTrunc`: `TRUE`: exclude background (zero) cells; `FALSE`: include them.
- `main`: The title is blank by default.
- `xlab`: The x label is taken from the `estimate` argument by default.
- `estimate`: The attribute of interest that the histogram will represent.
- `...`: Other graphics arguments to be passed on to `hist`.

`signature(x = "sampSurf")` Creates a histogram of the sampling distibution of the surface from the individual cell values.

usage...

```
hist(x,
     zeroTrunc = TRUE,
     xlab = x@estimate,
     main = NA,
     col = 'gray90',
     ... )
```

- `x`: An object that is of class “sampSurf”.
- `zeroTrunc`: `TRUE`: exclude all zero-valued background cells (default); `FALSE`: include all cells.
- `...`: Other graphics arguments to be passed on to `hist`.

Monte Carlo methods:

`signature(x = "mcsContainer")` Histograms of the relative error or volume variance over the collection are supported. Please note that this method applies to objects of class “antithetic-Container” as well.

usage...

```
hist(x,
     stat = c('relErrPct', 'volVar'),
     xlab = stat,
     main = NA,
     col = 'gray90',
     ... )
```

- `x`: An object that is of class “mcsContainer”.
- `stat`: Relative error in percent on volume (default), or variance on volume.
- `...`: Other graphics arguments to be passed on to `hist`.

`horizontalLineIZ` *Generate Objects of Class "`horizontalLineIZ`"*

Description

This is the generic function for class “horizontalLineIZ”. Please see the associated method in `horizontalLineIZ-methods` for more details.

Usage

```
horizontalLineIZ(standingTree, angleGauge, lineLength, ...)
```

Arguments

`standingTree`	Signature object of class "`standingTree`".
`angleGauge`	Signature object of class "`angleGauge`".
`lineLength`	The length of the line segment in the appropriate units (feet for “English”, meters for “metric”).
`...`	See methods.

Details

Since only one method exists for this generic, its signature arguments coincide with the above. Please see `horizontalLineIZ-methods` for more details.

Value

A valid object of class "`horizontalLineIZ`."

Author(s)

Jeffrey H. Gove

See Also

Class "`horizontalLineIZ`", and `horizontalLineIZ-methods`.

Examples

```
st = standingTree(dbh=25, solidType=2.5, height=15) #metric
ag = angleGauge(baf=4) #metric
hls.iz = horizontalLineIZ(st, angleGauge=ag, lineLength=50, orientation=345)
summary(hls.iz)
plot(hls.iz, axes=TRUE, cex=2)
```

`horizontalLineIZ-class`

Class `"horizontalLineIZ"`

Description

This class holds the inclusion zone definition for the horizontal line sampling method for standing trees.

Objects from the Class

Objects can be created by calls of the form `new("horizontalLineIZ",...)`. However, this is not recommended because the objects are fairly complex. Instead, one can use the object constructor `horizontalLineIZ` to create new objects.

Slots

This class adds several slots to the "`standingTreeIZ`" base class specification, please see the superclasses (see below) for more slot definitions.

`angleGauge`: Object of class `"angleGauge"`: A subclass object from the "`ArealSampling`" class. Please see the help for the "`angleGauge`" class for more information on the slots associated with angle gauge objects.

`lineSegment`: Object of class `"lineSegment"`: Also a subclass object from the "`ArealSampling`" class. Please see the help for the "`lineSegment`" class for more information on the slots associated with line segment objects.

`width`: Object of class `"numeric"`: This is the width of the inclusion zone in the appropriate units (feet for “English”, meters for “metric”).

`length`: Object of class `"numeric"`: This is the length of the inclusion zone (line segment) in appropriate units (feet for “English”, meters for “metric”).

`izPerim`: Object of class `"matrix"`: A matrix representation of the inclusion zone in homogeneous coordinates. This can be manipulated and plotted as desired for easy access to the inclusion zone where needed.

`area`: Object of class `"numeric"`: The area of the inclusion zone in the correct units.

`perimeter`: Object of class `"SpatialPolygons"`: The `"SpatialPolygons"` object corresponding to the perimeter of the inclusion zone.

`pgArea`: Object of class `"numeric"`: This is the area of the 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, and adjust if necessary the number of points defining the inclusion area perimeter.

Extends

Class "`standingTreeIZ`", directly.
Class "`InclusionZone`", by class "standingTreeIZ", distance 2.

Methods

izGrid `signature(izObject = "horizontalLineIZ",tract = "Tract")`: “InclusionZoneGrid” constructor

perimeter `signature(object = "horizontalLineIZ")`: Return the object perimeter

plot `signature(x = "horizontalLineIZ",y = "missing")`: Plot the object

summary `signature(object = "horizontalLineIZ")`: Object summary

Author(s)

Jeffrey H. Gove

References

T. G. Gregroire and H. T. Valentine. 2009. *Sampling strategies for natural resources and the environment*. Chapman and Hall/CRC, 496p.

“*The InclusionZone Class*” vignette.

See Also

See the "`standingTreeIZs`" container class and the object constructor `horizontalLineIZ`

Examples

```
showClass("horizontalLineIZ")
```

`horizontalLineIZ-methods`
Methods for `"horizontalLineIZ"` *object constuction in Package* **sampSurf**

Description

This is the one method for generic function `horizontalLineIZ` in Package **sampSurf** that allows for creation of objects of class "`horizontalLineIZ`."

Methods

`signature(standingTree = "standingTree", angleGauge = "angleGauge", lineLength = "numeric")`
Note that an object of class "lineSegment" will be constructed from the values passed in the `lineLength` and `orientation` arguments...

usage...

```
horizontalLineIZ(standingTree,
                 angleGauge,
                 lineLength,
                 orientation = 0,
           description = 'inclusion zone for horizontal line sampling method',
                 spID = paste('hls',.StemEnv$randomID(),sep=':'),
                 spUnits = CRS(projargs=as.character(NA)),
                 ... )
```

- `standingTree`: An object of class "standingTree" for which the inclusion zone is to be determined.
- `angleGauge`: An object of class "angleGauge".
- `lineLength`: The length of the line segment in the appropriate units (feet for "English", meters for "metric").
- `orientation`: The line orientation clockwise from *north* as an azimuth in degrees.
- `description`: A character vector description of the object.
- `spID`: A unique identifier that will be used in displaying the spatial polygon for the inclusion zone component of the object.
- `spUnits`: A valid CRS object specifying the Coordinate Reference System. This defaults to NA, which means you want to use your own user-defined system, say for a sample plot located in the field.
- `dots`: Arguments to be passed on.

---

horizontalPointCMCIZ *Generate Objects of Class "horizontalPointCMCIZ"*

Description

This generic is the base constructor for creating objects of class "horizontalPointCMCIZ". Please see the associated horizontalPointCMCIZ-methods method for more details.

Usage

```
horizontalPointCMCIZ(standingTree, angleGauge, ...)
```

Arguments

standingTree	Signature object of class "standingTree".
angleGauge	Signature object of class "angleGauge".
...	See associated method.

Details

It is very important to note both here and in the method documentation that the call to this constructor is the appropriate place to include any extra arguments that should be passed on to the `crudeMonteCarlo` constructor for control of subsampling within the inclusion zone.

Value

A valid object of class "`horizontalPointCMCIZ`".

Note

Please see the two vignettes cited above for more details on this and related classes.

In the following run, it is possible to look at the contents of the `mcsObj` slot for the object. Please note that any slot within this object other than those used to control the CMC subsampling (e.g., `segBnds`) will not be used to estimate volume for the tree. This object will be used to *control* subsampling at each grid cell location within the inclusion zone when applied to the `izGrid` constructor. Therefore, while the volume and variance in this slot's object are applicable to the tree, they are not related to the final estimate that will be derived within `sampSurf`.

Author(s)

Jeffrey H. Gove

References

Gove, J. H. 2013. Monte Carlo sampling methods in **sampSurf**. Package vignette.

Gove, J. H. 2013. The "InclusionZone" Class. **sampSurf** Package vignette.

See Also

Other Monte Carlo methods that work with horizontal point sampling include: `horizontalPointISIZ`, `horizontalPointCVIZ`, `criticalHeightIZ`, `importanceCHSIZ`, `antitheticICHSIZ`, `pairedAICHSIZ`.

Examples

```
#
# restrict CMC sampling between 10-20m height, with 2
# subsamples to be taken on the tree at each grid cell...
#
st = standingTree(dbh=50, solidType=4, height=25)
ag = angleGauge(baf=4)
cmchps.iz = horizontalPointCMCIZ(st, ag, segBnds=c(10,20),
            n.s=2)
summary(cmchps.iz)
plot(cmchps.iz)
```

horizontalPointCMCIZ-class *Class* "horizontalPointCMCIZ"

Description

This class allows one to perform crude Monte Carlo sampling within a given tree as a second stage sample, when the tree has been selected by horizontal point sampling in the first stage. Thus, the class definition is simply a combination of the "MonteCarloSamplingIZ" and "horizontalPointIZ" classes. No new slots have been added to this class, please see the above two classes for the totality of slot definitions in the class structure.

Objects from the Class

Objects can be created by calls of the form `new("horizontalPointCMCIZ",...)`. However, because the object is fairly complex, using the object constructor of the same name `horizontalPointCMCIZ` is the preferred method for creating valid objects of this class.

Slots

Please see the slot definitions in the "MonteCarloSamplingIZ" and "horizontalPointIZ" classes and their subclasses where applicable for all of the slot definitions.

In addition, the two vignettes below will be of some help as the class structures are also described therein.

Extends

Class "MonteCarloSamplingIZ", directly.
Class "horizontalPointIZ", directly.
Class "horizontalPointMonteCarloSamplingIZ", directly.
Class "circularPlotIZ", by class "horizontalPointIZ", distance 2.
Class "standingTreeIZ", by class "horizontalPointIZ", distance 3.
Class "InclusionZone", by class "horizontalPointIZ", distance 4.

Methods

No methods defined with class "horizontalPointCMCIZ" in the signature.

Author(s)

Jeffrey H. Gove

References

Gove, J. H. 2013. Monte Carlo sampling methods in **sampSurf**. Package vignette.

Gove, J. H. 2013. The “InclusionZone” Class. **sampSurf** Package vignette.

See Also

Other Monte Carlo methods that work with horizontal point sampling include: `horizontalPointISIZ`, `horizontalPointCVIZ`, `criticalHeightIZ`, `importanceCHSIZ`, `antitheticICHSIZ`, `pairedAICHSIZ`.

Examples

```
showClass("horizontalPointCMCIZ")
```

`horizontalPointCMCIZ-methods`

Methods for "`horizontalPointCMCIZ`*" object construction in Package* **sampSurf**

Description

The one constructor method that is available for creating valid objects of class "`horizontalPointCMCIZ`" is documented below. Please especially see the note on extra arguments that can be passed below.

Methods

`signature(standingTree = "standingTree", angleGauge = "angleGauge")`

usage...

```
horizontalPointCMCIZ(standingTree,
                     angleGauge,
                     antithetic = FALSE,
               description = 'Inclusion zone for horizontal point CMC sampling method',
                     spID = paste('hps.cmc',.StemEnv$randomID(),sep=':'),
                     spUnits = CRS(projargs=as.character(NA)),
                     ...)
```

- `standingTree`: An object of class "`standingTree`" for which the inclusion zone is to be determined.
- `angleGauge`: An object of class "`angleGauge`".
- `antithetic`: `TRUE`: use antithetic crude Monte Carlo sampling; `FALSE`: just normal CMC sampling.
- `description`: A character vector description of the object.
- `spID`: A unique identifier that will be used in displaying the spatial polygon for the inclusion zone component of the object.
- `spUnits`: A valid `CRS` object specifying the Coordinate Reference System. This defaults to `NA`, which means you want to use your own user-defined system, say for a sample plot located in the field.
- `...`: Arguments to be passed on, especially those that can be used in the call to the `crudeMonteCarlo` constructor for control of subsampling within the inclusion zone. This will take place when the grid cell estimates are established via CMC within the corresponding `izGrid` method.

horizontalPointCVIZ *Generate Objects of Class "horizontalPointCVIZ"*

Description

This generic is the base constructor for creating objects of class “horizontalPointCVIZ”. Please see the associated horizontalPointCVIZ-methods method for more details.

Usage

```
horizontalPointCVIZ(standingTree, angleGauge, ...)
```

Arguments

standingTree	Signature object of class "standingTree".
angleGauge	Signature object of class "angleGauge".
...	See associated method.

Details

It is very important to note both here and in the method documentation that the call to this constructor is the appropriate place to include any extra arguments that should be passed on to the controlVariate constructor for control of subsampling within the inclusion zone. In addition, different proxy functions can have extra arguments associated with them beyond the three required ones. Again, this is the place to pass any of these extra arguments to the proxy functions. Please see getProxy for documentation on the built-in proxy functions, and note the different arguments associated with each function.

Value

A valid object of class “horizontalPointCVIZ”.

Note

Please see the two vignettes cited above for more details on this and related classes.

In the following run, it is possible to look at the contents of the mcsObj slot for the object. Please note that any slot within this object other than those used to control the CV subsampling (e.g., segBnds) will not be used to estimate volume for the tree. This object will be used to *control* subsampling at each grid cell location within the inclusion zone when applied to the izGrid constructor. Therefore, while the volume and variance in this slot’s object are applicable to the tree, they are not related to the final estimate that will be derived within sampSurf.

Author(s)

Jeffrey H. Gove

References

Gove, J. H. 2013. Monte Carlo sampling methods in **sampSurf**. Package vignette.

Gove, J. H. 2013. The "InclusionZone" Class. **sampSurf** Package vignette.

See Also

Other Monte Carlo methods that work with horizontal point sampling include: horizontalPointCMCIZ, horizontalPointISIZ, criticalHeightIZ, importanceCHSIZ, antitheticICHSIZ, pairedAICHSIZ.

Examples

```
#
# restrict CV sampling between 10-20m height and use the
# "wbProxy" function with solid type = 0.9*4 (close to true taper);
# with 2 subsamples to be taken on the tree at each grid cell...
#
st = standingTree(dbh=50, solidType=4, height=25)
ag = angleGauge(baf=4)
cvhps.iz = horizontalPointCVIZ(st, ag, segBnds=c(10,20),
           proxy='wbProxy', solidTypeProxy=0.9, n.s=2)
summary(cvhps.iz)
plot(cvhps.iz)
```

---

horizontalPointCVIZ-class *Class* "horizontalPointCVIZ"

Description

This class allows one to perform control variate sampling within a given tree as a second stage sample, when the tree has been selected by horizontal point sampling in the first stage. Thus, the class definition is simply a combination of the "MonteCarloSamplingIZ" and "horizontalPointIZ" classes. No new slots have been added to this class, please see the above two classes for the totality of slot definitions in the class structure.

Objects from the Class

Objects can be created by calls of the form `new("horizontalPointCVIZ",...)`. However, because the object is fairly complex, using the object constructor of the same name horizontalPointCVIZ is the preferred method for creating valid objects of this class.

Slots

Please see the slot definitions in the "MonteCarloSamplingIZ" and "horizontalPointIZ" classes and their subclasses where applicable for all of the slot definitions.

In addition, the two vignettes below will be of some help as the class structures are also described therein.

Extends

Class "`MonteCarloSamplingIZ`", directly.
Class "`horizontalPointIZ`", directly.
Class "`horizontalPointMonteCarloSamplingIZ`", directly.
Class "`circularPlotIZ`", by class "horizontalPointIZ", distance 2.
Class "`standingTreeIZ`", by class "horizontalPointIZ", distance 3.
Class "`InclusionZone`", by class "horizontalPointIZ", distance 4.

Methods

No methods defined with class "horizontalPointCVIZ" in the signature.

Author(s)

Jeffrey H. Gove

References

Gove, J. H. 2013. Monte Carlo sampling methods in **sampSurf**. Package vignette.

Gove, J. H. 2013. The “InclusionZone” Class. **sampSurf** Package vignette.

See Also

Other Monte Carlo methods that work with horizontal point sampling include: `horizontalPointCMCIZ`, `horizontalPointISIZ`, `criticalHeightIZ`, `importanceCHSIZ`, `antitheticICHSIZ`, `pairedAICHSIZ`.

Examples

```
showClass("horizontalPointCVIZ")
```

`horizontalPointCVIZ-methods`
Methods for “`horizontalPointCVIZ`*” object construction in Package* **sampSurf**

Description

The one constructor method that is available for creating valid objects of class "`horizontalPointCVIZ`" is documented below. Please especially see the note on extra arguments that can be passed below.

Methods

`signature(standingTree = "standingTree", angleGauge = "angleGauge")`

usage...

```
horizontalPointCVIZ(standingTree,
                    angleGauge,
                    antithetic = FALSE,
             description = 'Inclusion zone: horizontal point with control variate sampling',
                    spID = paste('hps.cv',.StemEnv$randomID(),sep=':'),
                    spUnits = CRS(projargs=as.character(NA)),
                    ... )
```

- `standingTree`: An object of class "`standingTree`" for which the inclusion zone is to be determined.
- `angleGauge`: An object of class "`angleGauge`".
- `antithetic`: `TRUE`: use antithetic control variate sampling; `FALSE`: just normal control variate sampling.
- `description`: A character vector description of the object.
- `spID`: A unique identifier that will be used in displaying the spatial polygon for the inclusion zone component of the object.
- `spUnits`: A valid `CRS` object specifying the Coordinate Reference System. This defaults to `NA`, which means you want to use your own user-defined system, say for a sample plot located in the field.
- `...`: Arguments to be passed on, especially those that can be used in the call to the `controlVariate` constructor (and associated proxy function) for control of subsampling within the inclusion zone. This will take place when the grid cell estimates are established via CV sampling within the corresponding `izGrid` method.

`horizontalPointISIZ` *Generate Objects of Class "`horizontalPointISIZ`"*

Description

This generic is the base constructor for creating objects of class “`horizontalPointISIZ`”. Please see the associated `horizontalPointISIZ-methods` method for more details.

Usage

```
horizontalPointISIZ(standingTree, angleGauge, ...)
```

Arguments

`standingTree`	Signature object of class "`standingTree`".
`angleGauge`	Signature object of class "`angleGauge`".
`...`	See associated method.

Details

It is very important to note both here and in the method documentation that the call to this constructor is the appropriate place to include any extra arguments that should be passed on to the `importanceSampling` constructor for control of subsampling within the inclusion zone. In addition, different proxy functions can have extra arguments associated with them beyond the three required ones. Again, this is the place to pass any of these extra arguments to the proxy functions. Please see `getProxy` for documentation on the built-in proxy functions, and note the different arguments associated with each function.

Value

A valid object of class "`horizontalPointISIZ`".

Note

Please see the two vignettes cited above for more details on this and related classes.

In the following run, it is possible to look at the contents of the `mcsObj` slot for the object. Please note that any slot within this object other than those used to control the importance subsampling (e.g., `segBnds`) will not be used to estimate volume for the tree. This object will be used to *control* subsampling at each grid cell location within the inclusion zone when applied to the `izGrid` constructor. Therefore, while the volume and variance in this slot's object are applicable to the tree, they are not related to the final estimate that will be derived within `sampSurf`.

Author(s)

Jeffrey H. Gove

References

Gove, J. H. 2013. Monte Carlo sampling methods in **sampSurf**. Package vignette.

Gove, J. H. 2013. The "InclusionZone" Class. **sampSurf** Package vignette.

See Also

Other Monte Carlo methods that work with horizontal point sampling include: `horizontalPointCMCIZ`, `horizontalPointCVIZ`, `criticalHeightIZ`, `importanceCHSIZ`, `antitheticICHSIZ`, `pairedAICHSIZ`.

Examples

```
#
# restrict importance sampling between 10-20m height and use the
# "wbProxy" function with solid type = 0.9*4 (close to true taper);
# with 2 subsamples to be taken on the tree at each grid cell...
#
st = standingTree(dbh=50, solidType=4, height=25)
ag = angleGauge(baf=4)
ishps.iz = horizontalPointISIZ(st, ag, segBnds=c(10,20),
           proxy='wbProxy', solidTypeProxy=0.9, n.s=2)
summary(ishps.iz)
plot(ishps.iz)
```

horizontalPointISIZ-class   *Class* "horizontalPointISIZ"

Description

This class allows one to perform importance sampling within a given tree as a second stage sample, when the tree has been selected by horizontal point sampling in the first stage. Thus, the class definition is simply a combination of the "MonteCarloSamplingIZ" and "horizontalPointIZ" classes. No new slots have been added to this class, please see the above two classes for the totality of slot definitions in the class structure.

Objects from the Class

Objects can be created by calls of the form `new("horizontalPointISIZ",...)`. However, because the object is fairly complex, using the object constructor of the same name `horizontalPointISIZ` is the preferred method for creating valid objects of this class.

Slots

Please see the slot definitions in the "MonteCarloSamplingIZ" and "horizontalPointIZ" classes and their subclasses where applicable for all of the slot definitions.

In addition, the two vignettes below will be of some help as the class structures are also described therein.

Extends

Class "MonteCarloSamplingIZ", directly.
Class "horizontalPointIZ", directly.
Class "horizontalPointMonteCarloSamplingIZ", directly.
Class "circularPlotIZ", by class "horizontalPointIZ", distance 2.
Class "standingTreeIZ", by class "horizontalPointIZ", distance 3.
Class "InclusionZone", by class "horizontalPointIZ", distance 4.

Methods

No methods defined with class "horizontalPointISIZ" in the signature.

Author(s)

Jeffrey H. Gove

References

Gove, J. H. 2013. Monte Carlo sampling methods in **sampSurf**. Package vignette.

Gove, J. H. 2013. The “InclusionZone” Class. **sampSurf** Package vignette.

See Also

Other Monte Carlo methods that work with horizontal point sampling include: `horizontalPointCMCIZ`, `horizontalPointCVIZ`, `criticalHeightIZ`, `importanceCHSIZ`, `antitheticICHSIZ`, `pairedAICHSIZ`.

Examples

```
showClass("horizontalPointISIZ")
```

`horizontalPointISIZ-methods`

Methods for "`horizontalPointISIZ`*" object construction in Package* **sampSurf**

Description

The one constructor method that is available for creating valid objects of class "`horizontalPointISIZ`" is documented below. Please especially see the note on extra arguments that can be passed below.

Methods

`signature(standingTree = "standingTree", angleGauge = "angleGauge")`

usage...

```
horizontalPointISIZ(standingTree,
                    angleGauge,
                    antithetic = FALSE,
             description = 'Inclusion zone for horizontal point importance sampling method',
                    spID = paste('hps.is',.StemEnv$randomID(),sep=':'),
                    spUnits = CRS(projargs=as.character(NA)),
                    ...)
```

- `standingTree`: An object of class "`standingTree`" for which the inclusion zone is to be determined.
- `angleGauge`: An object of class "`angleGauge`".
- `antithetic`: `TRUE`: use antithetic importance sampling; `FALSE`: just normal importance sampling.
- `description`: A character vector description of the object.
- `spID`: A unique identifier that will be used in displaying the spatial polygon for the inclusion zone component of the object.
- `spUnits`: A valid `CRS` object specifying the Coordinate Reference System. This defaults to `NA`, which means you want to use your own user-defined system, say for a sample plot located in the field.
- `...`: Arguments to be passed on, especially those that can be used in the call to the `importanceSampling` constructor (and associated proxy function) for control of subsampling within the inclusion zone. This will take place when the grid cell estimates are established via IS within the corresponding `izGrid` method.

horizontalPointIZ *Generate Objects of Class "horizontalPointIZ"*

Description

This is the generic function for class “horizontalPointIZ”. Please see the associated method in horizontalPointIZ-methods for more details.

Usage

```
horizontalPointIZ(standingTree, angleGauge, ...)
```

Arguments

standingTree	Signature object of class "standingTree".
angleGauge	Signature object of class "angleGauge".
...	See methods.

Details

Since only one method exists for this generic, its signature arguments coincide with the above. Please see horizontalPointIZ-methods for more details.

Value

A valid object of class "horizontalPointIZ."

Author(s)

Jeffrey H. Gove

See Also

Class "horizontalPointIZ", and horizontalPointIZ-methods.

Examples

```
st = standingTree(dbh=50, solidType=4, height=25)
ag = angleGauge(baf=4)
iz.hps = horizontalPointIZ(st, ag)
summary(iz.hps)
plot(iz.hps, axes=TRUE, cex=2)
```

`horizontalPointIZ-class` *Class* `"horizontalPointIZ"`

Description

This class holds the inclusion zone definition for the horizontal point sampling method for standing trees.

Objects from the Class

Objects can be created by calls of the form `new("horizontalPointIZ",...)`. However, this is not recommended because the objects are fairly complex. Instead, one can use the object constructor `horizontalPointIZ` to create new objects.

Slots

This class adds one slot to the "`circularPlotIZ`" class specification, please see the superclasses (see below) for more definitions.

`angleGauge`**:** Object of class `"angleGauge"`: A subclass object from the "`ArealSampling`" class. Please see the help for the "`angleGauge`" class for more information on the slots associated with angle gauge objects.

Extends

Class "`circularPlotIZ`", directly.
Class "`standingTreeIZ`", by class "circularPlotIZ", distance 2.
Class "`InclusionZone`", by class "circularPlotIZ", distance 3.

Methods

summary `signature(object = "horizontalPointIZ")`: Show an object summary.

Author(s)

Jeffrey H. Gove

References

T. G. Gregroire and H. T. Valentine. 2009. *Sampling strategies for natural resources and the environment*. Chapman and Hall/CRC, 496p.

"*The InclusionZone Class*" vignette.

See Also

"`circularPlotIZ`", and the "`standingTreeIZs`" container class

Examples

```
showClass("horizontalPointIZ")
```

---

horizontalPointIZ-methods *Methods for* "horizontalPointIZ" *object construction in Package* **sampSurf**

Description

This is the one method for generic function horizontalPointIZ in Package 'sampSurf' that allows for creation of objects of class "horizontalPointIZ."

Methods

signature(standingTree = "standingTree", angleGauge = "angleGauge")

usage...

```
horizontalPointIZ(standingTree,
                  angleGauge,
            description = 'inclusion zone for horizontal point sampling method',
                  spID = paste('hps',.StemEnv$randomID(),sep=':'),
                  spUnits = CRS(projargs=as.character(NA)),
                  ...)
```

- standingTree: An object of class "standingTree" for which the inclusion zone is to be determined.
- angleGauge: An object of class "angleGauge".
- description: A character vector description of the object.
- spID: A unique identifier that will be used in displaying the spatial polygon for the circular plot component of the object.
- spUnits: A valid CRS object specifying the Coordinate Reference System. This defaults to NA, which means you want to use your own user-defined system, say for a sample plot located in the field.
- dots: Arguments to be passed on.

---

hybridDLPDSIZ *Generate Objects of Class "hybridDLPDSIZ"*

Description

This is the generic definition for generating objects of class "hybridDLPDSIZ." There is only one constructor method corresponding to this generic: hybridDLPDSIZ-methods.

Usage

```
hybridDLPDSIZ(downLog, pds, dls, ...)
```

Arguments

`downLog`	Signature object of class "`downLog`".
`pds`	Signature object of class "`perpendicularDistance`" containing the pertinent perpendicular distance sampling information.
`dls`	Signature object of class "dlsNumeric" which will accept either an object of class "`distanceLimited`" containing the pertinent distance limited sampling information or a "numeric" object specifying the distance limit.
`...`	See methods.

Details

Since only one method exists for this generic, its signature arguments coincide with the above definitions. Please see `hybridDLPDSIZ-methods` for more details.

Value

A valid object of class "`hybridDLPDSIZ`."

Author(s)

Jeffrey H. Gove

References

"The InclusionZone Class" vignette.

See Also

Class "`hybridDLPDSIZ`", and `hybridDLPDSIZ-methods`.

Examples

```
dl = downLog(buttDiam=15, solidType=4, logAngle=pi/3, logLen=10, units='English')
pdsEng = perpendicularDistance(kpds=6, units='English')
dlsEng = distanceLimited(2, units='English')
iz.hdlpds = hybridDLPDSIZ(dl, pds=pdsEng, dls=dlsEng)
iz.hdlpds
```

hybridDLPDSIZ-class *Class* "hybridDLPDSIZ"

Description

This class holds the inclusion zone definition for the 'hybrid distance limited perpendicular distance sampling' method for sampling down coarse woody debris (Ducey et. al 2012). This class is fairly complicated because there are three possibilities for the components of the inclusion zone. It is best to read *"The InclusionZone Class"* vignette, along with the references for more information—the why's and wherefore's are not presented here, only the class documentation.

Objects from the Class

Objects can be created by calls of the form `new("hybridDLPDSIZ",...)`. However, this is not recommended because the objects are fairly complex. Instead, one can use the object constructor `hybridDLPDSIZ` to create new objects.

Slots

This class is a direct descendent (subclass) of `"distanceLimitedPDSIZ"` and adds no new slots to that definition. Please see the `"distanceLimitedPDSIZ"` class definition for details.

Extends

Class `"distanceLimitedPDSIZ"`, directly.
Class `"perpendicularDistanceIZ"`, by class "distanceLimitedPDSIZ", distance 2.
Class `"downLogIZ"`, by class "distanceLimitedPDSIZ", distance 3.
Class `"pdsIZNull"`, by class "distanceLimitedPDSIZ", distance 3.
Class `"InclusionZone"`, by class "distanceLimitedPDSIZ", distance 4.

Methods

No methods defined with class "hybridDLPDSIZ" in the signature; instead see those defined for `"distanceLimitedPDSIZ"`.

Author(s)

Jeffrey H. Gove

References

Ducey, M. J., Williams, M. S., Gove, J. H., Roberge, S. and Kenning, R. S., 2012. Distance limited perpendicular distance sampling for coarse woody material: Theory and field results. *Forestry* (in review).

See Also

`downLogIZs` container class

Examples

```
showClass("hybridDLPDSIZ")
```

`hybridDLPDSIZ-methods` *Methods for Function* `hybridDLPDSIZ` *in Package* **sampSurf**

Description

This is the one method for generic function `hybridDLPDSIZ` in Package 'sampSurf' that allows for creation of objects of class "`hybridDLPDSIZ`."

Methods

`signature(downLog = "downLog", pds = "perpendicularDistance", dls = "dlsNumeric")`

usage...

```
hybridDLPDSIZ(downLog,
              pds,
              dls,
         description = 'inclusion zone for down log hybrid distance limited PDS',
              spID = paste('hdlpds',.StemEnv$randomID(),sep=':'),
              spUnits = CRS(projargs=as.character(NA)),
              pdsType = .StemEnv$pdsTypes,
              ... )
```

- `downLog`: An object of class "`downLog`" for which the inclusion zone is to be determined.
- `pds`: An object of class "`perpendicularDistance`" that supplies the information on the perpendicular distance method for constructing the inclusion zone.
- `dls`: An object of class "`distanceLimited`" that supplies the information on the distance limited method for constructing the inclusion zone. Alternatively a "numeric" distance limit that will be converted to a "distanceLimited" object.
- `description`: A character vector description of the object.
- `spID`: A unique identifier that will be used in displaying the spatial polygon for the inclusion zone component of the object.
- `spUnits`: A valid `CRS` object specifying the Coordinate Reference System. This defaults to `NA`, which means you want to use your own user-defined system, say for a sample plot located in the field.
- `pdsType`: A character string that specifies the type of perpendicular distance sampling used with regard to the selection (i.e., probability proportional to...) of the log. It can be one of "`volume`", "`surfaceArea`" or "`coverageArea`". (See also `.StemEnv$pdsTypes` for legal types.)
- `dots`: Arguments to be passed on.

importanceCHSIZ *Generate Objects of Class "importanceCHSIZ"*

Description

This is the generic function for class “importanceCHSIZ”. Please see the associated method in `importanceCHSIZ-methods` for more details.

Usage

```
importanceCHSIZ(standingTree, angleGauge, ...)
```

Arguments

`standingTree`	Signature object of class "`standingTree`".
`angleGauge`	Signature object of class "`angleGauge`".
`...`	See methods.

Details

Since only one method exists for this generic, its signature arguments coincide with the above. Please see `importanceCHSIZ-methods` for more details.

Value

A valid object of class "`importanceCHSIZ`."

Author(s)

Jeffrey H. Gove

See Also

Class "`importanceCHSIZ`", and `importanceCHSIZ-methods`.

Examples

```
st = standingTree(dbh=50, solidType=4, height=25)
ag = angleGauge(baf=4)
iz.ichs = importanceCHSIZ(st, ag)
summary(iz.ichs)
plot(iz.ichs, axes=TRUE, cex=2)
```

`importanceCHSIZ-class` *Class* `"importanceCHSIZ"`

Description

This class holds the inclusion zone definition for the importance sampling variant/protocol of critical height sampling for standing trees.

Objects from the Class

Objects can be created by calls of the form `new("importanceCHSIZ",...)`. However, this is not recommended because the objects are fairly complex. Instead, one can use the object constructor "`importanceCHSIZ`" to create new objects.

Slots

This class is a subclass of “criticalHeightIZ” (see below). It adds no new slots to the definition of that class, so please refer to it for the slot definitions.

Extends

Class "`criticalHeightIZ`", directly.
Class "`horizontalPointIZ`", by class "criticalHeightIZ", distance 2.
Class "`circularPlotIZ`", by class "criticalHeightIZ", distance 3.
Class "`standingTreeIZ`", by class "criticalHeightIZ", distance 4.
Class "`InclusionZone`", by class "criticalHeightIZ", distance 5.

Methods

izGrid `signature(izObject = "importanceCHSIZ",tract = "Tract")`: “InclusionZoneGrid” constructor

Author(s)

Jeffrey H. Gove

References

T. B. Lynch and J. H. Gove. 2013. An antithetic variate to facilitate upper-stem height measurements for critical height sampling and fixed-radius plot sampling with importance sampling. *Canadian Journal of Forest Research* (forthcoming).

“*The InclusionZone Class*” vignette.

See Also

See also the "`circularPlotIZ`", "`horizontalPointIZ`" and "`criticalHeightIZ`" classes, and the "`standingTreeIZs`" container class

Examples

```
showClass("importanceCHSIZ")
```

---

importanceCHSIZ-methods   *Methods for* "importanceCHSIZ" *object constuction in Package* **sampSurf**

Description

This documents the one method for generic function importanceCHSIZ in Package **sampSurf** that allows for creation of objects of class "importanceCHSIZ."

Methods

signature(standingTree = "standingTree", angleGauge = "angleGauge")

usage...

```
importanceCHSIZ(standingTree,
                angleGauge,
                referenceHeight = .StemEnv$referenceCHSIZ,
            description = 'inclusion zone for importance CH sampling method',
                spID = paste('ichs',.StemEnv$randomID(),sep=':'),
                spUnits = CRS(projargs=as.character(NA)),
                ...)
```

- standingTree: An object of class "standingTree" for which the inclusion zone is to be determined.
- angleGauge: An object of class "angleGauge".
- referenceHeight: The height on the stem at which the inclusion zone is to be determined. Currently the choices are “butt” (default) or “dbh”. These are found in the argument assignment .StemEnv$referenceCHSIZ above.
- description: A character vector description of the object.
- spID: A unique identifier that will be used in displaying the spatial polygon for the inclusion zone component of the object.
- spUnits: A valid CRS object specifying the Coordinate Reference System. This defaults to NA, which means you want to use your own user-defined system, say for a sample plot located in the field.
- dots: Arguments to be passed on.

`importanceSampling` *Generate Objects of Class "`importanceSampling`"*

Description

This generic has five methods, they are used to apply importance (sub) sampling to an individual `"Stem"` object, or collections of `"Stem"` objects. See `importanceSampling-methods` for details.

Usage

```
importanceSampling(object, ...)
```

Arguments

`object`	This is the signature argument, see the `importanceSampling-methods` for possible values.
`...`	Arguments that can be passed along to the proxy function.

Details

Briefly, with importance sampling for bole volume (or some segment of the bole) one uses a proxy taper function from which to draw samples and thereby concentrate the samples in the lower portion of the bole, where there is more volume and measurements are easier. The diferent built-in proxy functions and their use are detailed in the vignette cited below. In addition, one can supply one's own proxy function if desired.

Value

A valid object of class "`importanceSampling`" or "`mcsContainer`", depending on which method was used.

Author(s)

Jeffrey H. Gove

References

Gove, J. H. 2013. Monte Carlo sampling methods in **sampSurf**. Package vignette.

See Also

See `importanceSampling-methods` for methods. Other similar generics for Monte Carlo methods include: `crudeMonteCarlo`, `controlVariate`, `antitheticSampling`.

Examples

```
#
# estimate volume between 10 and 15 m, using 5 random heights...
#
sTree = standingTree(dbh = 40, topDiam = 0, height = 20, solidType = 2.8)
sTree.is = importanceSampling(sTree, n.s = 5, segBnds = c(10,15), startSeed = 114,
           proxy = 'wbProxy', solidTypeProxy = 2.5)
sTree.is
```

---

importanceSampling-class

Class "importanceSampling"

Description

This is the class definition that allows for the application of importance sampling to downLog or standingTree objects. Examples of the class usage can be found in the Monte Carlo sampling vignette referenced below.

Objects from the Class

Objects can be created by calls of the form new("importanceSampling",...). However, an object constructor of the same name, importanceSampling, has been provided and is the preferred method for creating objects that are ensured to be valid.

Slots

Please note that diameters below are presumed to be in the *same* units as length, i.e., meters for "metric", and feet for "English" units. Cross-sectional areas are in compatible units.

In addition to the slots provided by the virtual superclass "MonteCarloSampling", and the immediate superclass "crudeMonteCarlo", the following slots are represented (please see the superclasses for shared slot definitions)...

proxy**:** Object of class "character": The name of the proxy function used. For built-in proxies, choose one of "gvProxy" or "wbProxy". These are explained in detail in the vignette cited below.

Extends

Class "crudeMonteCarlo", directly.
Class "MonteCarloSampling", by class "crudeMonteCarlo", distance 2.

Methods

No new methods defined with class "importanceSampling" in the signature. However, various methods such as summary and plot are available through inheritance.

Author(s)

Jeffrey H. Gove

References

Gove, J. H. 2013. Monte Carlo sampling methods in **sampSurf**. Package vignette.

See Also

`MonteCarloSampling`, `crudeMonteCarlo`, `controlVariate`, `antitheticSampling`.

Examples

```
showClass("importanceSampling")
```

`importanceSampling-methods`

Methods for `importanceSampling` *object construction in Package* **sampSurf**

Description

The methods described below for the construction of objects of class "`importanceSampling`" fall into two categories as follows...

1. The `object` passed is a "`Stem`" subclass object: importance sampling is applied to the individual stem.

2. The `object` passed is a collection ("`StemContainer`") of "`Stem`" subclass objects: importance sampling is applied to each stem in the collection based on the other arguments passed.

In adition, there is a separate method for the case when `object` is of class "`list`". This is the base constructor that really performs all of the importance sampling code on the individual stems. The other methods are simply wrappers that call this method. The `list` constructor is detailed below for completeness; however, please do not use it, pass one of the other types of objects instead to use one of the other methods, this will ensure proper results inasmuch as is possible.

Methods

Each of the methods has either the same argument list, or a subset of arguments that correspond to the `list` signature method. Refer to that method for any arguments that have a universal interpretation over all methods.

`signature(object = "downLog")`

usage...

```
importanceSampling(object,
                   segBnds = c(low = 0,  up = object@logLen),
                   n.s = 1,
                   startSeed = NA,
                   u.s = NA,
                   proxy = 'gvProxy',
                   alphaLevel = 0.05,
                   description = 'Importance Sampling',
                   ... )
```

- `object`: An object of class "`downLog`".

`signature(object = "downLogs")`

usage. . .

```
importanceSampling(object,
                   segBnds = NULL,
                   n.s = 1,
                   startSeed = NA,
                   u.s = NA,
                   proxy = 'gvProxy',
                   alphaLevel = 0.05,
                   description = 'Importance Sampling',
                   ... )
```

- `object`: A container object of class "`downLogs`" with one or more "`downLog`" objects.
- `segBnds`: The segment bounds, see the definition for the `list` method. *Note:* These bounds are used for all logs in the collection, so it is up to you to make sure they are legal for each log.
- `startSeed`: By default, the stream is started using this seed (see below for the `list` method) and the current random number stream is continued for each log in the collection. This results in a *different* set of random numbers for each log (but all keyed off this starting value).
- `u.s`: If this is `NULL` or `NA`, then the `n.s` and `startSeed` combination are used as described below for the `list` method. However, if this is a vector, then it is applied to each log. Therefore, the *same* set of random numbers will be applied to *each* log in the collection.

`signature(object = "list")` Please do not use this method directly, use one of the others documented here that will ultimately call this method.

usage. . .

```
importanceSampling(object,
                   segBnds = c(low = 0, up = object$height),
                   n.s = 1,
                   startSeed = NA,
                   u.s = NA,
                   proxy = 'gvProxy',
                   alphaLevel = 0.05,
                   controlVariate = FALSE,
                   description = 'Monte Carlo Sampling',
                   ... )
```

- `object`: An object of class "`list`".
- `segBnds`: A vector of length two giving the lower and upper height/length bounds for volume estimation within the bole. These bounds correspond to the limits of integration along the bole. If either of the bounds are `NULL` or `NA`, the entire bole is used (default).
- `n.s`: The number of sampled heights desired within `segBnds` for volume estimation.
- `startSeed`: The scalar seed for the random number generator used in the call to the class constructor. Please see the documentation in `initRandomSeed` for possible values and their meaning.
- `u.s`: The uniform random numbers used in selecting the sampling points along the bole. If this is either `NULL` or `NA`, then `n.s` and `startSeed` will be used to determine the random numbers. If this is a numeric vector, then `n.s` is set to its length, and `u.s` is used as the random number stream. No checking is done on the bounds of the numbers so *be careful* if using the latter option. It is most useful in antithetic sampling where the 1-`u.s` stream is used (automatically).
- `proxy`: A character name specifying the proxy function to be used in importance sampling. See the vignette referenced in the generic for details.
- `alphaLevel`: The two-tailed alpha-level for confidence interval construction.
- `controlVariate`: `TRUE`: use control variate sampling; `FALSE`: either crude Monte Carlo or importance sampling, depending on the `proxy` passed.
- `description`: A character vector description of the object.
- `...`: Arguments to be passed on to the proxy function. For collections, these apply to each stem in the collection.

`signature(object = "standingTree")`

usage...

```
importanceSampling(object,
                   segBnds = c(low = 0,  up = object@height),
                   n.s = 1,
                   startSeed = NA,
                   u.s = NA,
                   proxy = 'gvProxy',
                   alphaLevel = 0.05,
                   description = 'Importance Sampling',
                   ... )
```

- `object`: An object of class "`standingTree`".

`signature(object = "standingTrees")`

usage...

```
importanceSampling(object,
                   segBnds = NULL,
                   n.s = 1,
                   startSeed = NA,
                   u.s = NA,
                   proxy = 'gvProxy',
                   alphaLevel = 0.05,
                   description = 'Importance Sampling',
                   ... )
```

- object: A container object of class "standingTrees" with one or more "standingTree" objects.
- segBnds: The segment bounds, see the definition for the list method. *Note:* These bounds are used for all trees in the collection, so it is up to you to make sure they are legal for each tree.
- startSeed: By default, the stream is started using this seed (see below for the list method) and the current random number stream is continued for each tree in the collection. This results in a *different* set of random numbers for each tree (but all keyed off this starting value).
- u.s: If this is NULL or NA, then the n.s and startSeed combination are used as described below for the list method. However, if this is a vector, then it is applied to each tree. Therefore, the *same* set of random numbers will be applied to *each* tree in the collection.

InclusionZone-class *Class "InclusionZone"*

Description

This is a virtual class that defines the root structure for inclusion zones used in areal sampling methods. In general, subclasses will use objects generated from both "ArealSampling" and "Stem" classes to form the functional object. Therefore, the object will be tied to the actual "Stem" object and sampling method in use. Hence, individual “InclusionZone” objects will be made for each stem (i.e., downed log or standing tree) in a collection or population when the final objective is a sampling surface simulation.

Objects from the Class

A virtual Class: No objects may be created from it.

Slots

description: Object of class "character": Some descriptive text about this class.

units: Object of class "character": A character string specifying the units of measure. Legal values are “English” and “metric.”

bbox: Object of class "matrix": The bounding box enclosing the overall object and its inclusion zone.

spUnits: Object of class "CRS": A valid string of class "CRS" denoting the spatial units coordinate system as in package **sp**.

puaBlowup: Object of class "numeric": The per unit area blowup or expansion factor, based on an acre or hectare depending upon the units of measure.

puaEstimates: Object of class "list": A list of estimates for per unit area quantities associated with the “Stem” component of the object.

userExtra: Object of class "ANY": This can be anything else the user wants to include with the object. Often it would be in the form of a list object. (This slot may be removed in the future.)

Methods

bbox `signature(obj = "InclusionZone")`: Return the bounding box

plot `signature(x = "InclusionZone", y = "missing")`: Plot the object

show `signature(object = "InclusionZone")`: Show the object

summary `signature(object = "InclusionZone")`: Object summary

Author(s)

Jeffrey H. Gove

References

"*The InclusionZone Class*" vignette.

See Also

For subclasses associated with down logs, see: `downLogIZ`, `standUpIZ`, `chainSawIZ`, `sausageIZ`, `pointRelascopeIZ`, `perpendicularDistanceIZ`, `omnibusPDSIZ`, `distanceLimitedPDSIZ`, `omnibusDLPDSIZ`, `distanceLimitedIZ`, `distanceLimitedMCIZ`, and the `downLogIZs` container class. \

For subclasses associated with standing trees, see: `standingTreeIZ`, `circularPlotIZ`, `horizontalPointIZ`, `horizontalLineIZ`, and the `standingTreeIZs` container class.

Examples

```
showClass("InclusionZone")
```

`InclusionZoneGrid-class`

Class "InclusionZoneGrid"

Description

This class really provides the functional building block for piecing inclusion zones together into the final sampling surface object. These functions should not be required by the casual user of this package, but will be required for writing extensions for new methods as they are added. Basically, the class defines a subgrid that just encapsulates an "`InclusionZone`" object. Attribute values (e.g., volume) are assigned to the subgrid cells falling within the inclusion zone of the object, with zero values outside the zone. This subgrid is aligned to the overall tract grid in the constructor, and upon return, can be "heaped" or "piled" additively onto the "`Tract`" object to develop the sampling surface.

In order to make a complete enumeration of the chainsaw method under Protocol 1 of Gove and Van Deusen (2011), we need an extension to this class. The subclass "`csFullInclusionZoneGrid`" should be consulted for such tasks.

Objects from the Class

Objects can be created by calls of the form `new("InclusionZoneGrid",...)`. However, this is not recommended because the objects are fairly complex. Instead, one can use the object constructor `izGrid` to create new objects. More details are found in *"The InclusionZoneGrid Class"* vignette.

Slots

`description`**:** Object of class `"character"`: A description of the object if desired.

`iz`**:** Object of class `"InclusionZone"`: An object of one of the "InclusionZone" subclasses such as "`standUpIZ`."

`grid`**:** Object of class `"RasterLayer"`: This is the minimal encompassing grid that must be aligned to a "Tract" object's extents in order to build a sampling surface.

`data`**:** Object of class `"data.frame"`: A data frame holding the values for each of the per unit area estimates available in the "InclusionZone" object in the columns, with rows matching grid cells.

`bbox`**:** Object of class `"matrix"`: 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 (e.g., the sausage method), but other times it does not (e.g., chainsaw method). This is used primarily in graphing the object.

Methods

bbox `signature(obj = "InclusionZoneGrid")`: Get encompasing bbox

heapIZ `signature(izgObject = "InclusionZoneGrid",tract = "Tract")`: Heap to form a sampling surface

plot `signature(x = "InclusionZoneGrid",y = "missing")`: Plot the grid

show `signature(object = "InclusionZoneGrid")`: Show object information

summary `signature(object = "InclusionZoneGrid")`: Show grid summary

Author(s)

Jeffrey H. Gove

References

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

See Also

`InclusionZone`, `Stem`, `Tract`, `sampSurf`, `csFullInclusionZoneGrid`, `mirageInclusionZoneGrid`

Examples

```
showClass("InclusionZoneGrid")
```

`initRandomSeed` *Initialize the Random Number Seed*

Description

This routine will initialize `.Random.seed` to the desired seed, continue the current seed, or if none exists, set via the clock time.

Usage

```
initRandomSeed(startSeed = NA, runQuiet = TRUE, ...)
```

Arguments

`startSeed`	`NULL` or `NA`: default to current stream, unless `.Random.seed` does not exit, in which case the seed is set from the system clock time; otherwise, specify a scalar seed value start a new stream.
`runQuiet`	`TRUE`: no feedback; `FALSE`: show some results.
`...`	Not used, for gobbling arguments only.

Details

The default uniform RNG in R is“Mersenne-Twister” which produces a ‘seed set’ when initialized. This seed set seems to be initialized to postion 624 so that the next draw will change the seed set (and hence the position to 1). Subsequently it will take 624 more draws to change the seed set, to a new one &c. So what is returned by default from this function is a vector of length 626.

Value

The newly set seed or current seed invisibly.

Author(s)

Jeffrey H. Gove

See Also

`.Random.seed`, `set.seed`

Examples

```
## Not run:
initRandomSeed()
initRandomSeed(12)
rm(.Random.seed)
initRandomSeed(NULL)

## End(Not run)
```

izContainer *Common Setup for Subclass Object Creation*

Description

This function simply does some background work that each of "downLogIZs" and "standingTreeIZs" subclasses have in common for object creation. It is *not* meant to be called as a general function by the casual user.

Usage

```
izContainer(object, ...)
```

Arguments

`object`	the one signature object, a `list`.
`...`	Just gobbled for now.

Details

See the returned value below. Note that this is pre-object creation, so it does not make sense to have a bbox method to calculate the overall bounding box. A bbox method based on a `list` object is also out of the question because the list could contain anything; in this routine the list passed must only contain valid objects of class "InclusionZone" (i.e., one of its subclasses).

Value

Currently returns just the overall bounding box of all the objects in the `list`. However, it may acquire more functionality later on.

Author(s)

Jeffrey H. Gove

See Also

"izContainer","downLogIZs", "standingTreeIZs" izContainer-class   *Class* "izContainer"

Description

This is a virtual class whose purpose is to define a *container* class for objects of class "InclusionZone." It provides slots and validity checks that would be common to all subclasses. Please note that this is not a completely functional container class in the traditional sense at present as it does not have replacement, deletion, or addition functions. Please see "*The InclusionZone Class*" vignette for more information.

Objects from the Class

This is a virtual class, so no objects can be created of this type. However, subclass objects can be created by their constructors (see, e.g., "downLogIZs").

Slots

`iZones`**:** Object of class `"list"`: This slot holds a list of objects that all correspond to the same subclass of the "InclusionZone" class. In other words, each item must have been generated from the same sampling method; i.e., all of class "sausageIZ," for example.

`units`**:** Object of class `"character"`: The same units as the "Stem" class objects in the collection; note that they must all be measured in the same units.

`bbox`**:** Object of class `"matrix"`: The overall bounding box for the collection used in plotting.

`description`**:** Object of class `"character"`: A description of the collection.

Methods

bbox `signature(obj = "izContainer")`: Return the minimal bounding box matrix.

hist `signature(x = "izContainer")`: Histogram for population of objects.

perimeter `signature(object = "izContainer")`: Graphical perimeter corresponding to the mnimal bounding box.

plot `signature(x = "izContainer",y = "missing")`: Graphical plot of the collection.

sampSurf `signature(object = "izContainer",tract = "Tract")`: Create a sampling surface for the collection on a given "Tract".

show `signature(object = "izContainer")`: Print some information about the collection.

summary `signature(object = "izContainer")`: A summary of the collection.

Author(s)

Jeffrey H. Gove

See Also

`downLogIZs`, `standingTreeIZs`

Examples

```
showClass("izContainer")
```

---

izGrid *Generate Objects of Class "InclusionZoneGrid"*

Description

This is the generic definition for generating objects of class "InclusionZoneGrid." There are several methods corresponding to this generic that may be found in `izGrid-methods`. Most users will not require the use of these methods in a primary use sense, they are more helper methods in that they are necessary to build the final sampling surface. More details and examples are found in *"The InclusionZoneGrid Class"* vignette.

Usage

```
izGrid(izObject, tract, ...)
```

Arguments

`izObject`	Signature object of a subclass of "`InclusionZone`".
`tract`	Signature object of class "`Tract`".
`...`	See methods.

Details

There are any number of details concerning this class that are described in the vignette mentioned above. It provides the basic idea behind the class and the use of the methods for each specific sampling protocol. Because these methods will not necessarily be useful to the casual user, we leave the details to that document.

Value

A valid object of class "`InclusionZoneGrid`."

Author(s)

Jeffrey H. Gove

See Also

`InclusionZone`, `Stem`, `Tract`, `sampSurf`, `izGridConstruct`, `izGrid-methods`

Examples

```
tr = Tract(c(x=20,y=20), cellSize=0.5)
btr = bufferedTract(5, tr)
dlogs = downLogs(1, btr@bufferRect)
sup = standUpIZ(dlogs@logs$log.1, 2)
izgSU = izGrid(sup, btr)
plot(izgSU, axes=TRUE)
izgSU
```

`izGrid-methods` *Methods for "InclusionZoneGrid" object construction in Package 'sampSurf'*

Description

The following methods will construct valid objects of class "`InclusionZoneGrid`" (or subclass) for different sampling protocols or methods based on the "`InclusionZone`" of the first signature argument. Please see *"The InclusionZoneGrid Class"* vignette for more details and examples.

Methods

The following constructors all share the formal arguments (after the two signature arguments) defined in the `standUpIZ` method. Any additional arguments are listed where appropriate. The `description` argument, will of course be different for each method.

In addition, in what follows the constructor methods are organized by those applicable to sampling methods associated with *(i)* down logs, *(ii)* standing trees, and *(iii)* other.

Down log constructors. . . :

`signature(izObject = "standUpIZ", tract = "Tract")` This constructor is for the stand-up protocol.

usage. . .

```
izGrid(izObject,
       tract,
       description = 'standUpIZ inclusion zone grid object',
       wholeIZ = TRUE,
       ... )
```

- `izObject`: An object of class "standUpIZ."
- `tract`: An object of class "Tract" or subclass.
- `description`: A description of the object as a character string.
- `wholeIZ`: `TRUE`: make a background grid covering the entire object including the stem and the inclusion zone; `FALSE`: make the grid cover just the inclusion zone.
- ... : Other arguments to be passed along as described in the final constructor method below (matrix signature argumnent).

`signature(izObject = "sausageIZ", tract = "Tract")` This constructor is for the sausage method of sampling down logs. Everything is the same here as for the stand-up method above, with the exception of the first argument, which must be an object of class "`sausageIZ`".

`signature(izObject = "chainSawIZ", tract = "Tract")` This constructor is for the chainsaw method of sampling down logs. Everything is the same here as for the stand-up method above, with the exception of the first argument, which must be an object of class "`chainSawIZ`". Now, one important thing to keep in mind is that the inclusion zone for the chainsaw method is a point (within a grid cell), so setting `wholeIZ=FALSE` here will give just the one grid cell that corresponds to the plot center point as a background grid. *Note:* If you want to enumerate all cells withing a sausage-shaped inclusion zone under the chainsaw method, see instead the "`csFullInclusionZoneGrid`" class.

`signature(izObject = "fullChainSawIZ", tract = "Tract")` This constructor is for the full chainsaw method of sampling down logs. Everything is the same here as for the stand-up method above, with the exception of the first argument, which must be an object of class "`fullChainSawIZ`". Objects created by this constructor are of class "`csFullInclusionZoneGrid`," which is a subclass of "`InclusionZoneGrid`"

`signature(izObject = "pointRelascopeIZ", tract = "Tract")` This constructor is for the point relascope method of sampling down logs. Everything is the same here as for the stand-up method above, with the exception of the first argument, which must be an object of class "`pointRelascopeIZ`".

`signature(izObject = "perpendicularDistanceIZ", tract = "Tract")` This constructor is for the canonical PDS method of sampling down logs. Everything is the same here as for the stand-up method above, with the exception of the first argument, which must be an object of class "`perpendicularDistanceIZ`".

`signature(izObject = "omnibusPDSIZ", tract = "Tract")` This constructor is for the omnibus PDS method of sampling down logs. Everything is the same here as for the stand-up method above, with the exception of the first argument, which must be an object of class "`omnibusPDSIZ`" and the addition of the following argument…

- `runQuiet`: `TRUE`: no feedback on progress; `FALSE`: some feedback on progress

`signature(izObject = "distanceLimitedPDSIZ", tract = "Tract")` This constructor is for the distance limited PDS (canonical, omnibus or hybrid) method of sampling down logs. Everything is the same here as for the "omnibusPDSIZ" method above, with the exception of the first argument, which must be an object of class "`distanceLimitedPDSIZ`". This method will also work for objects of class "`omnibusDLPDSIZ`" and "`hybridDLPDSIZ`".

`signature(izObject = "distanceLimitedIZ", tract = "Tract")` This constructor is for the distance limited method of sampling down logs. Everything is the same here as for the "omnibusPDSIZ" method above, with the exception of the first argument, which must be an object of class "`distanceLimitedIZ`".

`signature(izObject = "distanceLimitedMCIZ", tract = "Tract")` This constructor is for the distance limited Monte Carlo method of sampling down logs. Everything is the same here as for the "distanceLimitedIZ" method above, with the exception of the first argument, which must be an object of class "`distanceLimitedMCIZ`".

Standing tree constructors…:

`signature(izObject = "circularPlotIZ", tract = "Tract")` This constructor is for the circular plot and is also used directly for "`horizontalPointIZ`" objects.

`signature(izObject = "horizontalLineIZ", tract = "Tract")` This constructor is for the horizontal line sampling ("`horizontalLineIZ`") method for standing trees.

`signature(izObject = "horizontalPointMonteCarloSamplingIZ", tract = "Tract")` This constructor is for Monte Carlo subsampling coupled with the horizontal point sampling method for standing trees. It is applicable to objects of class "horizontalPointCMCIZ", "horizontalPointISIZ", and "horizontalPointCVIZ".

`signature(izObject = "criticalHeightIZ", tract = "Tract")` This constructor is for the critical height sampling ("criticalHeightIZ") method for standing trees.

`signature(izObject = "importanceCHSIZ", tract = "Tract")` This constructor is for the importance sampling variant to critical height sampling ("importanceCHSIZ") method for standing trees.

`signature(izObject = "antitheticICHSIZ", tract = "Tract")` This constructor is for the antithetic importance sampling variant to critical height sampling ("antitheticICHSIZ") method for standing trees.

`signature(izObject = "pairedAICHSIZ", tract = "Tract")` This constructor is for the paired antithetic importance sampling variant to critical height sampling ("pairedAICHSIZ") method for standing trees.

Other constructors. . . :

`signature(izObject = "matrix", tract = "Tract")` There is no need to use this constructor unless you are perhaps developing a new sampling method for the package. It is a helper function with some standard code that is required for most methods. It is used to construct the encompassing background grid object.

usage. . .

```
izGrid(izObject,
       tract,
       data = 0,
       useCrop = TRUE,
       ... )
```

- `izObject`: This must be a bbox matrix taken directly from the respective "InclusionZone" object.
- `tract`: An object of class "Tract" or subclass.
- `data`: A scalar value for the background cells in the encompassing bounding grid that will be created.
- `useCrop`: `TRUE`: use the **raster** crop function; `FALSE`: use the alternative method (see the source code for details).

`izGridConstruct` *Common code for constructing objects of class “InclusionZoneGrid”*

Description

This routine is called by all izGrid methods that assign a constant surface within an object’s inclusion zone. It should not, therefore, be used for sampling protocols such as the chainsaw method, where a variable surface is computed within the inclusion zone.

Usage

```
izGridConstruct(izObject, tract,
                description = "sausageIZ inclusion zone grid object",
                wholeIZ = TRUE, ...)
```

Arguments

`izObject`	An object of class "`InclusionZone`", with the exceptions stated above.
`tract`	An object of class "Tract" or subclass.
`description`	A description of the object as a character string.
`wholeIZ`	`TRUE`: make a background grid covering the entire object including the stem and the inclusion zone; `FALSE`: make the grid cover just the inclusion zone.
`...`	Other arguments to be passed along (not currently used).

Details

This routine should never need to be called by the user at the command line. It is meant to be used with calls set up in the `izGrid-methods`. It simply collects code that is common to all methods that assign a constant surface height with a given inclusion zone. The inclusion zone can be of any shape commonly encountered in areal sampling methods. As long as it is a legal "`InclusionZone`" object, it should work.

Value

A valid object of class "`InclusionZoneGrid`."

Author(s)

Jeffrey H. Gove

See Also

`izGrid-methods`, class "`InclusionZoneGrid`"

`izGridMirage` *Generate Objects of Class "`mirageInclusionZoneGrid`"*

Description

This generic has one method that will create objects of class “mirageInclusionZoneGrid”. The normal use is through generation of a sampling surface, but the routine can be called as is to produce results for individual stem inclusion zones on a “mirageTract”.

Usage

```
izGridMirage(izObject, tract, ...)
```

Arguments

`izObject`	Signature object of a subclass of "`InclusionZone`".
`tract`	Signature object of class "`mirageTract`".
`...`	See `izGridMirage-methods`.

Details

Objects of class “mirageInclusionZoneGrid” are farily complex and can hold information on external overlap areas is applicable. There is one method for this generic that can be used to create “mirageInclusionZoneGrid” object as described in `izGridMirage-methods`.

Value

A valid object of class "`mirageInclusionZoneGrid`."

Author(s)

Jeffrey H. Gove

References

See the “Mirage Method” and “InclusionZoneGrid” vignettes for examples.

See Also

`izGrid`, `InclusionZoneGrid`,`mirageInclusionZoneGrid`, `sampSurf`

Examples

```
tr = Tract(c(x=20,y=20), cellSize=1)
mtr = mirageTract(tr)
st = standingTree(centerOffset=c(x=16,y=16))
ag = angleGauge(4)     #4=baf
hps.iz = horizontalPointIZ(st, angleGauge=ag)
hps.izg = izGridMirage(hps.iz, mtr, truncateOverlap=FALSE)
plot(hps.izg, tract=mtr)
#now show external grid estimates...
plot(hps.izg, tract=mtr, showExtended = TRUE)
```

`izGridMirage-methods` *Methods for Function* `izGridMirage` *in Package* **sampSurf**

Description

There is one method for generic function `izGridMirage` that will create objects of class "`mirageInclusionZoneGrid`".

Methods

`signature(izObject = "InclusionZone", tract = "mirageTract")` Any arguments not defined below have the same definitions as those for the `izGrid` constructor (which is called first for the given `izObject`).

usage. . .

```
izGridMirage(izObject,
             tract,
             description = 'izGrid object using mirage',
             wholeIZ = TRUE,
             truncateOverlap = FALSE,
             ... )
```

- `izObject`: An object of class "InclusionZone"
- `tract`: An object of class "mirageTract".
- `description`: A description of the object as a character string.
- `truncateOverlap`: `TRUE`: truncate the estimates to zero for cells external to the tract; `FALSE`: keep the estimates. This only applies to the object in the `izGrid.extended` slot of the object. If `TRUE`, the extended grid will still be there, only with background cells rather than estimates. If `FALSE`, the estimates for the cells in the extended grid will be preserved.
- ... : Other arguments to be passed along as described in the `izGrid-methods` constructor documentation.

`lineSegment` *Generate Objects of Class "`lineSegment`"*

Description

This generic function has only one method (`lineSegment-methods`) used as a constructor function for objects that are of class "`lineSegment`". This method should be used in preference to `new` to insure a valid object.

Usage

```
lineSegment(length, orientation, ...)
```

Arguments

`length`	The length of the line segment in the appropriate units (feet for "English" or meters for "metric").
`orientation`	The line orientation clockwise from *north* as an azimuth in degrees.
`...`	Arguments that are defined in `lineSegment-methods`

Details

Only one method currently exists for object generation. Its arguments are documented in `lineSegment-methods`.

Value

A valid object of class "`lineSegment`"

Author(s)

Jeffrey H. Gove

See Also

`lineSegment-methods`

Examples

```
# a one chain segment...
ls = lineSegment(length=66, orientation=45, centerPoint=c(x=100, y=80), units='English')
summary(ls)
plot(ls, showLineCenter=TRUE, cex=2)
```

`lineSegment-class` *Class* `"lineSegment"`*: sample line segments*

Description

A subclass of "`ArealSampling`" that can be used to create line segment objects for use in methods where sampling is conducted along a line.

Objects from the Class

Objects can be created by calls of the form `new("lineSegment",...)`; however, this is not recommended due to the necessity to have the spatial representation correct. Preferably, one should use the `lineSegment` constructor for this class.

Slots

`orientation`**:** Object of class `"numeric"`: The orientation of the line segment clockwise from *north* as an azimuth in radians. Please note that this is different from the `logAngle` slot in "`downLog`" objects, which is defined counter-clockwise from due east, rather then north. Note that the constructor expects the orientation to be in degrees, not radians.

`length`**:** Object of class `"numeric"`: The length of the line segment in the correct units.

`segment`**:** Object of class `"SpatialLines"`: The graphical object corresponding to the line segment itself.

`location`**:** Object of class `"SpatialPoints"`: This is a `"SpatialPoints"` representation of the location of the object. In the “lineSegment” class, this is the center of the line segment, which will often correspond to the `location` slot in the “Stem” object under sampling surface simulations.

`spID`**:** Object of class `"character"`: A unique identifier that is used in the “SpatialPolygons” representation of the object.

`spUnits`**:** Object of class `"CRS"`: A valid string of class “CRS” denoting the spatial units coordinate system (?CRS for more information) as in package **sp**.

Extends

Class "`ArealSampling`", directly.

Methods

bbox `signature(obj = "lineSegment")`: Return the bounding box.

plot `signature(x = "lineSegment", y = "missing")`: Graphical display of the object.

summary `signature(object = "lineSegment")`: A summary of the object.

Author(s)

Jeffrey H. Gove

See Also

The "`ArealSampling`" and "`horizontalLineIZ`" classes.

Examples

```
showClass("lineSegment")
```

`lineSegment-methods` *Methods for "`lineSegment`" object creation in Package* **sampSurf**

Description

There is currently only one method based on the `lineSegment` generic that is used for object construction. It is detailed below.

Methods

`signature(length = "numeric", orientation = "numeric")` This method takes the segment length and orientation as the signature arguments along with other optional aruments described as follows...

usage...

```
lineSegment(length,
            orientation,
            units = 'metric',
            spUnits = CRS(projargs=as.character(NA)),
            centerPoint = c(x=0, y=0),
            description = 'line segment',
            spID = paste('ls',.StemEnv$randomID(),sep=':'),
            ...)
```

- `length`: The length of the line segment in the appropriate units (feet for “English” or meters for “metric”).

- `orientation`: The line orientation from *north* as an azimuth in degrees; e.g. `orientation=45`.
- `units`: Either “English” or “metric”. These must be conformable with the projection in `spUnits`.
- `spUnits`: A valid `CRS` object specifying the Coordinate Reference System. This defaults to `NA`, which means you want to use your own user-defined system, say for a sample line located in the field.
- `centerPoint`: The location of the center of the line segment in the appropriate spatial units. This should be a numeric vector of length 2 with names "x" and "y".
- `description`: A character vector description of the tract.
- `spID`: Each object should have its own *unique* identifier that is used in constructing the `Lines` object for the segment. This becomes very important when combining individual line segments into a population. If nothing is supplied, a random ID is generated.

`mcsContainer` *Generate Objects of Class "`mcsContainer`"*

Description

The methods for this function will allow the creation of valid "`mcsContainer`" class objects. Please note that it is preferable to use the constructors named after the Monte Carlo method being used, rather than calling this function directly. For example, to use importance sampling, call the `importanceSampling` constructor with a container of "`Stem`" subclass objects. Please see the vignette below for detailed examples. See also `mcsContainer-methods` for method details.

Usage

```
mcsContainer(object, ...)
```

Arguments

`object`	The signature object for the generic.
`...`	Just gobbled presently.

Details

The vignette below gives examples on creating "mcsContainer" objects for each of the Monte Carlo subsampling methods available in **sampSurf**. The respective methods are simpler to use and more intuitive, and their use will ensure that a valid container object is returned.

Value

A valid object of class "`mcsContainer`."

Note

Please note that this is not a completely functional container class in the traditional sense at present as it does not have replacement, deletion, or addition functions. If you need to do any of these operations, perform them on the `list` object (in the `mcsObjs` slot) and then recreate the container. If the object is not re-built after, e.g., deletion, the summary statistics will be incorrect.

Author(s)

Jeffrey H. Gove

References

Gove, J. H. 2013. Monte Carlo sampling methods in **sampSurf**. Package vignette.

See Also

Please see the direct constructors: `crudeMonteCarlo`, `importanceSampling` and `controlVariate` for methods to create collections of objects under these subsampling schemes. For antithetic versions of these methods, see the `antitheticContainer` class.

Examples

```
sTrees = standingTrees(5, startSeed = 12)
sTrees.cmc = crudeMonteCarlo(sTrees, n.s = 10)
sTrees.cmc
print(sTrees.cmc@stats, digits = 4)
```

`mcsContainer-class` *Class* `"mcsContainer"`

Description

This class allows one to store a collection of objects that are all a subclass of "`MonteCarloSampling`."

Objects from the Class

Objects can be created by calls of the form `new("mcsContainer",...)`. However, using the constructor methods with the same name as the subsampling method used is preferred (see `mcsContainer`). Please refer to the vignette reference below for more examples.

Slots

`mcsObjs`: Object of class `"list"`: Each object in the list must be of the same class for a valid object.

`stats`: Object of class `"matrix"`: Summary statistics for the objects in slot `mcsObjs`.

`description`: Object of class `"character"`: A character description of the contents if desired.

Methods

antitheticSampling `signature(object = "mcsContainer")`: Apply antithetic sampling to each object in the collection.

hist `signature(x = "mcsContainer")`: Generate histograms for the collection.

plot `signature(x = "mcsContainer", y = "missing")`: Plot the container collection.

show `signature(object = "mcsContainer")`: Print a summary.

summary `signature(object = "mcsContainer")`: Print a summary.

Note

Please note that this is not a completely functional container class in the traditional sense at present as it does not have replacement, deletion, or addition functions. If you need to do any of these operations, perform them on the `list` object (in the `mcsObjs` slot) and then recreate the container. If the object is not re-built after, e.g., deletion, the summary statistics will be incorrect.

Author(s)

Jeffrey H. Gove

References

Gove, J. H. 2013. Monte Carlo sampling methods in **sampSurf**. Package vignette.

See Also

`mcsContainer`, `antitheticContainer`.

Examples

```
showClass("mcsContainer")
```

`mcsContainer-methods` *Methods for* "`mcsContainer`" *object construction in Package* **sampSurf**

Description

There is only one method for the `mcsContainer` generic function. Please note that this method should not be called directly. Instead, one should use one of the methods named for the subsampling procedure: `crudeMonteCarlo`, `importanceSampling` and `controlVariate` as described in the generic `mcsContainer`. These methods call this function with a `list` object to actually create the container, ensuring that everything is set correctly for the collection and the subsampling method used.

Methods

signature(object = "list")

usage...

```
mcsContainer(object,
             description = 'Monte Carlo Sampling container object',
             ... )
```

- object: An object of class "list".

---

mirageInclusionZoneGrid-class
Class "mirageInclusionZoneGrid"

Description

This class is a subclass of "InclusionZoneGrid" that is specifically for use in developing sampling surfaces where the mirage method is to be implemented on a "mirageTract". It has a separate constructor method, which should be used to construct the object.

Objects from the Class

Objects can be created by calls of the form new("mirageInclusionZoneGrid",...). However, this is not recommended because the objects are quite complex. Instead, please use the object constructor izGridMirage to create new objects. More details are found in the vignettes listed below.

Slots

As noted, this class is a subclass of "InclusionZoneGrid"; it adds several slots to the class, all other slots not defined below are as described in the superclass...

slopOver**:** Object of class "logical": A vector of of length four which flags where there was any boundary overlap in the cardinal ('north','south','east','west') directions: TRUE if so, FALSE otherwise. Note that one can easily check this vector to determine whether the associated *.polygon and *.grid slots are non-NULL.

north.polygon**:** Object of class "SPNULL": If slopover['north'], then this slot holds the external portion of the inclusion zone polygon due to boundary slopover on the north side. Otherwise, it is NULL.

north.grid**:** Object of class "RLNULL": If slopover['north'], then this slot holds the external portion of the extended grid on the north side. Otherwise, it is NULL.

south.polygon**:** Object of class "SPNULL": If slopover['south'], then this slot holds the external portion of the inclusion zone polygon due to boundary slopover on the south side. Otherwise, it is NULL.

south.grid**:** Object of class "RLNULL": If slopover['south'], then this slot holds the external portion of the extended grid on the south side. Otherwise, it is NULL.

`east.polygon`**:** Object of class `"SPNULL"`: If `slopover['east']`, then this slot holds the external portion of the inclusion zone polygon due to boundary slopover on the east side. Otherwise, it is `NULL`.

`east.grid`**:** Object of class `"RLNULL"`: If `slopover['east']`, then this slot holds the external portion of the extended grid on the east side. Otherwise, it is `NULL`.

`west.polygon`**:** Object of class `"SPNULL"`: If `slopover['west']`, then this slot holds the external portion of the inclusion zone polygon due to boundary slopover on the west side. Otherwise, it is `NULL`.

`west.grid`**:** Object of class `"RLNULL"`: If `slopover['west']`, then this slot holds the external portion of the extended grid on the west side. Otherwise, it is `NULL`.

`izGrid.extended`**:** Object of class `"izgNULL"`: This is the object showing the extended grid that encompases any slopover regions external to the boundary, where applicable. Otherwise, it is `NULL` if there is no slopover on any side.

Extends

Class `"InclusionZoneGrid"`, directly.
Class `"izgNULL"`, by class "InclusionZoneGrid", distance 2.

Methods

plot `signature(x = "mirageInclusionZoneGrid",y = "missing")`: This will plot the object.

Author(s)

Jeffrey H. Gove

References

See the “Mirage Method” and “InclusionZoneGrid” vignettes for more explanation.

See Also

`izGridMirage`, `InclusionZoneGrid`, `sampSurf`

Examples

```
showClass("mirageInclusionZoneGrid")
```

`mirageTract` *Generate Objects of Class "`mirageTract`"*

Description

This generic will allow the construction of tract objects within **sampSurf** that will use the mirage method for boundary overlap correction on any inclusion zones where slopover is encountered. This constructor method should be used in preference to `new` to insure a valid object.

Usage

```
mirageTract(tract, ...)
```

Arguments

`tract`	An object of class "`Tract`".
`...`	Items to be gobbled for now.

Details

It is very simple to create an object of this class, all one requires is an existing “Tract” object as shown in the example below and the methods: `mirageTract-methods`.

Value

A valid object of class "`mirageTract`."

Author(s)

Jeffrey H. Gove

References

See the “Mirage Method” vignette for more explanation.

See Also

`Tract`

Examples

```
tract = Tract(c(x=20,y=20), cellSize=1)    #one-meter resolution
(mtract = mirageTract(tract))
```

`mirageTract-class` *Class* `"mirageTract"`

Description

This class is a subclass of the `"Tract"` class. It provides no new functionality on its own. Its purpose is to define a class that will convey to other components in the system the implicit request to apply the mirage method to any stems and inclusion zones on the tract that overlap the boundary. For example, using a "mirageTract" in one of the `sampSurf` constructors will tell the constructor that the mirage method should be used on the simulation.

Objects from the Class

Objects can be created by calls of the form `new("mirageTract",...)`. However, this is not recommended due to the complexity of the class. Instead, use the `mirageTract` constructor to create objects.

Slots

There are no new slots add to this class, please see the slot definition for the superclass `"Tract"`.

Extends

Class `"Tract"`, directly.
Class `"RasterLayer"`, by class "Tract", distance 2.
Class `"Raster"`, by class "Tract", distance 3.
Class `"RLNULL"`, by class "RasterLayer", distance 3.
Class `"BasicRaster"`, by class "Tract", distance 4.

Methods

izGridMirage `signature(izObject = "InclusionZone",tract = "mirageTract")`: Creates `"mirageInclusionZoneGr`
objects.

Author(s)

Jeffrey H. Gove

References

Gregoire and Valentine (2008) for the mirage method.

See Also

`Tract`, `bufferedTract`

Examples

```
showClass("mirageTract")
```

`mirageTract-methods` *Methods for* `mirageTract` *object construction in Package* **sampSurf**

Description

There is only one existing method available for the `mirageTract` generic function, which will generate valid objects of class "`mirageTract`".

Methods

`signature(tract = "Tract")`

usage...

```
mirageTract(tract,
            ... )
```

- `tract`: A valid tract object.
- `...`: Gobbled.

`monte` *Generate Objects of Class "*`monte`*"*

Description

This is the generic definition for generating objects of class "monte." There are currently several methods corresponding to this generic whose documentation may be found in `monte-methods`.

Usage

```
monte(object, ...)
```

Arguments

`object`	Signature argument, which differs for each method. This specifies the population from which samples will be drawn.
`...`	See methods.

Details

The methods associated with this generic should be used to construct objects of class "`monte`." These objects are specifically designed to hold information about Monte Carlo experiments where one resamples from a known population to infer efficiency and perhaps locate any bias in different sampling estimators. The constructor methods can be used to look at traditional normal theory and bootstrap confidence intervals in terms of nominal catch rates for the population mean.

Value

A valid object of class "monte."

Author(s)

Jeffrey H. Gove

References

The "'monte": When is *n* Sufficiently Large?' vignette.

See Also

monte, monteSample

Examples

```
#
# these examples are commented-out because they consume
# cpu time when checking the package--just copy and paste
# them if you want to try them out...
#
# from a sampSurf object...
#
## Not run:
smTract = Tract(c(x=30,y=30), cellSize=0.5)
smbuffTr = bufferedTract(8,smTract)
ss.sa = sampSurf(10, smbuffTr, 'sausageIZ', plotRadius=3, estimate='Length')
m.sa = monte(ss.sa, n=c(10,20))
hist(m.sa)

## End(Not run)

#
# simple population...
#
## Not run:
mp = montePop(rnorm(100), n=c(10,30))
mt = monte(mp, mcSamples=250, R=150)     #takes n from mp object
mt

## End(Not run)
```

`monte-class` *Class* `"monte"`

Description

This is the class that contains the information about a basic repeated sampling (Monte Carlo) run. The class makes use of several other classes in the package that should be consulted for details. Also, the vignette referenced below is a good source of extended information and examples of how this class would be used.

Objects from the Class

Objects can be created by calls of the form `new("monte",...)`. However, use of the constructor generic "`monte`" with methods for creating objects is recommended. This is to be preferred over the use of `new` since the objects returned from the constructor are guaranteeted to be valid.

Slots

`pop`**:** Object of class `"montePop"`: A Monte Carlo population object, please see `montePop` for details.

`estimate`**:** Object of class `"character"`: In the case of `sampSurf` objects, this is the attribute for which the surface has been estimated.

`NTsamples`**:** Object of class `"monteNTSampleOrNULL"`: An object of class `monteNTSample`, or `NULL` if non-existent. See the constructor for details.

`BSsamples`**:** Object of class `"monteBSSampleOrNULL"`: An object of class `monteBSSample`, or `NULL` if non-existent. See the constructor for details.

`description`**:** Object of class `"character"`: Some descriptive text about the object.

Methods

hist `signature(x = "monte")`: Generate a set of histograms.

show `signature(object = "monte")`: Print the object summary succinctly.

summary `signature(object = "monte")`: Print the object summary.

Author(s)

Jeffrey H. Gove

References

The “‘monte”: When is *n* Sufficiently Large?’ vignette.

See Also

`montePop`, `monteSample`

Examples

```
showClass("monte")
```

---

monte-methods *Methods for "monte" Object Construction in Package* **sampSurf**

Description

The following methods for generic function monte will construct valid objects of class "monte". Please see the "'monte": When is *n* Sufficiently Large?' vignette for more details and examples.

Methods

Please note that each constructor has the same argument list, so it is only stated once below for the first constructor. The signature argument in each case specifies the *population* from which samples should be drawn.

`signature(object = "montePop")` This constructor method is only for use with objects of class "montePop". Ultimately, this constructor is called by all the others.

usage. . .

```
monte(object,
      zeroTruncate = TRUE,
      n = object@n,
      mcSamples = 100,
      type = c('both', 'normalTheory', 'bootstrap'),
      R = 100,
      alpha = 0.05,
      description = 'Monte Carlo Object',
      replace = TRUE,
      startSeed = NA,
      runQuiet = TRUE,
      ... )
```

- `object`: An object of class "montePop".
- `zeroTruncate`: `TRUE`: truncate the zero values from the population; `FALSE`: leave it as is.
- `n`: A vector of sample sizes that will be used to draw `mcSamples` Monte Carlo samples from the population.
- `mcSamples`: A scalar specifying the number of Monte Carlo samples to draw from the population of sizes `n`.
- `type`: The type of confidence intervals to calculate: normal theory, bootstrap, or both.
- `R`: A scalar specifying the number of bootstrap sample replicates to draw.
- `alpha`: The two-tailed alpha level for confidence interval calculation. I.e., for the 95% confidence level, specify `alpha=0.05`.

- `replace`: `TRUE`: sample with replacement; `FALSE`: sample without replacement. Note that this only affects how each of the individual `mcSamples` samples are drawn from the population; specifically, note that bootstrapping is always with replacement *from* these individual `mcSamples` samples by definition.
- `description`: A description of the object as a character string.
- `startSeed`: A scalar specifying a random number seed for the Monte Carlo draws. See `initRandomSeed` for details.
- `runQuiet`: `TRUE`: no feedback; `FALSE`: feedback if available.
- ... : Other arguments to be passed along to `montePop`, `monteNTSample` and `monteBSSample`.

`signature(object = "numeric")` This constructor method is only for use with objects of class "`numeric`"; i.e., a numeric vector.

usage...

```
monte(object,
      zeroTruncate = TRUE,
      n = c(10),
      mcSamples = 100,
      type = c('both', 'normalTheory', 'bootstrap'),
      R = 100,
      alpha = 0.05,
      description = 'Monte Carlo Object',
      replace = TRUE,
      startSeed = NA,
      runQuiet = TRUE,
      ... )
```

- `object`: An object of class "`numeric`". This should be a numeric vector of appropriate length for the population.

`signature(object = "sampSurf")` This constructor method is only for use with objects of class "`sampSurf`". The population will be extracted from the cell values in the object.

usage...

```
monte(object,
      zeroTruncate = TRUE,
      n = c(10),
      mcSamples = 100,
      type = c('both', 'normalTheory', 'bootstrap'),
      R = 100,
      alpha = 0.05,
      description = 'Monte Carlo Object',
      replace = TRUE,
      startSeed = NA,
      runQuiet = TRUE,
      ... )
```

- `object`: An object of class "`sampSurf`".

`monteBSSample` *Generate Objects of Class "`monteBSSample`"*

Description

This generic will allow the simple creation of objects of class "`monteBSSample`." Method definitions defining all arguments are found in `monteBSSample-methods`.

Usage

```
monteBSSample(population, ...)
```

Arguments

`population`	Signature argument, currently only objects of class "`montePop`" are supported.
`...`	See methods.

Details

See the associated methods for this constructor in `monteBSSample-methods`.

Value

A valid object of class "`monteBSSample`."

Author(s)

Jeffrey H. Gove

References

The "'monte": When is *n* Sufficiently Large?' vignette.

See Also

`monteSample`, `monteBSSample`

Examples

```
#
# can take a little time, cut and paste
# to try it...
#
## Not run:
mp = montePop(rnorm(100))
mbss = monteBSSample(mp, n=c(10,20), R=100)
mbss
hist(mbss)
```

```
## End(Not run)
```

---

monteBSSample-class *Class* "monteBSSample"

Description

This class contains the necessary structure for doing Monte Carlo sample size experiments under bootstrap sampling.

Objects from the Class

Objects can be created by calls of the form `new("monteBSSample",...)`. However, it is preferable to use the constructor method of the same name `monteBSSample` to minimize problems with potentially invalid objects.

Slots

`degenerate`: Object of class `"numeric"`: It may happen that, especially for small n, some of the samples drawn from the population can be degenerate (all the same value). When this happens, all of the bootstrap resamples will also be degenerate, and confidence interval estimation is impossible since it is based on the distribution of the bootstrap sample means. This slot is a numeric vector with the number of degenerate samples for each sample size in the `n` slot of the object.

`R`: Object of class `"numeric"`: The number of bootstrap sample replications.

`means`: Object of class `"data.frame"`: The data frame contains the overall bootstrap sample means for each of the `mcSamples` by `length(n)` samples drawn from the population. The overall bootstrap sample mean is defined here as the mean of the R individual (second-stage) bootstrap sample means for each case. Taking column means gives the overall mean for each of the sample sizes.

`lowerCIs`: Object of class `"data.frame"`: This is the lower "BCa" confidence interval endpoint for the $1 - \alpha/2$ confidence level. It is calculated from the distribution of bootstrap sample means that is created in bootstrap sampling for each Monte Carlo sample and sample size, n.

`upperCIs`: Object of class `"data.frame"`: This is the upper "BCa" confidence interval endpoint for the $1 - \alpha/2$ confidence level. It is calculated from the distribution of bootstrap sample means that is created in bootstrap sampling for each Monte Carlo sample and sample size, n.

Extends

Class `"monteSample"`, directly.
Class `"monteBSSampleOrNULL"`, directly.

Methods

summary `signature(object = "monteBSSample")`: Object summary.

Author(s)

Jeffrey H. Gove

References

The "'monte": When is *n* Sufficiently Large?' vignette.

See Also

`monte`, `montePop`

Examples

```
showClass("monteBSSample")
```

`monteBSSample-methods` *Methods for Function* `monteBSSample` *in Package* **sampSurf**

Description

The following methods will construct valid objects of class "`monteBSSample`." Please see the "'monte": When is *n* Sufficiently Large?' vignette for more details and examples.

Methods

`signature(population = "montePop")` This method works with an object of class "`montePop`" as the signature argument.

usage...

```
monteBSSample(population,
              n = c(10),
              mcSamples = 100,
              R = 100,
              alpha = 0.05,
              replace = TRUE,
              startSeed = NA,
              runQuiet = TRUE,
              ... )
```

- `population`: An object of class "`montePop`" containing the population values.
- `n`: A vector of sample sizes that will be used to draw `mcSamples` Monte Carlo samples from the population.
- `mcSamples`: A scalar specifying the number of Monte Carlo samples to draw from the population of sizes `n`.
- `R`: A scalar specifying the number of bootstrap sample replicates to draw.
- `alpha`: The two-tailed alpha level for confidence interval calculation. I.e., for the 95% confidence level, specify `alpha=0.05`.

- `replace`: `TRUE`: sample with replacement; `FALSE`: sample without replacement. This is for the drawing of the `mcSamples` samples of sizes `n`, note for the bootstrap replicates, which are always drawn from the samples with replacement.
- `startSeed`: A scalar specifying a random number seed for the Monte Carlo draws. See `initRandomSeed` for details.
- `runQuiet`: `TRUE`: no feedback; `FALSE`: feedback if available.
- ... : Other arguments to be passed along to `sample`.

`MonteCarloSampling-class`

Class `"MonteCarloSampling"`

Description

This is a virtual base class for apply Monte Carlo subsampling methods within a “downLog” or “standingTree” object. See below for subclasses based on the supported sampling methods.

Objects from the Class

A virtual Class: No objects may be created from it.

Slots

`stem`: Object of class `"Stem"`: This can be either a `downLog` or `standingTree` `subclass object`.

`segBnds`: Object of class `"numeric"`: A vector of length two giving the lower and upper height/length bounds for volume estimation within the bole. All of the following slot definitions below are relative to the segment of the bole defined by these bounds. These bounds correspond to the limits of integration along the bole.

`n.s`: Object of class `"numeric"`: The number of Monte Carlo samples (a scalar).

`startSeed`: Object of class `"numeric"`: The scalar seed for the random number generator used in the call to the class constructor. Please see the documentation in `initRandomSeed` for possible values and their meaning. Suffice it to say that storing this in the object allows for object replication. Note that if `startSeed = NA`, then the seed is not replicable, but the sampling run is by using the random numbers in the `u.s` slot.

`u.s`: Object of class `"numeric"`: The uniform random numbers used in selecting the sampling points along the bole.

`description`: Object of class `"character"`: A description of the object if desired (defaults are given for each class).

`userArgs`: Object of class `"list"`: Some proxy functions have extra arguments that are required when called from the constructor methods. This slot stores these arguments and their values from the call. This is necessary, e.g., for re-applying a given Monte Carlo method to the `(1-u.s)` points in antithetic sampling.

Methods

antitheticSampling `signature(object = "MonteCarloSampling")`: Allows for antithetic sampling given a subclass object.

show `signature(object = "MonteCarloSampling")`: For printing the subclass object.

summary `signature(object = "MonteCarloSampling")`: A printed summary of the subclass object.

Author(s)

Jeffrey H. Gove

References

Gove, J. H. 2013. Monte Carlo sampling methods in **sampSurf**. Package vignette.

See Also

The following subclasses and related: `crudeMonteCarlo`, `importanceSampling`, `controlVariate`, `antitheticSampling`.

Examples

```
showClass("MonteCarloSampling")
```

`MonteCarloSamplingIZ-class`

Class `"MonteCarloSamplingIZ"`

Description

This virtual class exists to be combined with an areal sampling class of choice. This class facilitates creation of a new areal sampling method that employs one of the Monte Carlo subsampling methods supported through subclasses of `"MonteCarloSampling"`. Generally it will only be of interest to someone desiring to write extensions to **sampSurf** in the form of Monte Carlo subsampling on down logs or standing trees within areal methods. Please see the class definition for `horizontalPointCMCIZ` for an example of how this can be combined with an areal method.

Objects from the Class

A virtual Class: No objects may be created from it.

Slots

`mcsObj`: Object of class `"MonteCarloSampling"` A subclass objects of `MonteCarloSampling`; please see that class for extant subclasses/sampling methods.

`antithetic`: Object of class `"logical"` `TRUE`: if antithetic sampling variant has been used for the object in the `mcsObj` slot; `FALSE`: if not.

`proxy`: Object of class `"character"` The named of the proxy function used in Monte Carlo sampling.

Methods

No methods defined with class "MonteCarloSamplingIZ" in the signature.

Author(s)

Jeffrey H. Gove

See Also

`MonteCarloSampling`, `horizontalPointCMCIZ`

Examples

```
showClass("MonteCarloSamplingIZ")
```

`monteNTSample` *Generate Objects of Class "`monteNTSample`"*

Description

This generic will allow the simple creation of objects of class "`monteNTSample`." Method definitions defining all arguments are found in `monteNTSample-methods`.

Usage

```
monteNTSample(population, ...)
```

Arguments

`population`	Signature argument, currently only objects of class "`montePop`" are supported.
`...`	See methods.

Details

See the associated methods for this constructor in `monteNTSample-methods`.

Value

A valid object of class "`monteNTSample`."

Author(s)

Jeffrey H. Gove

References

The "'monte": When is *n* Sufficiently Large?' vignette.

See Also

`monteSample`, `monteBSSample`

Examples

```
mp = montePop(rnorm(100))
mnts = monteNTSample(mp, n=c(10,20,30))
mnts
hist(mnts)
```

`monteNTSample-class` *Class* `"monteNTSample"`

Description

This class contains the necessary structure for doing normal theory sample size experiments under simple random sampling.

Objects from the Class

Objects can be created by calls of the form `new("monteNTSample",...)`. However, it is preferable to use the constructor method of the same name `monteNTSample` to minimize problems with potentially invalid objects.

Slots

Only one new slot is added here from the superclass. In addition, the definitions for three other slots that are method-dependent are also given.

`t.values`: Object of class `"numeric"`: Student's *t* values for each sample size `n` with two-tailed $alpha$-level `alpha`.

`means`: Object of class `"data.frame"`: The data frame contains the individual means for all `mcSamples` by `length(n)` samples drawn from the population. Taking column means gives the overall mean for each of the sample sizes.

`lowerCIs`: Object of class `"data.frame"`: This is the usual normal theory lower limit for each sample: $\bar{y} - t_{n-1}^{1-\alpha/2} s_{\bar{y}}$, where t is Student's t-value and $s_{\bar{y}}$ is the standard error of the mean for the sample.

`upperCIs`: Object of class `"data.frame"`: This is the usual normal theory upper limit for each sample: $\bar{y} + t_{n-1}^{1-\alpha/2} s_{\bar{y}}$, where t is Student's t-value and $s_{\bar{y}}$ is the standard error of the mean for the sample. object.

Extends

Class "`monteSample`", directly.
Class "`monteNTSampleOrNULL`", directly.

Methods

No methods defined with class "monteNTSample" in the signature.

Author(s)

Jeffrey H. Gove

References

The "'monte": When is *n* Sufficiently Large?' vignette.

See Also

`monte`, `montePop`

Examples

```
showClass("monteNTSample")
```

`monteNTSample-methods` *Methods for Function* `monteNTSample` *in Package* **sampSurf**

Description

The following methods will construct valid objects of class "`monteNTSample`." Please see the "'monte": When is *n* Sufficiently Large?' vignette for more details and examples.

Methods

`signature(population = "montePop")` This method works with an object of class "`montePop`" as the signature argument.

usage...

```
monteNTSample(population,
              n = c(10),
              mcSamples = 100,
              alpha = 0.05,
              replace = TRUE,
              startSeed = NA,
              runQuiet = TRUE,
              ... )
```

- population: An object of class "montePop" containing the population values.
- n: A vector of sample sizes that will be used to draw mcSamples Monte Carlo samples from the population.
- mcSamples: A scalar specifying the number of Monte Carlo samples to draw from the population of sizes n.
- alpha: The two-tailed alpha level for confidence interval calculation. I.e., for the 95% confidence level, specify alpha=0.05.
- replace: TRUE: sample with replacement; FALSE: sample without replacement.
- startSeed: A scalar specifying a random number seed for the Monte Carlo draws. See initRandomSeed for details.
- runQuiet: TRUE: no feedback; FALSE: feedback if available.
- ... : Other arguments to be passed along to sample.

---

montePop *Generate Objects of Class "montePop"*

Description

This is the generic definition for generating objects of class "montePop." There are several methods corresponding to this generic with different signatures that may be found in montePop-methods.

Usage

```
montePop(popVals, ...)
```

Arguments

popVals	Signature object, which differs for each method.
...	See methods.

Details

The methods associated with this generic should be used to construct objects of class "montePop." Such objects contain the population values and the basic population parameters (statistics), which will be calculated automatically by the constructor.

Value

A valid object of class "montePop."

Author(s)

Jeffrey H. Gove

References

The “‘monte”: When is *n* Sufficiently Large?’ vignette.

See Also

monte, monteSample

Examples

```
mp = montePop(rnorm(100))
mp
hist(mp)
```

---

montePop-class *Class* "montePop"

Description

Holds a population object with its calculated parameters for use in Monte Carlo repeated sampling experiments.

Objects from the Class

Objects can be created by calls of the form `new("montePop",...)`. However, it is probably better, and certainly easier, to use the constructor method `montePop` to create objects.

Slots

`mean`**:** Object of class `"numeric"`: The population mean.

`var`**:** Object of class `"numeric"`: The population variance.

`stDev`**:** Object of class `"numeric"`: The population standard deviation

`N`**:** Object of class `"numeric"`: The size of the population.

`total`**:** Object of class `"numeric"`: The population total.

`popVals`**:** Object of class `"numeric"`: The individual population values.

`zeroTruncated`**:** Object of class `"logical"`: `TRUE`: The population is zero-truncated; `FALSE` otherwise. This is used specifically with "`sampSurf`" objects.

`n`**:** Object of class `"numeric"`: The desired sample sizes for calculation of the following three slots. Note that this and the following slots can be `NA` if this sample size-based information is not desired.

`fpc`**:** Object of class `"numeric"`: The finite population correction for each sample size, or `NA`.

`varMean`**:** Object of class `"numeric"`: The variance of the mean for the population at each sample size, or `NA`.

`stErr`**:** Object of class `"numeric"`: The population standard error of the mean for each sample size, or `NA`.

`description`**:** Object of class `"character"`: a description of the population if desired.

Methods

hist `signature(x = "montePop")`: population histogram

monteBSSample `signature(population = "montePop")`: Bootstrap sampling from the population.

monteNTSample `signature(population = "montePop")`: Normal threory (simple random) sampling from the population.

show `signature(object = "montePop")`: the population.

summary `signature(object = "montePop")`: of the population.

Author(s)

Jeffrey H. Gove

References

The "'monte": When is *n* Sufficiently Large?' vignette.

See Also

`monte`, `monteSample`

Examples

```
showClass("montePop")
```

`montePop-methods` *Methods for "*`montePop`*" Object Construction in Package* **sampSurf**

Description

The following methods will construct valid objects of class "`montePop`". Please see the "'monte": When is *n* Sufficiently Large?' vignette for more details and examples.

Methods

`signature(popVals = "numeric")` This method is meant to be used with any population that can be represented as a numeric vector.

usage...

```
montePop(popVals,
         zeroTruncate = FALSE,
         n = NA,
         description = 'Monte Carlo Population Object',
         ... )
```

- `popVals`: A numeric vector containing the population values.

- `zeroTruncate`: `TRUE`: truncate the zero values from the population; `FALSE`: leave it as is.
- `n`: A vector of sample sizes that will be used in conjunction with drawing samples from the population; or, `NA` if not desired.
- `description`: A description of the object as a character string.
- ... : Other arguments to be passed along–not used at present.

`signature(popVals = "sampSurf")` This method is meant to be used with objects of class "`sampSurf`". All arguments other than the signature argument are as defined for the previous method.

usage...

```
montePop(popVals,
         zeroTruncate = FALSE,
         n = NA,
         description = 'Monte Carlo Population Object: sampSurf',
         ... )
```

- `popVals`: An object of class "`sampSurf`" from which the population (sampling surface grid) values will be extracted.

`monteSample-class` *Class* `"monteSample"`

Description

This is a virtual class defining the base representation for objects that hold information about repeated (Monte Carlo) sampling from population objects of class "`montePop`". It provides a basic class setup for looking at Monte Carlo convergence of as the sample size grows larger.

Objects from the Class

A virtual Class: No objects may be created from it. For usable subclasses please see `monteNTSample` and `monteBSSample`.

Slots

`mcSamples`: Object of class `"numeric"`: A scalar numeric specifying the number of Monte Carlo samples drawn from the population.

`n`: Object of class `"numeric"`: A numeric vector listing the different sample sizes recorded in the object that have been drawn from a "montePop" population object. So, if we have drawn samples of size n = 10,20,30, then this would hold `c(10,20,30)`, with associated names `c('n.10','n.20','n.30')`.

`alpha`: Object of class `"numeric"`: The two-tailed alpha level for which confidence intervals have been calculated. I.e., for the 95% confidence level `alpha = 0.05`

`replace`: Object of class `"logical"`: `TRUE` if the samples have been drawn from the population with replacement, `FALSE` otherwise.

`ranSeed`**:** Object of class `"numeric"`: The random number seed as a numeric vector. Please see the R documentation on `.Random.seed` for information on the format of this slot. Note that it is *not* a simple scalar integer "seed", but a vector of integers containing the state of the random number generator at the beginning of the simulations.

`fpc`**:** Object of class `"numeric"`: The finite population correction factors for each sample size `n`. The correction is: f = (N-n)/N.

`means`**:** Object of class `"data.frame"`: A data frame with `mcSamples` rows, and one column for each of the sample sizes in the `n` slot of the object. What is stored here depends on the subclass object type, so please see the respective definitions for these slots.

Note: The following six slots have the same dimensions as the `means` *slot.*

`vars`**:** Object of class `"data.frame"`: Contains the individual sample variances.

`stDevs`**:** Object of class `"data.frame"`: Contains the individual sample standard deviations.

`varMeans`**:** Object of class `"data.frame"`: Contains the individual variance of the mean values.

`stErrs`**:** Object of class `"data.frame"`: Contains the individual standard errors.

`lowerCIs`**:** Object of class `"data.frame"`: Contains the individual lower limit for the confidence intervals. This is defined differently for the different subclasses.

`upperCIs`**:** Object of class `"data.frame"`: Contains the individual upper limit for the confidence intervals. This is defined differently for the different subclasses.

`caught`**:** Object of class `"data.frame"`: Contains a flag where `TRUE` means the confidence interval caught the population mean and `FALSE` means it failed to catch the population mean. Taking column sums, therefore (since `TRUE == 1` and `FALSE == 0`) will give the number of intervals that caught the population mean for each sample size. This is used to calculate the next slot below.

`caughtPct`**:** Object of class `"numeric"`: The percentage of times the confidence intervals caught the population mean as calculated from the data frame in the `caught` slot of the object.

`stats`**:** Object of class `"data.frame"`: A summary data frame with rows as the *average* of each column (i.e., over all Monte Carlo samples) from the information in the data frames above (`means`, `vars`, `stDevs`, `varMeans`, `stErrs`, `lowerCIs`, and `upperCIs`). The interpretation of some of the rows depends on the subclass object as has been mentioned above, please see the vignette below for more details.

Methods

hist `signature(x = "monteSample")`: Histogram of the means by sample size

show `signature(object = "monteSample")`: Object summary.

summary `signature(object = "monteSample")`: Object summary.

Author(s)

Jeffrey H. Gove

References

The "'monte": When is *n* Sufficiently Large?' vignette.

See Also

monte, montePop; for subclasses, see: monteNTSample and monteBSSample.

Examples

```
showClass("monteSample")
```

---

omnibusDLPDSIZ *Generate Objects of Class "omnibusDLPDSIZ"*

Description

This is the generic definition for generating objects of class "omnibusDLPDSIZ." There is only one constructor method corresponding to this generic: omnibusDLPDSIZ-methods.

Usage

```
omnibusDLPDSIZ(downLog, pds, dls, ...)
```

Arguments

downLog	Signature object of class "downLog".
pds	Signature object of class "perpendicularDistance" containing the pertinent perpendicular distance sampling information.
dls	Signature object of class "dlsNumeric" which will accept either an object of class "distanceLimited" containing the pertinent distance limited sampling information or a "numeric" object specifying the distance limit.
...	See methods.

Details

Since only one method exists for this generic, its signature arguments coincide with the above definitions. Please see omnibusDLPDSIZ-methods for more details.

Value

A valid object of class "omnibusDLPDSIZ."

Author(s)

Jeffrey H. Gove

References

"The InclusionZone Class" vignette.

See Also

Class "omnibusDLPDSIZ", and omnibusDLPDSIZ-methods.

Examples

```
dl = downLog(buttDiam=15, solidType=4, logAngle=pi/3, logLen=10, units='English')
pdsEng = perpendicularDistance(kpds=6, units='English')
dlsEng = distanceLimited(2, units='English')
iz.odlpds = omnibusDLPDSIZ(dl, pds=pdsEng, dls=dlsEng)
iz.odlpds
```

---

omnibusDLPDSIZ-class *Class* "omnibusDLPDSIZ"

Description

This class holds the inclusion zone definition for the 'omnibus distance limited perpendicular distance sampling' method for sampling down coarse woody debris. This class is fairly complicated because there are three possibilities for the components of the inclusion zone. It is best to read *"The InclusionZone Class"* vignette, along with the references for more information—the why's and wherefore's are not presented here, only the class documentation.

Objects from the Class

Objects can be created by calls of the form `new("omnibusDLPDSIZ",...)`. However, this is not recommended because the objects are fairly complex. Instead, one can use the object constructor omnibusDLPDSIZ to create new objects.

Slots

This class is a direct descendent (subclass) of `"distanceLimitedPDSIZ"` and adds no new slots to that definition. Please see the "distanceLimitedPDSIZ" class definition for details.

Extends

Class "distanceLimitedPDSIZ", directly.
Class "perpendicularDistanceIZ", by class "distanceLimitedPDSIZ", distance 2.
Class "downLogIZ", by class "distanceLimitedPDSIZ", distance 3.
Class "pdsIZNull", by class "distanceLimitedPDSIZ", distance 3.
Class "InclusionZone", by class "distanceLimitedPDSIZ", distance 4.

Methods

No new methods are defined with class "omnibusDLPDSIZ" in the signature; instead see those defined for "distanceLimitedPDSIZ".

Author(s)

Jeffrey H. Gove

References

This method has not been published yet, please see the vignette mentioned above for details.

See Also

`downLogIZs` container class

Examples

```
showClass("omnibusDLPDSIZ")
```

`omnibusDLPDSIZ-methods` *Methods for* `"omnibusDLPDSIZ"` *Object Construction in Package* **sampSurf**

Description

This is the one method for generic function `omnibusDLPDSIZ` in Package 'sampSurf' that allows for creation of objects of class "`omnibusDLPDSIZ`."

Methods

`signature(downLog = "downLog", pds = "perpendicularDistance", dls = "dlsNumeric")`

usage...

```
omnibusDLPDSIZ(downLog,
                     pds,
                     dls,
               description = 'inclusion zone for down log omnibus distance limited PDS',
                     spID = paste('odlpds',.StemEnv$randomID(),sep=':'),
                     spUnits = CRS(projargs=as.character(NA)),
                     pdsType = .StemEnv$pdsTypes,
                     ... )
```

- `downLog`: An object of class "`downLog`" for which the inclusion zone is to be determined.
- `pds`: An object of class "`perpendicularDistance`" that supplies the information on the perpendicular distance method for constructing the inclusion zone.
- `dls`: An object of class "`distanceLimited`" that supplies the information on the distance limited method for constructing the inclusion zone. Alternatively a "numeric" distance limit that will be converted to a "distanceLimited" object.
- `description`: A character vector description of the object.
- `spID`: A unique identifier that will be used in displaying the spatial polygon for the inclusion zone component of the object.
- `spUnits`: A valid `CRS` object specifying the Coordinate Reference System. This defaults to `NA`, which means you want to use your own user-defined system, say for a sample plot located in the field.

- `pdsType`: A character string that specifies the type of perpendicular distance sampling used with regard to the selection (i.e., probability proportional to...) of the log. It can be one of "`volume`", "`surfaceArea`" or "`coverageArea`". (See also `.StemEnv$pdsTypes` for legal types.)
- `dots`: Arguments to be passed on.

`omnibusPDSIZ` *Generate Objects of Class "*`omnibusPDSIZ`*"*

Description

This is the generic definition for generating objects of class "omnibusPDSIZ." There is only one constructor method corresponding to this generic: `omnibusPDSIZ-methods`.

Usage

```
omnibusPDSIZ(downLog, pds, ...)
```

Arguments

`downLog`	Signature object of class "`downLog`".
`pds`	Signature object of class "`perpendicularDistance`" containing the pertinent perpendicular distance sampling information.
`...`	See methods.

Details

Since only one method exists for this generic, its signature arguments coincide with the above definitions. Please see `omnibusPDSIZ-methods` for more details.

Value

A valid object of class "`omnibusPDSIZ`."

Author(s)

Jeffrey H. Gove

References

Ducey, M. J., Williams, M. S., Gove, J. H. and Valentine, H. T. 2008. Simultaneous unbiased estimates of multiple downed wood attributes in perpendicular distance sampling. *Canadian Journal of Forest Research* **38**:2044–2051.

Williams, M. S. and Gove, J. H. 2003. Perpendicular distance sampling: an alternative method for sampling downed coarse woody debris. *Canadian Journal of Forest Research* **33**:1564–1579.

See Also

Class "omnibusPDSIZ", and omnibusPDSIZ-methods.

Examples

```
#
# creates an inclusion zone object for sampling with probability
# proportional to volume by default...
#
dl = downLog(buttDiam=15, solidType=4, logAngle=pi/3, logLen=10, units='English')
pdsEng = perpendicularDistance(kpds=6, units='English')
iz.opdsv = omnibusPDSIZ(dl, pdsEng)
iz.opdsv
```

---

omnibusPDSIZ-class *Class* "omnibusPDSIZ"

Description

This class holds the inclusion zone definition for the 'omnibus perpendicular distance sampling' method (Ducey et. al, 2008) for sampling down coarse woody debris.

Objects from the Class

Objects can be created by calls of the form `new("omnibusPDSIZ",...)`. However, this is not recommended because the objects are fairly complex. Instead, one can use the object constructor `omnibusPDSIZ` to create new objects.

Slots

This class is a direct descendent (subclass) of "`perpendicularDistanceIZ`" and adds no new slots to that definition. Please see the "`perpendicularDistanceIZ`" class definition for details.

Extends

Class "`perpendicularDistanceIZ`", directly.
Class "`pdsIZNull`", directly.
Class "`downLogIZ`", by class "perpendicularDistanceIZ", distance 2.
Class "`InclusionZone`", by class "perpendicularDistanceIZ", distance 3.

Methods

izGrid `signature(izObject = "omnibusPDSIZ",tract = "Tract")`: "InclusionZoneGrid" generic constructor

Author(s)

Jeffrey H. Gove

References

Ducey, M. J., Williams, M. S., Gove, J. H. and Valentine, H. T. 2008. Simultaneous unbiased estimates of multiple downed wood attributes in perpendicular distance sampling. *Canadian Journal of Forest Research* **38**:2044–2051.

Williams, M. S. and Gove, J. H. 2003. Perpendicular distance sampling: an alternative method for sampling downed coarse woody debris. *Canadian Journal of Forest Research* **33**:1564–1579.

See Also

`downLogIZs` container class

Examples

```
showClass("omnibusPDSIZ")
```

`omnibusPDSIZ-methods` *Methods for* `"omnibusPDSIZ"` *Object Construction in Package* **sampSurf**

Description

This is the one method for generic function `omnibusPDSIZ` in Package 'sampSurf' that allows for creation of objects of class "`omnibusPDSIZ`."

Methods

`signature(downLog = "downLog", pds = "omnibusPDS")`

usage...

```
omnibusPDSIZ(downLog,
             pds,
         description = 'inclusion zone for down log perpendicular distance sampling',
             spID = paste('pds',.StemEnv$randomID(),sep=':'),
             spUnits = CRS(projargs=as.character(NA)),
             pdsType = .StemEnv$pdsTypes,
             ... )
```

- `downLog`: An object of class "`downLog`" for which the inclusion zone is to be determined.
- `pds`: An object of class "`perpendicularDistance`" that supplies the information on the perpendicular distance method for constructing the inclusion zone.
- `description`: A character vector description of the object.
- `spID`: A unique identifier that will be used in displaying the spatial polygon for the circular plot component of the object.
- `spUnits`: A valid `CRS` object specifying the Coordinate Reference System. This defaults to `NA`, which means you want to use your own user-defined system, say for a sample plot located in the field.

- pdsType: A character string that specifies the type of perpendicular distance sampling used with regard to the selection (i.e., probability proportional to. . .) of the log. It can be one of "volume", "surfaceArea" or "coverageArea". (See also .StemEnv$pdsTypes for legal types.)
- dots: Arguments to be passed on.

---

pairedAICHSIZ *Generate Objects of Class "pairedAICHSIZ"*

Description

This is the generic function for class “pairedAICHSIZ”. Please see the associated method in pairedAICHSIZ-methods for more details.

Usage

```
pairedAICHSIZ(standingTree, angleGauge, ...)
```

Arguments

standingTree	Signature object of class "standingTree".
angleGauge	Signature object of class "angleGauge".
...	See methods.

Details

Since only one method exists for this generic, its signature arguments coincide with the above. Please see pairedAICHSIZ-methods for more details.

Value

A valid object of class "pairedAICHSIZ."

Author(s)

Jeffrey H. Gove

See Also

Class "pairedAICHSIZ", and pairedAICHSIZ-methods.

Examples

```
st = standingTree(dbh=50, solidType=4, height=25)
ag = angleGauge(baf=4)
iz.paichs = pairedAICHSIZ(st, ag)
summary(iz.paichs)
plot(iz.paichs, axes=TRUE, cex=2)
```

`pairedAICHSIZ-class` *Class* `"pairedAICHSIZ"`

Description

This class holds the inclusion zone definition for the paired antithetic importance sampling variant/protocol of critical height sampling for standing trees.

Objects from the Class

Objects can be created by calls of the form `new("pairedAICHSIZ",...)`. However, this is not recommended because the objects are fairly complex. Instead, one can use the object constructor "`pairedAICHSIZ`" to create new objects.

Slots

This class is a subclass of "importanceCHSIZ" (see below). It adds no new slots to the definition of that class, so please refer to it for the slot definitions.

Extends

Class "`importanceCHSIZ`", directly.
Class "`criticalHeightIZ`", by class "importanceCHSIZ", distance 2.
Class "`horizontalPointIZ`", by class "importanceCHSIZ", distance 3.
Class "`circularPlotIZ`", by class "importanceCHSIZ", distance 4.
Class "`standingTreeIZ`", by class "importanceCHSIZ", distance 5.
Class "`InclusionZone`", by class "importanceCHSIZ", distance 6.

Methods

izGrid `signature(izObject = "pairedAICHSIZ",tract = "Tract")`: "InclusionZoneGrid" constructor

Author(s)

Jeffrey H. Gove

References

T. B. Lynch and J. H. Gove. 2013. An antithetic variate to facilitate upper-stem height measurements for critical height sampling and fixed-radius plot sampling with importance sampling. *Canadian Journal of Forest Research* (forthcoming).

"*The InclusionZone Class*" vignette.

See Also

See also the "`circularPlotIZ`", "`horizontalPointIZ`", "`criticalHeightIZ`", "`importanceCHSIZ`", and "`antitheticICHSIZ`" classes, and the "`standingTreeIZs`" container class

Examples

```
showClass("pairedAICHSIZ")
```

`pairedAICHSIZ-methods` *Methods for* "`pairedAICHSIZ`" *object construction in Package* **sampSurf**

Description

This documents the one method for generic function `pairedAICHSIZ` in Package **sampSurf** that allows for creation of objects of class "`pairedAICHSIZ`."

Methods

`signature(standingTree = "standingTree", angleGauge = "angleGauge")`

usage...

```
pairedAICHSIZ(standingTree,
              angleGauge,
              referenceHeight = .StemEnv$referenceCHSIZ,
         description = 'inclusion zone for paired antithetic ICH sampling method',
              spID = paste('paichs',.StemEnv$randomID(),sep=':'),
              spUnits = CRS(projargs=as.character(NA)),
              ...)
```

- `standingTree`: An object of class "`standingTree`" for which the inclusion zone is to be determined.
- `angleGauge`: An object of class "`angleGauge`".
- `referenceHeight`: The height on the stem at which the inclusion zone is to be determined. Currently the choices are “butt” (default) or “dbh”. These are found in the argument assignment `.StemEnv$referenceCHSIZ` above.
- `description`: A character vector description of the object.
- `spID`: A unique identifier that will be used in displaying the spatial polygon for the inclusion zone component of the object.
- `spUnits`: A valid `CRS` object specifying the Coordinate Reference System. This defaults to `NA`, which means you want to use your own user-defined system, say for a sample plot located in the field.
- `dots`: Arguments to be passed on.

`perimeter` *Function to Return the Graphical Perimeter of an Object in Package 'sampSurf'*

Description

Most classes in the **sampSurf** package have some kind of spatial representation that conforms to a class in **sp** (for polygons) or **raster** (for grids). This generic has been defined to return graphical polygon object that most nearly matches the perimeter. For some objects this means returning the bounding box for, perhaps, a collection of logs, or for a "Tract" object.

Usage

```
perimeter(object, ...)
```

Arguments

`object`	Signature object, which differs for each method.
`...`	See methods.

Details

The methods defined for this generic are described in `perimeter-methods`. The function is quite simple, and works essentially the same for each type of object. Again, some leeway in exactly what is returned is taken because we can have individual objects, collections, or grid rather than polygonal objects. In the latter two cases, the perimeter normally would be the minimal bounding box. For other objects in classes that have a well-defined perimeter, such as a downLog, or a circular plot, these are returned. One can always plot their bounding box separately with the help of `link{bboxToPoly}`.

Value

A "`SpatialPolygons`" object that can be plotted directly.

Author(s)

Jeffrey H. Gove

See Also

`bbox`

Examples

```
showMethods("perimeter")
dlogs = downLogs(15, xlim=c(0,20), ylim=c(0,20), buttDiams=c(25,35))
dlogs.perim = perimeter(dlogs)
plot(dlogs.perim, axes=TRUE)
plot(dlogs, add=TRUE)
bbox(dlogs.perim)
```

`perimeter-methods` *Methods for Function perimeter in Package* **sampSurf**

Description

The methods below will extract the spatial perimeter in the form of a "`SpatialPolygons`" object, which can then be plotted. In some cases, where the actual perimeter of the object makes sense, this is returned; in others, where it does not, the perimeter of the bounding box is returned.

Methods

In what follows the constructor methods are loosely organized by those applicable to (e.g., sampling methods associated with) *(i)* down logs, *(ii)* standing trees, and *(iii)* other.

Down log constructors. . . :

`signature(object = "downLog")` Returns the spatial object associated with the perimeter of the down log.

`signature(object = "sausageIZ")` Returns the spatial object associated with the perimeter of the sausage-shaped inclusion zone.

`signature(object = "standUpIZ")` Returns the spatial object associated with the perimeter of the circular-shaped inclusion zone.

`signature(object = "chainSawIZ")` This will return different inclusion zone objects based on the second argument below. . .

usage. . .

```
perimeter(object,
          whatSense = c('point', 'plot', 'sausage'),
          ... )
```

- `object`: An object of one of class "chainSawIZ."
- `whatSense`: 'point' returns the circular plot center point; 'plot' returns the perimeter of the circular plot; 'sausage' returns the whole-log inclusion zone.
- ... : Other arguments to be passed along–not used at present.

`signature(object = "pointRelascopeIZ")` Returns the spatial object associated with the perimeter of the dual circle-shaped inclusion zone.

`signature(object = "perpendicularDistanceIZ")` Returns the spatial object associated with the perimeter of the inclusion zone—works on all subclasses; i.e., "`omnibusPDSIZ`", "`distanceLimitedPDSIZ`", "`omnibusDLPDSIZ`", and "`hybridDLPDSIZ`".

`signature(object = "distanceLimitedIZ")` Returns the spatial object associated with the perimeter of the inclusion zone. This alsow works for objects of class "`distanceLimitedMCIZ`".

Standing tree constructors. . . :

`signature(object = "standingTree")` Returns the spatial object associated with the perimeter of the standing tree.

`signature(object = "circularPlotIZ")` Returns the spatial object associated with the perimeter of the inclusion zone for objects of class "`circularPlotIZ`". It also will return the perimeter of any object that is a subclass; e.g., "`horizontalPointIZ`", "`criticalHeightIZ`", "`importanceCHSIZ`", "`antitheticICHSIZ`", "`pairedAICHSIZ`", "`horizontalPointCMCIZ`", "`horizontalPointISIZ`", and "`horizontalPointCVIZ`".

`signature(object = "horizontalLineIZ")` Returns the spatial object associated with the perimeter of the inclusion zone for objects of class "`horizontalLineIZ`".

Other constructors. . . :

`signature(object = "StemContainer")` Returns the spatial object associated with the bounding box for the collection of “downLog” or “standingTree” objects within the appropriate “downLogs” or “standingTrees” container objects, respectively.

`signature(object = "circularPlot")` Returns the spatial object associated with the perimeter of the circular plot.

`signature(object = "izContainer")` Returns the spatial object associated with the bounding box for the collection of “izContainer” subclass objects.

`signature(object = "Tract")` Returns the spatial object associated with the perimeter of the “Tract”.

`signature(object = "sampSurf")` Returns the spatial object associated with the perimeter of the `tract` slot in the object.

`perpendicularDistance` *Generate Objects of Class "*`perpendicularDistance`*"*

Description

This generic function has only one method used as a constructor function for objects that are of class "`perpendicularDistance`". This method should be used in preference to `new` to insure a valid object.

Usage

```
perpendicularDistance(kpds, ...)
```

Arguments

`kpds`	This is the signature argument: the *Kpds* factor in the appropriate units.
`...`	Arguments that are defined in `perpendicularDistance-methods`

Details

Only one method currently exists for object generation. Its arguments are documented in `perpendicularDistance-methods`

Value

A valid object of class "`perpendicularDistance`"

Author(s)

Jeffrey H. Gove

See Also

`perpendicularDistance-methods` and the "`ArealSampling`" class.

Examples

```
#
# this will have a volume factor of 5445 cubic feet...
#
(pdsEng = perpendicularDistance(4, units="English"))
```

---

perpendicularDistance-class

Class `"perpendicularDistance"`*: Perpendicular Distance Sampling*

Description

A subclass of "`ArealSampling`" that can be used to create objects that encapsulate all the parameters necessary for any variant of perpendicular distance sampling of down woody debris.

Objects from the Class

Objects can be created by calls of the form `new("perpendicularDistance",...)`, and while this is reasonable with this class because there are no graphical slots, it is still not recommended. The preferred method for creating new objects is via the `perpendicularDistance` constructor.

Slots

In addition to those slots in class "ArealSampling," the following are defined...

`factor`**:** Object of class `"numeric"`: The volume, surface area, or coverage area factor in the appropriate units.

`kpds`**:** Object of class `"numeric"`: The *Kpds* factor in the appropriate units.

Extends

Class "`ArealSampling`", directly.

Methods

summary `signature(object = "perpendicularDistance")`: prints a summary of the object

Author(s)

Jeffrey H. Gove

References

Williams, M. S. and Gove, J. H. 2003. Perpendicular distance sampling: an alternative method for sampling downed coarse woody debris. *Canadian Journal of Forest Research* **33**:1564–1579.

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

See Also

The "`ArealSampling`" class.

Examples

```
showClass("perpendicularDistance")
```

`perpendicularDistance-methods`

Methods for "perpendicularDistance" Object Construction in Package 'sampSurf'

Description

There is currently only one method based on the `perpendicularDistance` generic that is used for object construction. It is detailed below.

Methods

`signature(kpds = "numeric")` This method takes the *Kpds* factor in appropriate units as the signature argument along with other optional aruments described as follows. . .

usage. . .

```
perpendicularDistance(kpds,
                      units = 'metric',
                      description = 'perpendicular distance method',
                      ...)
```

- `kpds`: The *Kpds* factor in appropriate units.
- `units`: Either "English" or "metric".
- `description`: A character vector description of the object.

`perpendicularDistanceIZ` *Generate Objects of Class "`perpendicularDistanceIZ`"*

Description

This is the generic definition for generating objects of class "perpendicularDistanceIZ." There is only one constructor method corresponding to this generic: `perpendicularDistanceIZ-methods`.

Usage

```
perpendicularDistanceIZ(downLog, pds, ...)
```

Arguments

`downLog`	Signature object of class "`downLog`".
`pds`	Signature object of class "`perpendicularDistance`" containing the pertinent perpendicular distance sampling information.
`...`	See methods.

Details

Since only one method exists for this generic, its signature arguments coincide with the above definitions. Please see `perpendicularDistanceIZ-methods` for more details.

Value

A valid object of class "`perpendicularDistanceIZ`."

Author(s)

Jeffrey H. Gove

References

Williams, M. S. and Gove, J. H. 2003. Perpendicular distance sampling: an alternative method for sampling downed coarse woody debris. *Canadian Journal of Forest Research* **33**:1564–1579.

See Also

Class "`perpendicularDistanceIZ`", and `perpendicularDistanceIZ-methods`.

Examples

```
#
# creates an inclusion zone object for sampling with probability
# proportional to volume by default...
#
dl = downLog(buttDiam=15, solidType=4, logAngle=pi/3, logLen=10, units='English')
pdsEng = perpendicularDistance(kpds=6, units='English')
iz.pdsv = perpendicularDistanceIZ(dl, pdsEng)
iz.pdsv
```

`perpendicularDistanceIZ-class`
Class `"perpendicularDistanceIZ"`

Description

This class holds the inclusion zone definition for the ‘perpendicular distance sampling’ method (Williams & Gove, 2003) for sampling down coarse woody debris.

Objects from the Class

Objects can be created by calls of the form `new("perpendicularDistanceIZ",...)`. However, this is not recommended because the objects are fairly complex. Instead, one can use the object constructor `perpendicularDistanceIZ` to create new objects.

Slots

Most of the slots are described in the superclasses (see below), please see their help pages for more information. This class adds a few slots to the "downLogIZ" class specification.

`pds`**:** Object of class `"perpendicularDistance"`: This supplies the information for the *Kpds* factor, etc. for establishing the inclusion zone.

`izPerim`**:** Object of class `"matrix"`: A matrix representation of the inclusion zone in homogeneous coordinates. This can be manipulated and plotted as desired for easy access to the inclusion zone where needed.

`area`**:** Object of class `"numeric"`: The exact area of the inclusion zone.

`perimeter`**:** Object of class `"SpatialPolygons"`: This is the inclusion zone perimeter as a “SpatialPolygons” object.

`pgArea`**:** Object of class `"numeric"`: This is the area of the 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, and adjust if necessary the number of points defining the inclusion area perimeter.

`pdsType`**:** Object of class `"character"`: Specifies the type of perpendicular distance sampling used with regard to the selection (i.e., probability proportional to. . .) of the log. It can be one of `"volume"`, `"surfaceArea"` or `"coverageArea"`. (See also `.StemEnv$pdsTypes` for legal types.)

Extends

Class "`downLogIZ`", directly.
Class "`pdsIZNull`", directly.
Class "`InclusionZone`", by class "downLogIZ", distance 2.

Methods

izGrid `signature(izObject = "perpendicularDistanceIZ", tract = "Tract")`: "InclusionZone-Grid" constructor

perimeter `signature(object = "perpendicularDistanceIZ")`: Return the object perimeter

plot `signature(x = "perpendicularDistanceIZ", y = "missing")`: Plot the object

summary `signature(object = "perpendicularDistanceIZ")`: Object summary

Author(s)

Jeffrey H. Gove

References

Williams, M. S. and Gove, J. H. 2003. Perpendicular distance sampling: an alternative method for sampling downed coarse woody debris. *Canadian Journal of Forest Research* **33**:1564–1579.

See Also

`downLogIZs` container class

Examples

```
showClass("perpendicularDistanceIZ")
```

`perpendicularDistanceIZ-methods`
Methods for "`perpendicularDistanceIZ`" *Object Construction in Package* **sampSurf**

Description

This is the one method for generic function `perpendicularDistanceIZ` in Package 'sampSurf' that allows for creation of objects of class "`perpendicularDistanceIZ`."

Methods

signature(downLog = "downLog", pds = "perpendicularDistance")

usage...

```
perpendicularDistanceIZ(downLog,
                        pds,
                description = 'inclusion zone for down log perpendicular distance sampling',
                        spID = paste('pds',.StemEnv$randomID(),sep=':'),
                        spUnits = CRS(projargs=as.character(NA)),
                        pdsType = .StemEnv$pdsTypes,
                        ... )
```

- downLog: An object of class "downLog" for which the inclusion zone is to be determined.
- pds: An object of class "perpendicularDistance" that supplies the information on the perpendicular distance method for constructing the inclusion zone.
- description: A character vector description of the object.
- spID: A unique identifier that will be used in displaying the spatial polygon for the inclusion zone component of the object.
- spUnits: A valid CRS object specifying the Coordinate Reference System. This defaults to NA, which means you want to use your own user-defined system, say for a sample plot located in the field.
- pdsType: A character string that specifies the type of perpendicular distance sampling used with regard to the selection (i.e., probability proportional to...) of the log. It can be one of "volume", "surfaceArea" or "coverageArea". (See also .StemEnv$pdsTypes for legal types.)
- dots: Arguments to be passed on.

---

plot *Plot objects within package 'sampSurf'*

Description

Each different class of object in this package has a plot method associated with it that is based on R's graphics::plot generic. These all have the same form for the first signature arguments as shown below, but many have different arguments beyond those normmaly found in plot to add some flexibility to the rendering of each object. These are all given in plot-methods.

Details

The different classes within the package are unique enough that there is not much functionality in common with regard to shared arguments. Two notable exceptions are that often the individual plot methods for a given object do not show axes and assume that a new plot is desired. Two common arguments that can be used to override these assumptions are axes=TRUE and add=TRUE. An example is shown below.

Value

The methods do not, in general, return any values.

Author(s)

Jeffrey H. Gove

See Also

plot-methods, graphics::plot, raster::plot

Examples

```
tr = Tract(c(x=25,y=25), cellSize=0.5)
btr = bufferedTract(5, tr)
dlogs = downLogs(20, btr, buttDiams=c(25,35),logLens=c(3,10))
plot(btr, axes=TRUE, bufferColor='lightblue')
plot(dlogs, add=TRUE, showLogCenter=TRUE, logCenterColor='grey30', cex=2)
```

`plot-methods` *Methods for* **graphics** *Function* `plot` *in Package 'sampSurf'*

Description

The methods described here are for the `plot` generic as used in the **sampSurf** package. Each of the methods shown below has different arguments that customize it for a different class of objects found in this package. Method dispatch is based on the signature `x,y` arguments, and often, the latter is "missing" so only the first argument is necessary to associate the appropriate method.

Methods

The methods below are catagorized by their main signature argument, which corresponds to one of the classes in package "sampSurf".

The following arguments may be used in all of the routines below. They share the same definition in each method and are defined as...

- `add`: `TRUE`: add the desired graphics to an existing plot; `FALSE`: create a new plot.
- `asp`: The `plot` `asp` aspect parameter. Best to leave this at the default, which is good for mapping displays.
- `axes`: `TRUE`: plot axes; `FALSE`: no axes.

Some of the routines may specify a different default for these depending upon the desired outcome. Obviously, the defaults can be overridden, and they can be used in routines where they are not explicitly part of the argument definition for the most part.

For the most part, the methods below are organized alphabetically in sections based on the class. The exception is the stem-based Monte Carlo subsampling methods, which are at the end (areal versions are, however, treated within the "InclusionZone" section).

The "ArealSampling" Class

`signature(x = "ArealSampling", y = "missing")` The base method for this class, it only plots the `location` slot of subclass objects. Note that since "ArealSampling" is a virtual class, this method just provides very generic capabilities that will eventually be shared among all subclasses and will never therefore be called outright based on its signature (but only through `callNextMethod`).

usage. . .

```
plot(x,
     pchIZCenter = 20,
     izCenterColor = .StemEnv$izCenterColor,
     asp = 1,
     ... )
```

- `x`: An object that is a subclass of "ArealSampling".
- `pchIZCenter`: The `pch` parameter for the inclusion zone center.
- `izCenterColor`: An R color to be used for the inclusion zone center point location.
- `...`: Other graphics arguments to be passed on to `plot` or `points`.

`signature(x = "circularPlot", y = "missing")` This adds the necessary functionality to plot objects of class "`circularPlot`".

usage. . .

```
plot(x,
     axes = FALSE,
     izColor = .StemEnv$izColor,
     pchPlotCenter = 3,
     showPlotCenter = FALSE,
     showPerimeter = TRUE,
     borderColor = .StemEnv$izBorderColor,
     plotCenterColor = .StemEnv$izCenterColor,
     asp = 1,
     ... )
```

- `x`: An object that is of class "circularPlot" or any subclass that might eventually be defined.
- `izColor`: An R color for the interior of the plot's inclusion zone.
- `pchPlotCenter`: The `pch` value for the plot center.
- `showPlotCenter`: `TRUE`: show the plot center; `FALSE`: do not plot it.
- `showPerimeter`: `TRUE`: show the plot perimeter; `FALSE`: do not plot it. Please note that the only way to show the whole inclusion zone is to show th perimeter. If you want either the interior of the zone or the perimeter but not both, set the color for the one you do not want displayed to `NA`.
- `borderColor`: An R color for the border/perimeter of the plot's inclusion zone.
- `plotCenterColor`: An R color for the center point of the plot's inclusion zone.
- `...`: Other graphics arguments to be passed on to `plot`.

`signature(x = "lineSegment", y = "missing")` This adds the necessary functionality to plot objects of class "`lineSegment`".

usage...

```
plot(x,
     axes = FALSE,
     pchLineCenter = 20,
     showLineCenter = FALSE,
     showLineSegment = TRUE,
     lineColor = .StemEnv$izBorderColor,
     lineCenterColor = .StemEnv$izCenterColor,
     asp = 1,
     ... )
```

- `x`: An object that is of class "lineSegment" or any subclass that might eventually be defined.
- `pchLineCenter`: The `pch` value for the line center.
- `showLineCenter`: `TRUE`: show the line center; `FALSE`: do not plot it.
- `lineColor`: An R color for the line.
- `lineCenterColor`: An R color for the line's center point.
- `...`: Other graphics arguments to be passed on to `plot`.

The "InclusionZone" Class

`signature(x = "InclusionZone", y = "missing")` The base method for this class, it does not plot anything visible other than axes if desired; its role is to set up the extents of the plot from a subclass object's bounding box. Note that since "InclusionZone" is a virtual class, this method just provides very generic capabilities that will eventually be shared among all subclasses and will never therefore be called outright based on its signature (but only through `callNextMethod`).

usage...

```
plot(x,
     axes = FALSE,
     asp = 1,
     ... )
```

- `x`: An object that is a subclass of "InclusionZone".
- `...`: Other graphics arguments to be passed on to `plot`.

`signature(x = "standUpIZ", y = "missing")` This adds the necessary functionality to plot objects of class "`standUpIZ`".

usage...

```
plot(x,
     axes = FALSE,
     showLog = TRUE,
     izColor = .StemEnv$izColor,
     izBorder = .StemEnv$izBorderColor,
     add = FALSE,
     asp = 1,
     ... )
```

- x: An object that is of class “standUpIZ” or any subclass that might be eventualy defined.
- showLog: TRUE: plot the log object; FALSE: do not show it.
- izColor: An R color for the interior of the inclusion zone.
- izBorder: An R color for the border/perimeter of the inclusion zone.
- ...: Other graphics arguments to be passed on to plot, or to the plot methods for the individual objects.

signature(x = "sausageIZ", y = "missing") This adds the necessary functionality to plot objects of class "sausageIZ".

usage...

```
plot(x,
     axes = FALSE,
     showLog = TRUE,
     izColor = .StemEnv$izColor,
     izBorder = .StemEnv$izBorderColor,
     add = FALSE,
     asp = 1,
     ... )
```

- x: An object that is of class “sausageIZ” or any subclass that might be eventualy defined.
- showLog: TRUE: plot the log object; FALSE: do not show it.
- izColor: An R color for the interior of the inclusion zone.
- izBorder: An R color for the border/perimeter of the inclusion zone.
- ...: Other graphics arguments to be passed on to plot, or to the plot methods for the individual objects.

signature(x = "chainSawIZ", y = "missing") This adds the necessary functionality to plot objects of class "chainSawIZ".

usage...

```
plot(x,
     axes = FALSE,
     showLog = TRUE,
     izColor = .StemEnv$izColor,
     izBorder = .StemEnv$izBorderColor,
     showSliver = TRUE,
     ltySliver = 'dashed',
     sliverBorder = 'black',
     sliverColor = transparentColorBase('coral', .StemEnv$alphaTrans),
     showBolt = TRUE,
     ltyBolt = 'dotted',
     boltBorder =  transparentColorBase('grey20', .StemEnv$alphaTrans),
     add = FALSE,
     asp = 1,
     ... )
```

- x: An object that is of class “chainSawIZ” or any subclass that might be eventualy defined.
- showLog: TRUE: plot the log object; FALSE: do not show it.
- izColor: An R color for the interior of the inclusion zone.

- `izBorder`: An R color for the border/perimeter of the inclusion zone.
- `showSliver`: `TRUE`: show the sliver intersection area; `FALSE`: do not show it.
- `ltySliver`: The line type (see: `par`) for the delineation of the sliver.
- `sliverBorder`: An R color for the border/perimeter of the sliver section.
- `sliverColor`: An R color for the interior of the sliver section; remember, this will plot on top of the log interior color.
- `showBolt`: `TRUE`: show the minimal bounding bolt area; `FALSE`: do not show it.
- `ltyBolt`: The line type (see: `par`) for the delineation of the bounding bolt.
- `boltBorder`: An R color for the border/perimeter of the bounding bolt section.
- `...`: Other graphics arguments to be passed on to `plot`, or to the plot methods for the individual objects.

`signature(x = "pointRelascopeIZ", y = "missing")` This adds the necessary functionality to plot objects of class "`pointRelascopeIZ`".

usage...

```
plot(x,
     axes = FALSE,
     showLog = TRUE,
     izColor = .StemEnv$izColor,
     izBorder = .StemEnv$izBorderColor,
     add = FALSE,
     asp = 1,
     showDualCenters = FALSE,
     dcColor = .StemEnv$izBorderColor,
     ... )
```

- `x`: An object that is of class "pointRelascopeIZ" or any subclass that might be eventualy defined.
- `showLog`: `TRUE`: plot the log object; `FALSE`: do not show it.
- `izColor`: An R color for the interior of the inclusion zone.
- `izBorder`: An R color for the border/perimeter of the inclusion zone.
- `showDualCenters`: TRUE: mark the dual circle centers; FALSE: nothing.
- `dcColor`: An R color for the dual circle centers.
- `...`: Other graphics arguments to be passed on to `plot`, or to the plot methods for the individual objects.

`signature(x = "perpendicularDistanceIZ", y = "missing")` This adds the necessary functionality to plot objects of class "`perpendicularDistanceIZ`" and subclasses (e.g., "`omnibusPDSIZ`").

usage...

```
plot(x,
     axes = FALSE,
     showLog = TRUE,
     izColor = .StemEnv$izColor,
     izBorder = .StemEnv$izBorderColor,
     add = FALSE,
     asp = 1,
     ... )
```

- x: An object that is of class "perpendicularDistanceIZ" or subclass.
- showLog: TRUE: plot the log object; FALSE: do not show it.
- izColor: An R color for the interior of the inclusion zone.
- izBorder: An R color for the border/perimeter of the inclusion zone.
- ...: Other graphics arguments to be passed on to plot, or to the plot methods for the individual objects.

signature(x = "distanceLimitedPDSIZ", y = "missing") This adds the necessary functionality to plot objects of class "distanceLimitedPDSIZ" and subclasses (e.g., "omnibusDLPDSIZ" and "hybridDLPDSIZ"). Please note that the different components of the inclusion zone can be displayed to show the transition between protocols (if any). In addition, the overall "perpendicularDistanceIZ" zone can also be displayed, showing what the inclusion zone would look like if the entire log were treated under the PDS protocol.

usage...

```
plot(x,
     axes = FALSE,
     showLog = TRUE,
     izColor = .StemEnv$izColor,
     izBorder = .StemEnv$izBorderColor,
     add = FALSE,
     asp = 1,
     showFullPDSIZ = FALSE,
     showDLSPart = FALSE,
     showPDSPart = FALSE,
     ... )
```

- x: An object that is of class "distanceLimitedPDSIZ" or subclass.
- showLog: TRUE: plot the log object; FALSE: do not show it.
- izColor: An R color for the interior of the inclusion zone.
- izBorder: An R color for the border/perimeter of the inclusion zone.
- showFullPDSIZ: TRUE: display the full PDS inclusion zone (as if the log were sampled fully with PDS); FALSE: do not display it.
- showDLSPart: TRUE: display the "distanceLimitedIZ" portion of the inclusion zone (if any); FALSE: do not display it.
- showPDSPart: TRUE: display the "perpendicularDistanceIZ" portion of the inclusion zone (if any); FALSE: do not display it.
- ...: Other graphics arguments to be passed on to plot, or to the plot methods for the individual objects.

signature(x = "distanceLimitedIZ", y = "missing") This adds the necessary functionality to plot objects of class "distanceLimitedIZ" and subclass "distanceLimitedMCIZ".

usage...

```
plot(x,
     axes = FALSE,
     showLog = TRUE,
     izColor = .StemEnv$izColor,
     izBorder = .StemEnv$izBorderColor,
     add = FALSE,
     asp = 1,
     ... )
```

- x: An object that is of class “distanceLimitedIZ” or subclass.
- showLog: TRUE: plot the log object; FALSE: do not show it.
- izColor: An R color for the interior of the inclusion zone.
- izBorder: An R color for the border/perimeter of the inclusion zone.
- ...: Other graphics arguments to be passed on to plot, or to the plot methods for the individual objects.

signature(x = "circularPlotIZ", y = "missing") This adds the necessary functionality to plot objects of class "circularPlotIZ" as well as those of its subclass "horizontalPointIZ". In addition, the following subclasses derived from “horizontalPointIZ” also use this method: "horizontalPointCMCIZ", "horizontalPointISIZ", "horizontalPointCVIZ" "criticalHeightIZ", "importanceCHSIZ", "antitheticICHSIZ", and "pairedAICHSIZ".

usage...

```
plot(x,
     axes = FALSE,
     showTree = TRUE,
     izColor = .StemEnv$izColor,
     izBorder = .StemEnv$izBorderColor,
     add = FALSE,
     asp = 1,
     ... )
```

- x: An object that is of class “circularPlotIZ” or subclass.
- showTree: TRUE: plot the tree object; FALSE: do not show it.
- izColor: An R color for the interior of the inclusion zone.
- izBorder: An R color for the border/perimeter of the inclusion zone.
- ...: Other graphics arguments to be passed on to plot, or to the plot methods for the individual objects.

signature(x = "horizontalLineIZ", y = "missing") This adds the necessary functionality to plot objects of class "horizontalLineIZ".

usage...

```
plot(x,
     axes = FALSE,
     showLineSegment = TRUE,
     ltyLineSegment = 'dashed',
     showTree = TRUE,
     izColor = .StemEnv$izColor,
     izBorder = .StemEnv$izBorderColor,
     add = FALSE,
     asp = 1,
     ... )
```

- x: An object that is of class “horizontalLineIZ”.
- showLineSegment:TRUE: show the line segment; FALSE: do not show it.

- `ltyLineSegment`: A valid `lty` value of the `plot` parameter.
- `showTree`: `TRUE`: plot the tree object; `FALSE`: do not show it.
- `izColor`: An R color for the interior of the inclusion zone.
- `izBorder`: An R color for the border/perimeter of the inclusion zone.
- `...`: Other graphics arguments to be passed on to `plot`, or to the plot methods for the individual objects.

The "InclusionZoneGrid" Class

`signature(x = "InclusionZoneGrid", y = "missing")` The base method for this class, it actually supplies all of the functionality for all existing subclasses, since the internal "InclusionZone" object will have its own plotting method (see above), which will automatically be used, and this object is all that really differentiates the objects of class "`InclusionZoneGrid`".

usage. . .

```
plot(x,
     axes = TRUE,
     gridColor = .StemEnv$blue.colors(1000),
     asp = 1,
     izColor = NA,
     gridLines = TRUE,
     gridLineColor = .StemEnv$gridLineColor,
     gridCenters = FALSE,
     gridCenterColor = .StemEnv$gridCenterColor,
     estimate = names(c(.StemEnv$puaEstimates, .StemEnv$ppEstimates)),
     lwdGrid = 1,
     ... )
```

- `x`: An object that is of class "InclusionZoneGrid" or any subclass that might be eventualy defined.
- `gridColor`: A valid R color for the background grid. Note that a single color is fine for most sampling methods, but for an inclusion zone whose surface varies within a single object like the "chainSawIZ", you must supply a `colorRampPalette` in order to see the variation correctly.
- `izColor`: An R color for the interior of the inclusion zone. Note that the default is set to `NA` so that no filling is done within the inclusion zone to show the grid more clearly.
- `gridLines`: `TRUE`: show grid lines delineated for more emphasis on the individual grid cells; `FALSE`: no grid lines. Note that this can get busy if the area is too large.
- `gridLineColor`: An R color for the grid lines.
- `gridCenters`: `TRUE`: show grid cell centers within the individual grid cells; `FALSE`: no grid cell centers. Note that this can get busy if the area is too large.
- `gridCenterColor`: An R color for the grid cell centers.
- `estimate`: A character variable corresponding to the desired estimate attribute; these include 'volume', 'Length' (logs), 'Density', 'surfaceArea', 'coverageArea' (logs), 'basalArea' (trees), 'biomass' and 'carbon'. The sample 'depth' surface is now also available.
- `lwdGrid`: The graphics parameter line width (see: `par`) for the grid lines.
- `...`: Other graphical arguments to be passed on to `plot`, or to the plot methods for the individual objects (i.e., "InclusionZone", "downLog").

The “izContainer” Class

`signature(x = "izContainer", y = "missing")` This adds the necessary functionality to plot subclass objects of container class "`izContainer`", including both "`downLogIZs`" and "`standingTreeIZs`" . Note that arguments passed in this call that are destined for individual “downLogIZ” or “downLog” objects within the collection will apply to *all* objects of that class within the collection; similarly for standing tree components.

usage...

```
plot(x,
     axes = FALSE,
     add = FALSE,
     asp = 1,
     ... )
```

- `x`: An object that is of subclass “izContainer”.
- `...`: Other graphics arguments to be passed on to `plot`, or to the plot methods for the individual objects.

The “mirageInclusionZoneGrid” Class

`signature(x = "mirageInclusionZoneGrid", y = "missing")` This method will call the superclass method for the desired “InclusionZoneGrid” object (see below). Please see the superclass method for other arguments that can be passed on.

usage...

```
plot(x,
     tract = NULL,
     tractBorder = 'red',
     lwdTract = 1.25,
     showExtended = FALSE,
     showReflectedSlivers = FALSE,
     colReflections = 'blue',
     ... )
```

- `x`: An object that is of class “mirageInclusionZoneGrid” or any subclass that might be eventualy defined.
- `tract`: An object of class "`mirageTract`".
- `tractBorder`: The color to be used to plot the tract border polygon.
- `lwdTract`: The line width for the tract border polygon.
- `showExtended`: `TRUE`: show the extended grid in the `izGrid.extended` slot of the object; `FALSE`: show the actual grid object with only cells internal to the tract.
- `showReflectedSlivers`: `TRUE`: show the polygons associated with the external portions of the inclusion zone, but reflected back into the tract; `FALSE`: do not show these polygons.
- `colReflections`: The color to be used in displaying the reflected sliver polygons if `showReflectedSlivers=TRUE`.
- `...`: Other graphical arguments to be passed on to superclass method, or to `plot`, or to the plot methods for the individual objects (i.e., “InclusionZone”, “downLog”).

The “sampSurf” Class

`signature(x = "sampSurf", y = "missing")` This is the only extant method for plotting objects of class "`sampSurf`." It makes use of many of the plotting methods for the other objects in the **sampSurf** package, some of which also rely on plot methods from the **raster** package.

usage...

```
plot(x,
     showIZs = TRUE,
     izColor = NA,
     ... )
```

- `x`: An object that is of class “sampSurf” or any subclass that might be eventualy defined.
- `showIZs`: `TRUE`: plot the inclusion zones on the surface; `FALSE`: just plot the surface.
- `izColor`: This should always be `NA` or some very transparent color so the interior of the inclusion zones does not obscure the sampling surface details.
- `...`: Other graphical arguments to be passed on to the plot methods for the individual objects (e.g.., “Tract”, “InclusionZone”, “downLog”).

The “Stem” Class

`signature(x = "Stem", y = "missing")` The base method for this class, it only plots the `location` slot of subclass objects. Note that since “Stem” is a virtual class, this method just provides very generic capabilities that will eventually be shared among all subclasses and will never therefore be called outright based on its signature (but only through `callNextMethod`).

usage...

```
plot(x,
     pchStemLocation = 20,
     stemLocationColor = .StemEnv$logAttributeColor,
     ... )
```

- `x`: An object that is of class “Stem” or any subclass that might be eventualy defined.
- `pchStemLocation`: The `pch` parameter for the location of the stem center.
- `stemLocationColor`: The color for the symbol marking the stem/log location.
- `...`: Other graphics arguments to be passed on to `points`.

`signature(x = "downLog", y = "missing")` This adds the necessary functionality to plot objects of class "`downLog`".

usage...

```
plot(x,
     axes = FALSE,
     logColor = .StemEnv$logColor,
     showLogCenter = FALSE,
     pchLogCenter = 3,
     logCenterColor = .StemEnv$logAttributeColor,
     showNeedle = FALSE,
     logNeedleColor = .StemEnv$logAttributeColor,
     logBorderColor = .StemEnv$logBorderColor,
     asp = 1,
     ... )
```

- x: An object of class “downLog” or any subclass that might eventually be defined.
- logColor: An R color to be used for shading the log.
- showLogCenter: TRUE: plot log center location; FALSE: do not plot it.
- pchLogCenter: The pch value for the log center.
- logCenterColor: An R color to be used for the log center point location.
- showNeedle: TRUE: plot log ‘needle;’ FALSE: do not plot it.
- logNeedleColor: An R color to be used for the log ’needle.’
- logBorderColor: An R color to be used for drawing the perimeter of the log.
- ...: Other graphical arguments to be passed on to plot.

signature(x = "standingTree", y = "missing") This adds the necessary functionality to plot objects of class "standingTree".

usage...

```
plot(x,
     axes = FALSE,
     treeColor = .StemEnv$treeColor,
     showLocation = TRUE,
     pchLocation = 3,       #20 is also good
     locationColor = .StemEnv$treeAttributeColor,
     treeBorderColor = .StemEnv$treeBorderColor,   #log perimeter color
     asp = 1,
     ... )
```

- x: An object of class “downLog” or any subclass that might eventually be defined.
- treeColor: An R color to be used for shading the tree.
- showLocation: TRUE: plot tree center location; FALSE: do not plot it.
- pchLocation: The pch value for the tree center.
- locationColor: An R color to be used for the tree center point location.
- treeBorderColor: An R color to be used for drawing the perimeter of the tree.
- ...: Other graphical arguments to be passed on to plot.

The “StemContainer” Class

signature(x = "StemContainer", y = "missing") The base method for this class. Note that since “StemContainer” is a virtual class, this method just provides very generic capabilities that are shared among all subclasses and will never therefore be called outright based on its signature (but only through callNextMethod). Currently, it just sets up the overall extents for the plot area.

usage...

```
plot(x,
     axes = FALSE,
     add = FALSE,
     asp = 1,
     ... )
```

- x: An object that is of class “Stem” or any subclass that might be eventualy defined.
- ...: Other graphics arguments to be passed on.

`signature(x = "downLogs", y = "missing")` This adds the necessary functionality to plot objects of container class "`downLogs`". Note that arguments passed in this call that are destined for individual “downLog” objects within the collection will apply to *all* logs within the collection.

usage...

```
plot(x,
     axes = FALSE,
     add = FALSE,
     asp = 1,
     ... )
```

- `x`: An object that is of class “downLogs” or subclass thereof.
- `...`: Other graphics arguments to be passed on to `plot`, or to the plot methods for the individual downLog objects (please see above).

`signature(x = "standingTrees", y = "missing")` This adds the necessary functionality to plot objects of container class "`standingTrees`". Note that arguments passed in this call that are destined for individual “standingTree” objects within the collection will apply to *all* trees within the collection.

usage...

```
plot(x,
     axes = FALSE,
     add = FALSE,
     asp = 1,
     ... )
```

- `x`: An object that is of class “standingTrees” or subclass thereof.
- `...`: Other graphics arguments to be passed on to `plot`, or to the plot methods for the individual standingTree objects (please see above).

The “Tract” Class

`signature(x = "Tract", y = "missing")` This method is the base plot method for objects of class "`Tract`" and subclasses. Because “Tract” objects are a subclass of "`RasterLayer`," all of the plotting is done through calls to those methods.

usage...

```
plot(x,
     axes = TRUE,
     gridColor = .StemEnv$blue.colors(1000),
     asp = 1,
     useImage = TRUE,
     ... )
```

- `x`: An object that is of class “Tract” or subclass thereof.
- `gridColor`: A valid R color for the background grid. Note that a single color is fine for initial blank grids, but you can also supply a function from `colorRampPalette` in order to see and surface variation that is in the tract correctly; this is the default in shades of blue.

- useImage: TRUE: use the raster::image command to draw the tract (default); FALSE: use the raster::plot command instead, which gives the gradient legend. This is an option because the latter will not rezise the graphics correctly on a graphics window, though it is fine for a hardcopy and if you do not resize the device. This may be fixed in raster eventually (Dec-2010).
- ...: Other graphical arguments to be passed on to raster::plot. In addition, one can add grid lines to the tract via arguments passed to gridCellEnhance.

signature(x = "bufferedTract", y = "missing") This is the plot method for objects of class "bufferedTract" and subclasses. It essentially calls the bas "Tract" method for all of the work and adds the buffer.

usage...

```
plot(x,
    bufferColor = transparentColorBase('blanchedalmond', .StemEnv$alphaTrans),
      axes = TRUE,
      gridColor = .StemEnv$blue.colors(1000),
      lwd = 2,
      asp = 1,
      ... )
```

- x: An object that is of class "bufferedTract" or subclass thereof.
- bufferColor: The color for the buffer area.
- gridColor: A valid R color for the background grid. Note that a single color is fine for initial blank grids, but you can also supply a function from colorRampPalette in order to see and surface variation that is in the tract correctly; this is the default in shades of blue.
- lwd: The line width for the border (see: par).
- ...: Other graphical arguments to be passed on to raster::plot. In addition, one can add grid lines to the tract via arguments passed to gridCellEnhance.

The "MonteCarloSampling" Class

signature(x = "crudeMonteCarlo", y = "missing") The "importanceSampling" and "controlVariate" sampling classes use this method as well.

usage...

```
plot(x,
      axes = FALSE,
      renderAs = c('profile', 'crossSection'),
      isHeightColor = .StemEnv$isHeightColor,
      ... )
```

- x: An object that is of class "crudeMonteCarlo" or subclass thereof.
- renderAs: 'profile' to see the sampling locations along the stem in a profile view; 'crossSection' will display a cross-sectional view of the same information.
- isHeightColor: A legal R color to use for drawing the sampling point locations.
- ...: Other graphical arguments to be passed on to plot.

The “mcsContainer” Class

`signature(x = "mcsContainer", y = "missing")` Display the true versus estimated volumes graphically. Please note that this method applies to objects of class “antitheticContainer” as well.

usage...

```
plot(x,
     xlab = 'Estimated volume',
     ylab = 'True volume',
     showDiagonal = TRUE,
     ... )
```

- `x`: An object that is of class “mcsContainer”.
- `showDiagonal`: `TRUE`: Plot the 1:1 line for reference.
- `...`: Other graphical arguments to be passed on to `plot`.

`plot3D` *Interactive 3D Plot of Objects From Package “sampSurf”*

Description

This routine uses the “`rgl`” package to display objects of class "`Tract`" or subclass, "`InclusionZoneGrid`" or subclass, and "`sampSurf`" objects in a three dimensional representation of the surface.

Details

This is not the generic function, that is defined in rasterVis::`plot3D`, which should be referred to for more details on both possible base arguments and functionality. Please also see the `plot3D-methods` for other possible arguments added for “sampSurf” class functionality.

Value

Nothing returned.

Note

One can generate a hardcopy of a plot using the appropriate commands in the “`rgl`” package.

Author(s)

Jeffrey H. Gove

See Also

rasterVis::`plot3D`

Examples

```
## Not run:
#
# create a buffered tract, sampling surface, and then display...
#
bufftr = bufferedTract(10, Tract(c(x=100,y=100),cellSize=0.5,units='metric'))
ss = sampSurf(25,bufftr,iZone='sausageIZ',plotRadius=4)
require(rgl)
plot3D(ss)

## End(Not run)
```

`plot3D-methods` *Methods for Function* `plot3D` *in Package* **sampSurf**

Description

The methods described here are for classes defined in the **sampSurf** package, based on the generic defined in the rasterVis::`plot3D` package. Please see the generic and its methods in the **rasterVis** package for definitions of other arguments that are not described here. Especially please note below that the `col` argument is a pallette generating *function*, not a simple R color. Also, the ... argument will pass along arguments defined in the generic function as usual.

Methods

`signature(x = "InclusionZoneGrid")`

usage...

```
plot3D(x,
       estimate = 'volume',
       col = .StemEnv$blue.colors,
       ...)
```

- `x`: An object that is of class “InclusionZoneGrid”.
- `estimate`: Any legal attribute estimate that is found in the object’s `data` slot. This will normally include the `.StemEnv$puaEstimates` and `.StemEnv$ppEstimates` attributes, though not all may apply to any given sampling method.

`signature(x = "sampSurf")`

usage...

```
plot3D(x,
       col = .StemEnv$blue.colors,
       ...)
```

- `x`: An object that is of class “sampSurf”.

`signature(x = "Tract")`

usage...

```
plot3D(x,
       col = .StemEnv$blue.colors,
       ...)
```

- x: An object that is of class "Tract" or subclass.

`pointRelascope` *Generate Objects of Class "pointRelascope"*

Description

This generic function has only one method used as a constructor function for objects that are of class "`pointRelascope`". This method should be used in preference to `new` to insure a valid object.

Usage

```
pointRelascope(angleDegrees, ...)
```

Arguments

`angleDegrees`	This is the signature argument: the relascope angle in degrees.
`...`	Arguments that are defined in `pointRelascope-methods`

Details

Only one method currently exists for object generation. Its arguments are documented in `pointRelascope-methods`.

Value

A valid object of class "`pointRelascope`"

Author(s)

Jeffrey H. Gove

See Also

`pointRelascope-methods` and the "`ArealSampling`" class.

Examples

```
#
#  create an object with a with 4:1 reach:width factor angle...
#
(angle = .StemEnv$rad2Deg(2*atan(1/4)))
prs = pointRelascope(angle, units='English')
prs
```

pointRelascope-class *Class* "pointRelascope"*: Point Relascope Sampling*

Description

A subclass of "ArealSampling" that can be used to create objects that encapsulate all the parameters necessary for point relascope sampling of down woody debris.

Objects from the Class

Objects can be created by calls of the form `new("pointRelascope",...)`, and while this is reasonable with this class because there are no graphical slots, it is still not recommended. The preferred method for creating new objects is via the `pointRelascope` constructor.

Slots

In addition to those slots in class "ArealSampling," the following are defined…

`angleDegrees`**:** Object of class "numeric": The relascope angle in degrees such that $0 < \nu \leq 90$ degrees.

`angleRadians`**:** Object of class "numeric": The relascope angle in radians.

`phi`**:** Object of class "numeric": The area factor multiplier, ϕ, for angle ν, as described in the references below.

`slFactor`**:** Object of class "numeric": The squared length factor, L, defining the constant amount of length-square per unit area (acre or hectare) as described in the references below.

`rwFactor`**:** Object of class "numeric": The reach:width ratio or factor that makes it simpler to keep track of some of the more useful relascope angles, especially when constructing a relascope.

Extends

Class "ArealSampling", directly.

Methods

summary `signature(object = "pointRelascope")`: prints a summary of the object

Author(s)

Jeffrey H. Gove

References

J. H. Gove, A. Ringvall, G. Stahl, and M. J. Ducey. 1999. Point relascope sampling of downed coarse woody debris. *Canadian Journal of Forest Research* **29**(11):1718–1726.

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

See Also

The "ArealSampling" class.

Examples

```
showClass("pointRelascope")
```

`pointRelascope-methods` *Methods for "pointRelascope" Object Construction in Package 'sampSurf'*

Description

There is currently only one method based on the `pointRelascope` generic that is used for object construction. It is detailed below.

Methods

`signature(angleDegrees = "numeric")` This method takes the relascope angle in degrees as the signature argument along with other optional aruments described as follows...

usage...

```
pointRelascope(angleDegrees,
               units = 'metric',
               description = 'point relascope method',
               ...)
```

- `angleDegrees`: The relascope angle in degrees.
- `units`: Either "English" or "metric".
- `description`: A character vector description of the object.

`pointRelascopeIZ` *Generate Objects of Class "*`pointRelascopeIZ`*"*

Description

This is the generic definition for generating objects of class "pointRelascopeIZ." There is only one constructor method corresponding to this generic: `pointRelascopeIZ-methods`.

Usage

```
pointRelascopeIZ(downLog, prs, ...)
```

Arguments

`downLog`	Signature object of class "`downLog`".
`prs`	Signature object of class `pointRelascope` containing the pertinent point relascope sampling information.
`...`	See methods.

Details

Since only one method exists for this generic, its signature arguments coincide with the above definitions. Please see `pointRelascopeIZ-methods` for more details.

Value

A valid object of class "`pointRelascopeIZ`."

Author(s)

Jeffrey H. Gove

References

J. H. Gove, A. Ringvall, G. Stahl, and M. J. Ducey. 1999. Point relascope sampling of downed coarse woody debris. *Canadian Journal of Forest Research* **29**(11):1718–1726.

See Also

Class "`pointRelascopeIZ`", and `pointRelascopeIZ-methods`.

Examples

```
dl = downLog(buttDiam=15, solidType=4, logAngle=pi/3, logLen=10, units='English')
(angle = .StemEnv$rad2Deg(2*atan(1/4)))
prs = pointRelascope(angle, units='English')
iz.prs = pointRelascopeIZ(dl, prs=prs)
summary(iz.prs)
plot(iz.prs, axes=TRUE, showDualCenters=TRUE, cex=2, showNeedle=TRUE)
```

`pointRelascopeIZ-class`

Class "pointRelascopeIZ"

Description

This class holds the inclusion zone definition for the ‘point relascope’ method (Gove et. al, 1999) for sampling down coarse woody debris.

Objects from the Class

Objects can be created by calls of the form `new("pointRelascopeIZ",...)`. However, this is not recommended because the objects are fairly complex. Instead, one can use the object constructor `pointRelascopeIZ` to create new objects.

Slots

Most of the slots are described in the superclasses (see below), please see their help pages for more information. This class adds a few slots to the "downLogIZ" class specification.

`prs`**:** Object of class `"pointRelascope"`: This supplies the information on the relascope angle, etc., used for constructing the inclusion zone.

`dualCircle`**:** Object of class `"matrix"`: A matrix representation of the dual circle inclusion zone in homogeneous coordinates. This can be manipulated and plotted as desired for easy access to the inclusion zone where needed.

`radius`**:** Object of class `"numeric"`: The radius of each of the dual circles in the inclusion zone.

`area`**:** Object of class `"numeric"`: The exact area of the inclusion zone.

`perimeter`**:** Object of class `"SpatialPolygons"`: This is the inclusion zone perimeter as a “SpatialPolygons” object.

`pgDualArea`**:** Object of class `"numeric"`: This is the area of the dual circle 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, and adjust if necessary the number of points defining the inclusion area perimeter.

`dualCenters`**:** Object of class `"matrix"` The center points of each of the dual circles as a “SpatialPoints” object.

Extends

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

Methods

izGrid `signature(izObject = "pointRelascopeIZ",tract = "Tract")`: “InclusionZoneGrid” generic constructor

perimeter `signature(object = "pointRelascopeIZ")`: Return the object perimeter

plot `signature(x = "pointRelascopeIZ",y = "missing")`: Plot the object

summary `signature(object = "pointRelascopeIZ")`: Object summary

Author(s)

Jeffrey H. Gove

References

J. H. Gove, A. Ringvall, G. Stahl, and M. J. Ducey. 1999. Point relascope sampling of downed coarse woody debris. *Canadian Journal of Forest Research* **29**(11):1718–1726.

See Also

downLogIZs container class

Examples

```
showClass("pointRelascopeIZ")
```

---

pointRelascopeIZ-methods *Method for "pointRelascopeIZ" Object Construction in Package* **sampSurf**

Description

This is the one method for generic function pointRelascopeIZ in Package 'sampSurf' that allows for creation of objects of class "pointRelascopeIZ."

Methods

signature(downLog = "downLog", prs = "pointRelascope")

usage...

```
pointRelascopeIZ(downLog,
                 prs,
                 nptsCircle = 100,
           description = 'inclusion zone for down log point relascope sampling method',
           spID = paste('prsIZ',format(runif(1,0,10000),digits=8),sep=':'),
                 spUnits = CRS(projargs=as.character(NA)),
                 ... )
```

- downLog: An object of class "downLog" for which the inclusion zone is to be determined.
- prs: An object of class "pointRelascope" that will contain the correct relascope angle, etc., for constructing the inclusion zone.
- nptsCircle: The number of points to use in the construction of each of the dual circles comprising the inclusion area.
- description: A character vector description of the object.
- spID: A unique identifier that will be used in displaying the spatial polygon for the inclusion zone component of the object.
- spUnits: A valid CRS object specifying the Coordinate Reference System. This defaults to NA, which means you want to use your own user-defined system, say for a sample plot located in the field.
- dots: Arguments to be passed on.

sampleLogs *Generates a Synthetic Population of Down Log Attributes*

Description

This routine will allow the generation of a collection of synthetic down log *attributes* in a `data.frame`. Note that the function returns a collection of log attributes in the sense of dimensions, etc., and *not* a collection of `downLog` objects, although it can be used to generate one if desired (see examples). Please see “The Stem Class” vignette for more details.

Usage

```
sampleLogs(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,
           ...)
```

Arguments

`nLogs`	The number of logs in the collection.
`buttDiams`	A length-two vector specifying the *range* of butt (large-end) diameters from which to uniformly draw the sample.
`topDiams`	A length-two vector specifying the *range* of top (small-end) diameters, in the form of a *proportion* of the `buttDiam` diameters, from which to uniformly draw the sample.
`logLens`	A length-two vector specifying the *range* of log lengths from which to uniformly draw the sample.
`logAngles`	A length-two vector specifying the *range* of log angles from which to draw the log lie angles; these are always counterclockwise relative to due East.
`solidTypes`	A length-two vector specifying the *range* in the log shape parameter for the default taper and volume equations; where: 1-2 is a neiloid, 2 is a cone, and >2 is a paraboloid.
`species`	A character vector of possible species from which to draw the sample uniformly. This can be any legal character string, and so can include codes, names, Latin names, etc.
`sampleRect`	A rectangle delineating the area within which the log centers will be sampled from. It must be in the form of a valid `bbox` matrix. If `NULL` or `NA`, a unit square bounding box will be created.

`startSeed`	A numeric start seed for the random number generator. `NA` will continue with the current random number stream if available, or create one if not.
`runQuiet`	`TRUE`: No feedback; `FALSE`: a little feedback.
`...`	Not used at present, just gobbles up other arguments.

Details

This routine does not care about units, it is up to the user to make sure that the ranges for diameters and lengths are reasonable for whatever system (i.e., "English" or "metric") one is working in. For example, if metric units are used, `buttDiams` would be in centimeters, while `logLens` would be in meters.

Value

A data frame with columns: `species,logLen,buttDiam,topDiam,solidType,x,y,logAngle,logAngle.D` where `x,y` is the center point of the log and `logAngle.D` is in degrees.

Author(s)

Jeffrey H. Gove

See Also

`downLogs-methods`

Examples

```
#
#  draw from a unit square...
#
sampleLogs(species=c('ewp','sm','978'),buttDiams=c(1,10), logLens=c(3,6))
#
# draw from a buffer...
#
bufftr = bufferedTract(10, Tract(c(x=50,y=50),cellSize=0.5))
sl = sampleLogs(10, sampleRect=bufftr@bufferRect, buttDiams=c(25,40))
dlogs = downLogs(sl)
plot(bufftr, axes=TRUE, gridColor='grey80')
plot(dlogs, add=TRUE)
```

`sampleTrees` *Generates a Synthetic Population of Standing Tree Attributes*

Description

This routine will allow the generation of a collection of synthetic standing tree *attributes* in a `data.frame`. Note that the function returns a collection of tree attributes in the sense of dimensions, etc., and *not* a collection of `standingTree` objects, although it can be used to generate one if desired (see examples). Please see “The Stem Class” vignette for more details.

Usage

```
sampleTrees(nTrees = 2,
            dbhs = c(8, 40),
            topDiams = c(0.4, 0.9),
            heights = c(5, 15),
            solidTypes = c(1, 10),
            species = .StemEnv$species,
            sampleRect = NULL,
            startSeed = NA,
            runQuiet = FALSE,
            ...)
```

Arguments

`nTrees`	The number of trees in the collection.
`dbhs`	A length-two vector specifying the *range* of breast height diameters from which to uniformly draw the sample.
`topDiams`	A length-two vector specifying the *range* of top diameters, in the form of a *proportion* of the `dbh` diameters, from which to uniformly draw the sample.
`heights`	A length-two vector specifying the *range* of tree heights from which to uniformly draw the sample.
`solidTypes`	A length-two vector specifying the *range* in the tree shape parameter for the default taper and volume equations; where: 1-2 is a neiloid, 2 is a cone, and >2 is a paraboloid.
`species`	A character vector of possible species from which to draw the sample uniformly. This can be any legal character string, and so can include codes, names, Latin names, etc.
`sampleRect`	A rectangle delineating the area within which the tree centers (pith at dbh) will be sampled from. It must be in the form of a valid `bbox` matrix. If `NULL` or `NA`, a unit square bounding box will be created.
`startSeed`	A numeric start seed for the random number generator. `NA` will continue with the current random number stream if available, or create one if not.
`runQuiet`	`TRUE`: No feedback; `FALSE`: a little feedback.
`...`	Not used at present, just gobbles up other arguments.

Details

This routine does not care about units, it is up to the user to make sure that the ranges for diameters and heights are reasonable for whatever system (i.e., "English" or "metric") one is working in. For example, if metric units are used, `dbhs` would be in centimeters, while `heights` would be in meters.

Value

A data frame with columns: `species,height,dbh,topDiam,solidType,x,y` where `x,y` is the center point of the tree pith at `dbh`.

Author(s)

Jeffrey H. Gove

See Also

`standingTrees-methods`

Examples

```
#
#  draw from a unit square (metric default)...
#
sampleTrees(species=c('ewp','sm','978'),dbhs=c(4,10), heights=c(3,6))
#
# draw from a buffer...
#
bufftr = bufferedTract(10, Tract(c(x=50,y=50),cellSize=0.5))
st = sampleTrees(10, sampleRect=bufftr@bufferRect, dbhs=c(20,40))
strees = standingTrees(st)
plot(bufftr, axes=TRUE, gridColor='grey80')
plot(strees, add=TRUE)
```

`sampSurf` *Generate Objects of Class "`sampSurf`"*

Description

This is the generic definition for generating objects of class "sampSurf." There are several methods corresponding to this generic that may be found in `sampSurf-methods`. One of the constructor methods in particular, averts the necessity for being familiar with the other classes in the package and can simply be run from scratch, requiring only an extant "`Tract`" object on which to draw the sample and compute the surface. More details and examples are found in *"The sampSurf Class"* vignette.

Usage

```
sampSurf(object, tract, ...)
```

Arguments

`object`	Signature object, which differs for each method.
`tract`	Signature object of class "`Tract`" or subclass.
`...`	See methods.

Details

The generation of objects of this class depends on the signature of the method used. One common thread is that all methods require an existing object of class "`Tract`" or one of its subclasses to be available. This will be the grid area over which the population is to be simulated and sampling surface determined. It is often convenient to set this area up to be equal to one acre or one hectare. However, one could match each "tract" to the size of a sample plot where logs or trees have been measured in the field, for example. And there is no reason why the tract must be restricted to a smaller size, except for the limitations of computer memory. The **raster** package routines should help with this because they will store large rasters (tracts) on disk, but I have not had the chance to test this out sufficiently yet. In any case, one can start from scratch and make all of the intervening objects necessary to create a sampling surface with the appropriate constructor method. Alternatively, one can bypass all the intermediate steps and use the constructor that takes the number of individual stems and a tract object, and let it do all of the work. These are both explained in the vignette sited above and in the `sampSurf-methods`.

Value

A valid object of class "`sampSurf`."

Author(s)

Jeffrey H. Gove

References

"The sampSurf Class" vignette.

See Also

`sampSurf-package`, `sampSurf-class`, `sampSurf-methods`

Examples

```
#
#  create a sampling surface with 3 logs using the "sausage" method...
#
tr = Tract(c(x=50,y=50), cellSize=0.5)
btr = bufferedTract(10, tr)
ssSA = sampSurf(3, btr, iZone = 'sausageIZ', plotRadius=3,
       buttDiam=c(30,50), startSeed=1001)
plot(ssSA, axes=TRUE)
summary(ssSA)
```

`sampSurf-class` *Class "sampSurf"*

Description

This class allows one to collect a population of "`Stem`" (actually subclasses thereof) objects along with their inclusion zones for a given sampling method or protocol and generate the sampling surface resulting from the method. One can think of the sampling surface grid as generating all possible samples of size one over the population based on the grid cell size. That is, the number of sample point locations is equivalent to the number of grid cells. At each grid cell, an estimate based on one sample is recorded. Because inclusion zones will often overlap, the estimate at the individual grid cells will be a function of those logs whose inclusion zones overlapped the grid cell center. In this way, the surface is constructed representing the samples at each point. One can think of the surface as a discrete approximation to the actual continuous surface—the approximation getting better as the grid cell size goes to zero (grid resolution increases).

Much of this is explained in far more detail in the references below. One can also generate multiple copies based on different log populations and average those surfaces to get a result that minimizes any possible irregularities in a particular sample draw of “Stem” objects. This has not been built into the package yet, but is simple enough to do, and can be added later.

Note that only one attribute (e.g., volume, surface area, etc.) can be represented for a given sampling surface. However, it is trivial to use the same population of logs and inclusion zones to estimate a surface for a different attribute on the same tract. It is also straightforward to use the same population of logs and base tract extents to compare different sampling methods or protocols.

Objects from the Class

Objects can be created by calls of the form `new("sampSurf",...)`. However, this is not recommended because the objects are fairly complex. Instead, one can use the object constructor `sampSurf` to create new objects. There is more than than one form of the function, so please refer to its methods for alternatives. More details are found in *"The sampSurf Class"* vignette.

Slots

`description`**:** Object of class `"character"`: a character description to be associated with this surface.

`izContainer`**:** Object of class `"izContainer"`: A collection or population of down log or standing tree "`InclusionZone`" objects.

`tract`**:** Object of class `"Tract"`: This is the tract over which to generate the sampling surface, and on which the objects in the `izContainer` slot exist.

`estimate`**:** Object of class `"character"`: The attribute estimate name for the sampling surface represented on this tract; i.e., "volume", or number of stems ("Density"), etc., per unit area.

`surfStats`**:** Object of class `"list"`: A list object with the summary statistics for the sampling surface with estimate attribute in the `estimate` slot.

Methods

bbox `signature(obj = "sampSurf")`: Retrieve the bounding box matrix.

hist `signature(x = "sampSurf")`: Histogram of the sampling distribution.

perimeter `signature(object = "sampSurf")`: Graphical perimeter of the surface.

plot3D `signature(x = "sampSurf")`: Display the surface in 3D with `rgl`.

plot `signature(x = "sampSurf",y = "missing")`: Plot the surface.

show `signature(object = "sampSurf")`: `raster`-like description of the surface grid.

summary `signature(object = "sampSurf")`: Detailed summary of the surface statistics.

Author(s)

Jeffrey H. Gove

References

Williams, M. S. 2001. New approach to areal sampling in ecological surveys. *Forest Ecology and Management* **154**:11–22.

Williams, M. S. 2001. Nonuniform random sampling: an alternative method of variance reduction for forest surveys. *Canadian Journal of Forest Research* **31**: 2080–2088.

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

See Also

`sampSurf-package`, `sampSurf`, `sampSurf-methods`

Examples

```
showClass("sampSurf")
```

`sampSurf-methods` *Methods for "sampSurf" object construction in Package 'sampSurf'*

Description

The following methods will construct valid objects of class "`sampSurf`". Please see *"The sampSurf Class"* vignette for more details and examples.

Methods

`signature(object = "izContainer", tract = "Tract")` This constructor will allow the most flexibility. One can create all of the individual objects from generating a population of "Stem" objects up through the corresponding collection of inclusion zones and give the latter, along with the “Tract” object to this constructor to generate the surface.

usage...

```
sampSurf(object,
         tract,
        estimate = unlist(c(.StemEnv$puaEstimates, .StemEnv$ppEstimates)),
         description = 'sampling surface object',
         runQuiet = FALSE,
         ... )
```

- `object`: An object of one of the subclasses of "izContainer."
- `tract`: An object of class "Tract" or subclass.
- `estimate`: A character variable corresponding to the desired estimate attribute; these include ’volume’, ’Length’ (logs), ’Density’, ’surfaceArea’, ’coverageArea’ (logs), ’basalArea’ (trees), ’biomass’ and ’carbon’. The sample ’depth’ surface is now also available.
- `description`: A description of the object as a character string.
- `runQuiet`: This routine will print a report of the grid cells as it visits them; set this to `TRUE`: (default) to see the report; `FALSE`: to run quietly, which is useful for larger simulations.
- ... : Other arguments to be passed along–not used at present.

`signature(object = "numeric", tract = "Tract")` This constructor masks much of the job of creating a sampling surface. Just specify the number of stems and a tract for signature arguments along with any options to generate a surface. All arguments except those listed below are as in the above constructor. Note that this constructor eventually calls the first constructor detailed above to generate the sampling surface.

usage...

```
sampSurf(object,
         tract,
         iZone,
         estimate = unlist(.StemEnv$puaEstimates),
         description = 'sampling surface object',
         runQuiet = FALSE,
         ...
        )
```

- `object`: The number of "Stem" subclass objects (i.e., down logs) to generate in the population.
- `iZone`: The type of inclusion zone or sampling method/protocol desired. The routine will determine whether to generate logs or trees based on this argument.
- ... : Other arguments to be passed along to routines that construct the downlog population, the inclusion zones, and eventually the first “sampSurf” constructor given above. For example, one can pass arguments affecting how the sample log population will be drawn as shown in the vignette mentioned above (e.g., `logLens=c(4,12)`).

sausageIZ *Generate Objects of Class "sausageIZ"*

Description

This is the generic definition for generating objects of class "sausageIZ." There is only one method corresponding to this generic: sausageIZ-methods.

Usage

```
sausageIZ(downLog, plotRadius, ...)
```

Arguments

downLog	Signature object of class "downLog".
plotRadius	Signature object for plot radius.
...	See methods.

Details

Since only one method exists for this generic, its signature arguments coincide with the above definitions. Please see sausageIZ-methods for more details.

Value

A valid object of class "sausageIZ."

Author(s)

Jeffrey H. Gove

References

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

See Also

Class "sausageIZ", and sausageIZ-methods.

Examples

```
dl = downLog(buttDiam=40, solidType=4, logAngle=4*pi/3, logLen=6)
iz.sa = sausageIZ(dl, plotRadius=3)
summary(iz.sa)
plot(iz.sa, axes=TRUE, showPlotCenter=TRUE, cex=2, showNeedle=TRUE)
```

`sausageIZ-class` *Class "sausageIZ"*

Description

This class holds the inclusion zone definition for the 'sausage' method (Gove and Van Deusen, 2011) for sampling down coarse woody debris.

Objects from the Class

Objects can be created by calls of the form `new("sausageIZ",...)`. However, this is not recommended because the objects are fairly complex. Instead, one can use the object constructor `sausageIZ` to create new objects.

Slots

Most of the slots are described in the superclasses (see below), please see their help pages for more information. This class adds a few slots to the "downLogIZ" class specification.

`sausage`**:** Object of class `"matrix"`: 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`**:** Object of class `"numeric"`: The radius for the fixed-area plot that determines the sausage inclusion zone area. This will be in units of feet for "English" or meters for "metric."

`area`**:** Object of class `"numeric"`: The exact area of the inclusion zone.

`perimeter`**:** Object of class `"SpatialPolygons"` This is the inclusion zone perimeter as a "SpatialPolygons" object.

`pgSausageArea`**:** Object of class `"numeric"`: 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.

Extends

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

Methods

izGrid `signature(izObject = "sausageIZ",tract = "Tract")`: "InclusionZoneGrid" generic constructor

perimeter `signature(object = "sausageIZ")`: Return the object perimeter

plot `signature(x = "sausageIZ",y = "missing")`: Plot the object

summary `signature(object = "sausageIZ")`: Object summary

Author(s)

Jeffrey H. Gove

References

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

See Also

`standUpIZ`, `chainSawIZ`, `fullChainSawIZ` and the `downLogIZs` container class.

Examples

```
showClass("sausageIZ")
```

`sausageIZ-methods` *Method for "sausageIZ" object construction in Package 'sampSurf'*

Description

This is the one method for generic function `sausageIZ` in Package 'sampSurf' that allows for creation of objects of class "`sausageIZ`."

Methods

`signature(downLog = "downLog", plotRadius = "numeric")`

usage...

```
sausageIZ(downLog,
          plotRadius,
          nptsHalfCircle = 50,
       description = 'inclusion zone for dowed log "sausage" sampling method',
          spID = unlist(strsplit(tempfile('sausageIZ:',''),'\/'))[2],
          spUnits = CRS(projargs=as.character(NA)),
          ... )
```

- `downLog`: An object of class "`downLog`" which the inclusion zone is to be determined for under the sausage method.
- `plotRadius`: The radius of the circular fixed-area plot in the correct units: feet for "English" and meters for "metric."
- `nptsHalfCircle`: The number of points to use in the half-circle at each end of the sausage-shaped inclusion zone for the polygon representation of the object.
- `description`: A character vector description of the object.
- `spID`: A unique identifier that will be used in displaying the spatial polygon component of the object.

- `spUnits`: A valid `CRS` object specifying the Coordinate Reference System. This defaults to `NA`, which means you want to use your own user-defined system, say for a sample plot located in the field.
- `dots`: Arguments to be passed on.

`segmentVolume` *Determine segment volumes within a "`Stem`" Object*

Description

This generic will allow the determination of segment volumes within individual "`downLog`" or "`standingTree`" objects.

Usage

```
segmentVolume(object, ...)
```

Arguments

`object`	See methods for the possibilites of this signature argument.
`...`	Just gobbled for now.

Details

If the object's `solidType` slot is not `NULL`, then the default taper equation is used to determine the volume based on the object's `solidType`. Otherwise, the segment volume is determined by fitting a spline to the object's taper data. Please see the methods for more information on usage: `segmentVolume-methods`

Value

The scalar volume of the segment chosen.

Author(s)

Jeffrey H. Gove

See Also

The `wbVolume` and `splineVolume` code within .StemEnv.

Examples

```
dl = downLog(buttDiam=12, topDiam=2, logLen=10, units="English")
segmentVolume(dl, segBnds=c(3,8))
```

segmentVolume-methods *Methods for Function* `segmentVolume` *in Package* **sampSurf**

Description

There are three methods for the `segmentVolume` generic function keyed off the one `object` signature. Each of them share the same argument list, but with one argument having slightly different meanings between the methods. The method associated with `list` objects should *not* be used unless you know what you are doing. Please use the wrappers in the other two methods below, which set the correct list structure up, and call the `list` version to do all the calculations.

Methods

`signature(object = "downLog")`

usage...

```
segmentVolume(object,
              segBnds = c(low=0,  up=object@logLen),
              ... )
```

- `object`: An object of class "downLog".
- `segBnds`: The lower and upper segment bounds in terms of log length in the correct units.

`signature(object = "list")` Please do not use this method, use the other two which call it correctly...

usage...

```
segmentVolume(object,
              segBnds = NULL,
              ... )
```

- `object`: An object of class "list".
- `segBnds`: The lower and upper segment bounds in terms of log length in the correct units. The default, `NULL`, will return total stem volume.

`signature(object = "standingTree")`

usage...

```
segmentVolume(object,
              segBnds = c(low=0,  up=object@height),
              ... )
```

- `object`: An object of class "standingTree".
- `segBnds`: The lower and upper segment bounds in terms of tree height in the correct units.

smithPlot	*A function to make H. F. Smith plots*

Description

This function will create so-called "Smith plots" from the simulation results of several objects of class sampSurf. The simulation results should be stored in a named list structure as described below for input to this routine. Please see *Smith (1938)* for the original study and *Gove (2017)* for a recent application.

Usage

```
smithPlot(hfs,
          showPlot = TRUE,
          ylab = "Total surface variance",
          xlab = "Average Inclusion Zone Area",
          type = "b",
          pch = 19,
          theme = c("ggplot", "custom", "plain", "economist"),
          cols = c("#00526D", "#00A3DB", "#7A2713", "#939598"),
          ...)
```

Arguments

hfs	A named list of named lists with the following structure... top = sampling methods; these should be named appropriately, e.g., 'hps', 'chs', etc. sub = within each sampling method, a list of sampSurf objects, one for each different plot size/baf, etc.; e.g., 'HPSbaf5', 'CHSbaf7', etc.
showPlot	TRUE: display the plot; FALSE: create and return the plot only.
ylab	Label for y-axis – see par.
xlab	Label for x-axis – see par.
type	See par.
pch	See par.
theme	A theme from the provided list – see **latticeExtra** 'styles' for more information.
cols	A vector of alternative colors for the line graphs; NULL will use the default theme colors; those listed above are essentially those from the economist theme.
...	Passed on to xyplot.

Details

A conceptual example setup for the hfs argument as used in the manuscript cited below would be...

$hps
[1] "HPSbaf3" "HPSbaf5" "HPSbaf7" "HPSbaf9"
$chs
[1] "CHSbaf3" "CHSbaf5" "CHSbaf7" "CHSbaf9"
$cmc
[1] "CMCbaf3" "CMCbaf5" "CMCbaf7" "CMCbaf9"
$cps
[1] "CPSrad5" "CPSrad6" "CPSrad7" "CPSrad9"

where the "$" names are the 'top' list entries, and then each with its own corresponding named 'sub' list for that sampling method with the different "sampSurf" objects corresponding to the row names in the respective sublist. Thus there are four sampling methods (top names = 'hps','chs','cmc','cps'), each with four sets of "sampSurf" simulations named as shown, for a total of 16 "sampSurf" objects in the entire list. Please see the example below for more information.

In general, `hfs` can be ragged; that is, the sublists do not need to be all the same length. Therefore some sampling methods may have fewer simulation results than others.

Value

A list is returned invisibly with. . .

`df`	The data frame that was used to create the figure.
`plt`	The lattice (class "trellis") plot object for the figure.

Note

Gove (2017) provides a graphical example created from this function corresponding to the example given below. That reference also shows how these plots can be extended for use with the results of a MODWT wavelet analysis. The main idea behind the variance relationship dates back to the seminal study of *Smith (1938).*

Author(s)

Jeffrey H. Gove

References

Gove, J. H. 2017. Some refinements on the comparison of areal sampling methods via simulation. *Forests*, (Submitted).

Smith, H. F. 1938. An empirical law describing heterogeneity in the yields of agricultural crops. *Journal of Agricultural Science*, **28**:1–23.

See Also

"sampSurf"

Examples

```
#
#  Using smithPlot entails generating several simulations for each sampling
#  method and thus some rather lengthy code--too much so for
#  reproduction here. However, assume that we have several sampSurf
#  simulation objects as follows (see example in Details above)...
#
#  horizontal point sampling:          sshps.3, sshps.5, sshps.7, sshps.9
#  critical height sampling:           sschs.3, sschs.5, sschs.7, sschs.9
#  HPS /w crude Monte Carlo sampling: sscmc.3, sscmc.5, sscmc.7, sscmc.9
#  all with metric bafs 3, 5, 7 and 9.
#
#  Also we have four sampSurf circular plot sampling objects with plot
#  radii 5, 6, 7, and 9m...
#
#  sscps.5, sscps.6, sscps.7, sscps.9
#
#  Then we can form the input lists required by smithPlot as...
#
## Not run:
hps = list(HPSbaf3 = sshps.3, HPSbaf5 = sshps.5, HPSbaf7 = sshps.7, HPSbaf9 = sshps.9)
chs = list(CHSbaf3 = sschs.3, CHSbaf5 = sschs.5, CHSbaf7 = sschs.7, CHSbaf9 = sschs.9)
cmc = list(CMCbaf3 = sscmc.3, CMCbaf5 = sscmc.5, CMCbaf7 = sscmc.7, CMCbaf9 = sscmc.9)
cps = list(CPSrad5 = sscps.5, CPSrad6 = sscps.6, CPSrad7 = sscps.7, CPSrad9 = sscps.9)
hfs = list(hps = hps, chs = chs, cmc = cmc, cps = cps)
#
#  Finally, we create the Smith plot using...
#
res = smithPlot(hfs)
names(res)  #show the result object names

## End(Not run)
```

---

spCircle *Create a Circular Spatial Object*

Description

This function simply creates a circular polygon object that is of class "SpatialPolygons". It also creates a "SpatialPoints" object holding the center point of the circle.

Usage

```
spCircle(radius,
         spUnits = CRS(projargs = as.character(NA)),
         centerPoint = c(x = 0, y = 0),
         nptsPerimeter = 100,
         spID = paste("circle", .StemEnv$randomID(), sep = ":"),
         ...)
```

Arguments

`radius`	The radius of the circle in the appropriate units.
`spUnits`	Object of class `"CRS"`: A legal proj.4 coordinate system; the default is to have user-defined coordinates.
`centerPoint`	The circle's center location in the appropriate units. This is a vector of length two with names "x" and "y".
`nptsPerimeter`	The number of points forming the perimeter of the polygon.
`spID`	A unique identifier that will be used in displaying the spatial polygon.
`...`	Not used currently.

Details

The polygon created for the circle will always have the individual internal "`Polygon`" object named '`pgCircle`', while the internal "`Polygons`" object will be named '`pgsCircle`'. These may need to be renamed in the returned object to make more sense. See the **sp** package for more details.

Value

A list with the following components...

`spCircle`	The "`SpatialPolygons`" polygon object.
`location`	The "`SpatialPoints`" point object.

Author(s)

Jeffrey H. Gove

See Also

The `standingTree` constructor.

Examples

```
dbh = 20
sp.dbh = spCircle(dbh/2, centerPoint=c(x=30,y=80), spID='tree.1')
plot(sp.dbh$spCircle)
plot(sp.dbh$location, add=TRUE, pch=3)
```

standingTree — *Generate Objects of Class "standingTree"*

Description

This generic function has methods based on the signature formal argument `object`. It is used as a constructor function for objects that are of class "`standingTree`". There are two methods that should be used in preference to `new` to insure a valid object.

Usage

```
standingTree(object, ...)
```

Arguments

`object`	This is the signature formal argument, see the methods for more details.
`...`	Formal arguments that are different for each method, see those for details.

Details

`standingTree` is defined completely with respect to the signature of its methods `standingTree-methods`. Methods are available for `data.frame`, and `missing` signatures. Other methods can obviously be added for other signatures as necessary.

Value

A valid object of class "`standingTree`."

Author(s)

Jeffrey H. Gove

References

“The Stem Class” vignette in this package.

See Also

`standingTree-methods`

Examples

```
#
#create a standingTree object and show it...
#
st = standingTree(dbh=20, solidType=3)
summary(st)
plot(st)
```

`standingTree-class` *Class* `"standingTree"`*: Representation of Standing Trees*

Description

A subclass of virtual class “`Stem`” that can be used to represent standing trees. The class provides for creation and graphical display of “standingTree” objects.

Detail examples and information concerning this class is found in “The Stem Class” vignette, which should be consulted for more details on creating and using objects from this class.

Objects from the Class

Objects can be created by calls of the form `new("standingTree",...)`. However, this is *not* recommended. The object has a number of required slots that can be somewhat complex to calculate. Therefore, a constructor function `standingTree` has been provided that will make the desired object with the correct values for the slots. It is strongly recommended that one use this generic's methods to create the tree objects.

Slots

Please note that all diameters below are presumed to be in the *same* units as height, i.e., meters for “metric”, and feet for “English” `units`.

In addition to the slots provided by the virtual superclass “`Stem`”, the following slots are represented…

`buttDiam`**:** Object of class `"numeric"`: The diameter at the butt (e.g., stump) of the tree in the same units a height.

`topDiam`**:** Object of class `"numeric"`: The diameter at the tip of the tree in the same units a height.

`height`**:** Object of class `"numeric"`: The total height of the tree in feet or meters.

`dbh`**:** Object of class `"numeric"`: The diameter at breast height of the tree.

`ba`**:** Object of class `"numeric"`: The tree basal area.

`solidType`**:** Object of class `"numericNULL"`: Object of class `"numericNULL"`: The form parameter in the simple taper and volume equation presented in Van Deusen (1990) and Gove and Van Deusen (2011).

`treeVol`**:** Object of class `"numeric"`: The tree's volume.

`surfaceArea`**:** Object of class `"numeric"`: The tree's surface area.

`biomass`**:** Object of class `"numeric"`: The tree's stem biomass. This will be `NA` if no conversion was specified at object creation.

`carbon`**:** Object of class `"numeric"`: The tree's stem carbon content. This will be `NA` if no conversion was specified at object creation.

`conversions`**:** Object of class `"numeric"`: A vector with names `c('volumeToWeight','weightToCarbon')` specifying the conversion factors for woody biomass and carbon content.

`taper`**:** Object of class `"data.frame"`: The tree's taper, either as specified from measurements, or as generated via `standingTree`.

`profile`**:** Object of class `"data.frame"`: The tree profile as generated from the taper. The tree is assumed to be oriented standing North, and the pith of the base at the origin.

`transTree`**:** Object of class `"matrix"`: Same as `profile`, but translated as desired.

`spTree`**:** Object of class `"SpatialPolygons"`: A `SpatialPolygons` representation of `transTree` (i.e., the full standing tree) via the `sp` package.

`spDBH`**:** Object of class `"SpatialPolygons"`: A `SpatialPolygons` representation of the tree dbh.

Extends

Class `"Stem"`, directly.

Methods

bbox `signature(obj = "standingTree")`: Returns the bounding box for the object.

perimeter `signature(object = "standingTree")`: Extract the perimeter as a `"SpatialPolygons"` object for possible plotting.

plot `signature(x = "standingTree", y = "missing")`: Plots the object.

summary `signature(object = "standingTree")`: A summary of the object.

Author(s)

Jeffrey H. Gove

References

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

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

"The Stem Class" vignette in this package.

See Also

`"Stem"`, `"downLog"`, `"StemContainer"`

Examples

```
showClass("standingTree")
```

`standingTree-methods` *Methods for* `standingTree` *Object Construction in Package* **sampSurf**

Description

There are currently two available methods for the generic `standingTree`. These methods generate objects of class "`standingTree`" that are valid objects. This is the preferred method for generating such objects, rather then using `new`.

Methods

`signature(object = "missing")` This constructor will be used when the signature argument is "missing." When `object` is `missing`, the taper data is generated from the internal taper function between the `buttDiam` (derived internally from taper data) and `topDiam` diameters for the tree, in `nSegs` sections. The taper function used is documented in *"The Stem Class"* vignette and references for "`standingTree`." The following arguments are part of the function call; all arguments with the same names as class slots are also defined in the class definition (and may be stored differently than the arguments).

usage...

```
standingTree(
             dbh = 20,                    #cm
             topDiam = 0,                 #cm
             height = 15,                 #meters
             nSegs = 20,
          solidType = 3,          #must have some taper model for butt diam
             treeVol = NULL,
             surfaceArea = NULL,
             biomass = NA,
             vol2wgt = NA,
             carbon = NA,
             wgt2carbon = NA,
             centerOffset = c(x=0, y=0),   #tree base-pith center offset
             species = '',
             treeID = paste('tree',.StemEnv$randomID(),sep=':'),
             description = NULL,
             userExtra = NULL,
             units = 'metric',
             spUnits = CRS(projargs=as.character(NA)),
             runQuiet = TRUE,
             ... )
```

- `dbh`: The breast-height diameter. For object creation, units are in either *inches* or *cm*. Internally, within the object, they are stored in the same units as height: *feet* or *meters*, depending on the value for `units`.
- `topDiam`: The tree diameter at the tip with same units as `dbh`.

- `height`: The tree height in meters or feet, depending on `units`.
- `nSegs`: The number of height segements to be generated from the taper function for the "missing" signature. Note that there will be `nSegs+1` diameter measurements for the tree taper.
- `solidType`: The type of solid for the default taper equation; the range is from 1 to 10, with 1 being a neiloid, 2 a cone and 3+ a paraboloid. When the `object` argument is missing it defaults to a value of 3.
- `treeVol`: The tree volume if precomputed, otherwise, if `NULL`, the tree volume will be computed from the taper volume equation or Smalian's formula if `solidType=NULL`.
- `surfaceArea`: The tree surface area if precomputed, otherwise, if `NULL`, the tree surface area will be computed from the taper equation or spline approximation.
- `biomass`: The tree *stem* woody biomass if precomputed, otherwise, if `NA`, the tree biomass will be computed from the volume and `vol2wgt` conversion.
- `vol2wgt`: The volume to weight conversion factor. If `NA` and `biomass` is passed, then it will be computed.
- `carbon`: The tree carbon content if precomputed, otherwise, if `NA`, the carbon content will be computed from the biomass and `wgt2carbon` conversion.
- `wgt2carbon`: The weight to carbon conversion factor. If `NA` and `carbon` is passed, then it will be computed.
- `centerOffset`: The tree's center position (pith at dbh) that will be used for the `location` slot. This is a vector of length two with names "x" and "y"; note that it can be length three with a "z" coordinate, but it is not used anywhere currently.
- `species`: Some species identifier as a character string.
- `treeID`: Each tree should have its own *unique* identifier that is used in constructing the `Polygons` object for the perimeter. This becomes very important when combining individual trees into a population or collection via the container class "`standingTrees`." If nothing is supplied, a random ID is generated.
- `description`: A character vector description of the tree.
- `userExtra`: Anything else that one wants to carry along with the tree.
- `units`: Either "English" or "metric". These must be conformable with the projection in `spUnits`.
- `spUnits`: A valid `CRS` object specifying the Coordinate Reference System. This defaults to `NA`, which means you want to use your own user-defined system, say for a sample plot where the tree has been located in the field.
- `runQuiet`: `TRUE`: no feedback during object creation; `FALSE`: prints some information along the way.
- ... : Other arguments to be passed along.

`signature(object = "data.frame")` When `object` is a "data.frame," then it is assumed that the data frame contains the taper data in the form of diameters and heights (as columns with labels "diameter" and "height", respectively), with diameters in the *same* units as height. All arguments except those listed below are the same as in the previous constructor…

usage…

```
standingTree(object
            solidType = NULL,            #defaults to null for passed taper
             treeVol = NULL,
             surfaceArea = NULL,
             biomass = NA,
             vol2wgt = NA,
             carbon = NA,
             wgt2carbon = NA,
             centerOffset = c(x=0, y=0),   #tree base-pith center offset
             species = '',
             treeID = paste('tree',.StemEnv$randomID(),sep=':'),
             description = NULL,
             userExtra = NULL,
             units = 'metric',
             spUnits = CRS(projargs=as.character(NA)),
             runQuiet = TRUE,
             ...)
```

- `object`: A data frame (see note below).
- `solidType`: `NULL` is the default, because it is assumed that the taper data are from field measurements or have been generated from a different taper equation where this would not be a meaningful parameter. One can therefore have an educated guess about the genesis of taper data within an object by querying this slot in the completed object.
- `treeVol`: The tree volume if precomputed, otherwise, if `NULL`, the tree volume will be computed from the taper data frame using Smalian's method.

Note

It may not be immediately apparent how the taper data in the data frame is to be structured if you have this data available either from measurements, or from a different taper equation. The best way to check this out is to simply create a dummy "standingTree" object and then `show` or `print` it for a summary, which will show the first few records of the structure. If that is not enough, then you can look at the structure with the `@` operator applied to the object's `taper` slot. Please also see the vignette mentioned above. Remember, the diameters in the taper data frame are expected to be in the same units as length for a data frame.

Author(s)

Jeffrey H. Gove

See Also

The "`standingTree`" class and the `standingTree` generic.

Examples

```
#create a standingTree object and show it
st = standingTree(dbh=20, solidType=2.1)
summary(st)
```

standingTreeIZ-class *Class* "standingTreeIZ"

Description

This represents an incremental change from the virtual base "InclusionZone" class. It is meant to clearly facilitate the division between standing trees and down logs with regard to inclusion zones, and it is a virtual class like its direct parent.

Objects from the Class

A virtual Class: No objects may be created from it.

Slots

Only one slot has been added to this class, please see the definition of the "InclusionZone" class for other inherited slots...

`standingTree`**:** Object of class "standingTree": Holds an object of class "standingTree".

Extends

Class "InclusionZone", directly.

Methods

summary `signature(object = "standingTreeIZ")`: Print a summary for any subclass object

Author(s)

Jeffrey H. Gove

References

"*The InclusionZone Class*" vignette.

See Also

For subclasses, see: circularPlotIZ, horizontalLineIZ, and the standingTreeIZs container class.

Examples

```
showClass("standingTreeIZ")
```

`standingTreeIZs` *Generate Objects of Class "*`standingTreeIZs`*"*

Description

This generic function has one method based on the signature formal argument `object`. It is used as a constructor function for objects that are of class "`standingTreeIZs`". The associated method should be used in preference to `new` to insure a valid object.

Usage

```
standingTreeIZs(object, ...)
```

Arguments

`object`	Signature formal argument.
`...`	Formal arguments that are different for each method, see those for details.

Details

The methods that can be used to generate objects of class “`standingTreeIZs`” are detailed here: `standingTreeIZs-methods`.

Value

A valid object of class "`standingTreeIZs`."

Author(s)

Jeffrey H. Gove

References

“*The InclusionZone Class*” vignette.

See Also

For subclasses of `standingTreeIZ`, see: `circularPlotIZ`,

Examples

```
sts = standingTrees(4, units='English', heights=c(10,30))
sts.cp = lapply(sts@trees, 'circularPlotIZ', plotRadius=20)
izs.cp = standingTreeIZs(sts.cp)
bbox(izs.cp)
plot(izs.cp, axes=TRUE)
```

`standingTreeIZs-class` *Class* `"standingTreeIZs"`

Description

Like the "`standingTrees`" class, this a 'container' class that allows multiple objects of a specific subclass of `standingTreeIZ` to be grouped into a population or collection. Its specific intent it to hold a collection of "`InclusionZone`" objects for a given tree population, that will ultimately be used to generate a sampling surface for a given sampling method. Please see "*The InclusionZone Class*" vignette for more information.

Objects from the Class

Objects can be created by calls of the form `new("standingTreeIZs",...)`; however, as in the other classes within this package, constructors have been written to simplify the process. A `standingTreeIZs` constructor should therefore be used in preference to `new`. The constructor, `standingTreeIZs`, has different forms that will be useful with different classes of objects.

Slots

The slots are the same as for the "`izContainer`" virtual superclass; please see it for details beyond what is below.

`iZones`**:** Object of class `"list"`: This slot holds a list of objects that all correspond to the same subclass of the "`standingTreeIZ`" class.

Extends

Class `"izContainer"`, directly.

Methods

No methods defined with class "standingTreeIZs" in the signature.

Author(s)

Jeffrey H. Gove

References

"*The InclusionZone Class*" vignette.

See Also

See the subclasses of `standingTreeIZ` for valid object types that can be stored in this container.

Examples

```
showClass("standingTreeIZs")
```

`standingTreeIZs-methods` *Method for "standingTreeIZs" object construction in Package* **sampSurf**

Description

The following are the constructor methods for class "`standingTreeIZs`" in Package 'sampSurf'. Please see "*The InclusionZone Class*" vignette for more information and examples on the usage of these methods, as well as the `standingTreeIZs` generic and class `standingTreeIZs`.

Methods

`signature(object = "list")` This method will create a container object out of a list containing "`standingTreeIZ`" objects.

usage...

```
standingTreeIZs(object,
                description = '',
                ... )
```

- `object`: A list object containing the collection of inclusion zones for a given subclass of "`standingTreeIZ`". Note that the list must contain objects that are all of the same class, or sampling method.
- `description`: A description of the collection.
- ... : Other arguments to be passed along (not used currently).

`signature(object = "standingTrees")` Create a container object out of a "`standingTrees`" container object and an "`InclusionZone`" subclass specification.

usage...

```
standingTreeIZs(object,
                iZone,
                description = '',
                ... )
```

- `object`: A valid "`standingTrees`" container object.
- `iZone`: A legal "InclusionZone" class specification (constructor name) that is relevant to "standingTree" objects, as a character string.
- `description`: A description of the collection.
- ... : Other arguments to be passed along, e.g., to the `iZone` constructor.

standingTrees *Generate Objects of Class "standingTrees"*

Description

This generic function has methods based on the signature formal arguments `object` and `container`. It is used as a constructor function for objects that are of class "`standingTrees`". This method should be used in preference to `new` to insure a valid object.

Usage

```
standingTrees(object, container, ...)
```

Arguments

`object`	Signature formal argument to key on whether trees are passed or generated.
`container`	Relevant if the trees are to be contained within some physical boundary/area like a "`Tract`" object.
`...`	Formal arguments that are different for each method, see those for details.

Details

The methods that can be used to generate objects of class "`standingTrees`" are detailed here: `standingTrees-methods`. As mentioned, each has a different signature and supporting formal arguments allowing you to generate a collection in various ways. Please see the above link for more details.

Value

A valid object of class "`standingTrees`."

Author(s)

Jeffrey H. Gove

References

"The Stem Class" vignette in this package.

See Also

`standingTree`, and classes: "`standingTree`", "`StemContainer`"

Examples

```
showMethods("standingTrees")
strees = standingTrees(15, xlim=c(0,20), ylim=c(10,40), dbhs=c(10,25))
summary(strees)
plot(strees, axes=TRUE)
```

`standingTrees-class` *Class* `"standingTrees"`

Description

The "standingTrees" class is a simplified container class that can hold multiple objects of class "`standingTree`". Its specific purpose is to hold a population of standing trees that are either generated synthetically as part of a simulation, or a collection from field measurements. The constructor of the same name has several different forms corresponding to possible argument signatures.

Objects from the Class

Objects can be created by calls of the form `new("standingTrees",...)`; however, as in the other classes within this package, constructors have been written to simplify the process. The `standingTrees` constructor should therefore be used in preference to `new`.

Slots

Please see the virtual base class, "`StemContainer`", for additional slots definitions.

`trees`**:** Object of class `"list"`: This holds the collection of "`standingTree`" objects.

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 standing trees. Because the statistics and bounding box are tied to the collection, a caution is in order regarding changing in any way the objects within your R code. 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.

Extends

Class `"StemContainer"`, directly.

Methods

hist `signature(x = "standingTrees")`: Displays a histogram of different variables in the collection.

plot `signature(x = "standingTrees",y = "missing")`: Plot the collection.

summary `signature(object = "standingTrees")`: Same as `show` currently.

Author(s)

Jeffrey H. Gove

References

"The Stem Class" vignette in this package.

See Also

Classes: "standingTree", "Stem", "StemContainer"; and to construct objects, the standingTrees constructor, perhaps using sampleTrees.

Examples

```
showClass("standingTrees")
buff = matrix(c(0,100,0,100), nrow=2, byrow=TRUE,
              dimnames=list(c('x','y'),c('min','max')))
st = sampleTrees(10, dbh = c(10,25), sampleRect = buff)
sts = standingTrees(st)
summary(sts)
```

---

standingTrees-methods *Methods for* standingTrees *Object Construction*

Description

The following are the constructor methods for class "standingTrees" in Package **sampSurf**. Please see "*The Stem Class*" vignette for more information and examples on the usage of the methods.

Methods

`signature(object = "list", container = "missing")` Ultimately, all of the constructors end up calling this one to generate the object after they have done what they were meant to, and then converted all of the "standingTree" objects into a `list` structure. So if one wishes, one can simply create one's own list of "standingTree" objects and use this constructor.

usage. . .

```
standingTrees(object,
              description = '',
              ... )
```

- `object`: A `list` as discussed above.
- `description`: A character description of the collection.
- ... : Other arguments to be passed along (not used currently).

`signature(object = "data.frame", container = "missing")` This particular method allows one to pass a data frame in the form generated from sampleTrees in the `object` argument to construct an object of class "standingTrees". Note, however, that the data frame does not have to be created by `sampleTrees`, and it does not have to contain synthetic/simulated trees. As long as all of the columns are present that are generated by `sampleTrees` it will be used; for example, the data frame can be constructed from a field sample of trees.

usage. . .

```
standingTrees(object, units = 'metric', ... )
```

- `object`: A data frame as discussed above.

- `units`: The units of measurement.
- ... : Other arguments to be passed along to the `standingTree` constructor—these apply uniformly to all trees in the collection.

`signature(object = "numeric", container = "bufferedTract")` This method will take as its first argument (`object`) the number of trees to be synthetically generated. The centers of the trees at breast height will all lie within the interior region of the "`bufferedTract`" object passed in the second (`container`) argument.

usage. . .

```
standingTrees(object, container, units = 'metric', ... )
```

- `object`: A numeric (truncated to integer) object specifying the number of trees to generate as discussed above.
- `container`: A "`bufferedTract`" object specifying the internal portion of the tract within which the tree centers will be generated.
- `units`: The units of measurement.
- ... : Other arguments to be passed along to the last constructor described below.

`signature(object = "numeric", container = "missing")` This constructor looks like it just generates trees randomly. But in reality, note the arguments below. It takes limits of a rectangular region as an alternative to specifying a bounding box matrix or a "bufferedTract" object as in two of the other constructors. Essentially, these limits get converted to a matrix bounding box and the appropriate constructor is used (see below).

usage. . .

```
standingTrees(object,
              xlim = c(0,100),
              ylim = c(0,50),
              units = 'metric',
              ...)
```

- `object`: See explanation in the following constructor method.
- `units`: The units of measurement.
- `xlim`: Limits for the bounding rectangle in `x`.
- `ylim`: Limits for the bounding rectangle in `y`.
- ... : Other arguments to be passed along as described in the last constructor method.

`signature(object = "numeric", container = "matrix")` Like the previous constructor, the trees will be generated so that their centers lie within a rectangular area specified by the `matrix` object in the second argument. Note that ultimately, the synthetic population of trees is generated by a call to `sampleTrees`.

usage. . .

```
standingTrees(object,
              container,
              units = 'metric',
             dbhs = c(8, 40),                 #cm for object construction!
              topDiams = c(0.4, 0.9),           #proportion multiplier
              heights = c(5,15),
              solidTypes = c(1,10),
              species = .StemEnv$species,
              ...)
```

- `object`: A numeric (truncated to integer) object specifying the number of trees to generate as discussed above.
- `container`: A `matrix` object specifically in the form of a `bbox`, bounding box object (e.g., see the `sp` package). It must be a `2x2` matrix with row names `c("x","y")` and the column names must be `c("min","max")`. The object specifies the internal portion of the tract within which the log centers will be generated.
- `units`: The units of measurement.
- `dbhs`: A length-two vector specifying the *range* of breast height diameters from which to uniformly draw the sample. Note that this range is assumed to be specified in cm for metric and inches for English units.
- `topDiams`: A length-two vector specifying the *range* of top (at the tip) diameters in the form of a *proportion* of the `dbh` diameters, from which to uniformly draw the sample. See also `sampleTrees`.
- `heights`: A length-two vector specifying the *range* of tree heights in feet (English) or meters (metric) from which to uniformly draw the sample.
- `solidTypes`: A length-two vector specifying the range in the tree shape parameter for the default taper and volume equations; where: 1-2 is a neiloid, 2 is a cone, and >2 is a paraboloid.
- `species`: A character vector of possible species from which to draw the sample uniformly. This can be any legal character string, and so can include codes, names, Latin names, etc.
- ... : Other arguments to be passed along to `sampleTrees`. Note specifically that one can control the random number stream by specifying a seed to be passed to `sampleTrees` with its `startSeed` argument. Additionally, this list can also contain arguments to be ultimately passed on to the `standingTree` constructor to be applied to each individual tree in the collection. For example, one can specify the number of segments desired in the taper function (`nSegs`) in this way.

`standUpIZ` *Generate Objects of Class "*`standUpIZ`*"*

Description

This is the generic definition for generating objects of class "standUpIZ." There is only one method corresponding to this generic: `standUpIZ-methods`.

Usage

```
standUpIZ(downLog, plotRadius, ...)
```

Arguments

`downLog`	Signature object of class "`downLog`".
`plotRadius`	Signature object for plot radius.
`...`	See methods.

Details

Since only one method exists for this generic, its signature arguments coincide with the above. Please see `standUpIZ-methods` for more details.

Value

A valid object of class "`standUpIZ`."

Author(s)

Jeffrey H. Gove

References

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

See Also

Class "`standUpIZ`", and `standUpIZ-methods`.

Examples

```
dl = downLog(buttDiam=40, solidType=4, logAngle=4*pi/3, logLen=6)
iz.su = standUpIZ(dl, 5)
summary(iz.su)
plot(iz.su, axes=TRUE, showPlotCenter=TRUE, cex=2, showNeedle=TRUE)
```

`standUpIZ-class` *Class "standUpIZ"*

Description

This class holds the inclusion zone definition for the “stand-up” method (Gove and Van Deusen, 2011) for sampling down coarse woody debris.

Objects from the Class

Objects can be created by calls of the form `new("standUpIZ",...)`. As usual in this class, however, this is not recommended because the objects are fairly complex. Instead, one can use the object constructor `standUpIZ` to create new objects.

Slots

Most of the slots are described in the superclasses (see below), please see their help pages for more information. Essentially, this class adds only one slot to the "downLogIZ" class specification.

`circularPlot`**:** Object of class `"circularPlot"`: An object from the "`ArealSampling`" subclass "`circularPlot`". Please see the help for this class for more information on the slots associated with circular plot objects.

Extends

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

Methods

izGrid `signature(izObject = "standUpIZ", tract = "Tract")`: "InclusionZoneGrid" generic constructor

perimeter `signature(object = "standUpIZ")`: Return the object perimeter

plot `signature(x = "standUpIZ", y = "missing")`: Plot the object

summary `signature(object = "standUpIZ")`: Object summary

Author(s)

Jeffrey H. Gove

References

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

See Also

`chainSawIZ`, `sausageIZ`, `fullChainSawIZ` and the `downLogIZs` container class.

Examples

```
showClass("standUpIZ")
```

standUpIZ-methods — *Method for "standUpIZ" object construction in Package* **sampSurf**

Description

This is the one method for generic function standUpIZ in Package 'sampSurf' that allows for creation of objects of class "standUpIZ."

Methods

`signature(downLog = "downLog", plotRadius = "numeric")`

usage...

```
standUpIZ(downLog,
          plotRadius,
          description = 'inclusion zone for "standup" method',
          spID = unlist(strsplit(tempfile('cp:',''),'\/'))[2],
          spUnits = CRS(projargs=as.character(NA)),
          ... )
```

- `downLog`: An object of class "downLog" which the inclusion zone is to be determined for under the stand-up method.
- `plotRadius`: The radius of the circular fixed-area plot in the correct units: feet for "English" and meters for "metric."
- `description`: A character vector description of the object.
- `spID`: A unique identifier that will be used in displaying the spatial polygon for the circular plot component of the object.
- `spUnits`: A valid CRS object specifying the Coordinate Reference System. This defaults to NA, which means you want to use your own user-defined system, say for a sample plot located in the field.
- `dots`: Arguments to be passed on.

Stem-class — *Class "Stem" for building tree and log like subclasses*

Description

This class is virtual. It provides the minimal structure to build subclasses that can be tailored to the desired application. Notably, the "downLog" and "standingTree" subclasses are already implemented. It provides the basis for graphical display via the sp package.

Objects from the Class

A virtual Class: No objects may be created from it.

Slots

`species`**:** Object of class `"character"`: Some form of identifier to species or group level.

`units`**:** Object of class `"character"`: The units of measurement system used, must be either "English" or "metric".

`location`**:** Object of class `"SpatialPoints"`: The x,y location of the object, can include z (elevation) but this is not used currently.

`spUnits`**:** Object of class `"CRS"`: A legal proj.4 coordinate system; the default is to have user-defined coordinates.

`description`**:** Object of class `"character"`: Some helpful comment about the object.

`userExtra`**:** Object of class `"ANY"`: This could be anyting the user wants to store with the object; specifically, it might hold information that is difficult to pass incorporate in any way given the constraints on "Stem" objects.

Methods

plot `signature(x = "Stem", y = "missing")`: Graphical display of the object. Minimal display is the object `location`.

show `signature(object = "Stem")`: Show main components or summary of an object.

summary `signature(object = "Stem")`: Concise summary of the object.

Author(s)

Jeffrey H. Gove

See Also

Subclasses "`downLog`" for downed logs and "`standingTree`" for standing trees.

Examples

```
showClass("Stem")
```

`StemContainer-class` *Class* `"StemContainer"` *for collections of tree and log objects*

Description

This is a virtual class that provides the minimal shared structure for the actual usable subclasses such as “standingTrees” and “downLogs”. In general, the idea for container objects is similar to what is found in some other languages such as `C++` or `Java`. The container object just hold a collection of objects of the appropriate subclass, and has methods that work on this collection as a whole, like plotting the collection. It is a simplified view of traditional container classes in that it does not currently allow for deletion or addition of objects.

Objects from the Class

A virtual Class: No objects may be created from it.

Slots

`units`**:** Object of class `"character"`: The units of measure used, must be either "English" or "metric".

`bbox`**:** Object of class `"matrix"`: The overall bounding box extents for the collection.

`stats`**:** Object of class `"data.frame"`: A summary of simple statistics on the collection; e.g., the mean and variance for volume etc.

`description`**:** Object of class `"character"`: A helpful comment about the collection.

Methods

bbox `signature(obj = "StemContainer")`: Calculates the overall bounding box matrix.

perimeter `signature(object = "StemContainer")`: Extract the perimeter of the collection corresponding to the bounding box as a "`SpatialPolygons`" object for possible plotting.

plot `signature(x = "StemContainer",y = "missing")`: Plots the collection.

show `signature(object = "StemContainer")`: Show component summary for the base slots in the collection.

summary `signature(object = "StemContainer")`: Same as show.

Author(s)

Jeffrey H. Gove

References

"The Stem Class" vignette in this package.

See Also

Subclasses: `downLogs` and `standingTrees`.

Examples

```
showClass("StemContainer")
```

`taperInterpolate` *Interpolate Diameters or Lengths in a "`Stem`" Object*

Description

Taper information is stored in a data frame within the `taper` slot of a "`downLog`" or "`standingTree`" object. This function lets one interpolate diameters at given lengths/heights, or vice versa from the taper data. This is done using either the built-in taper function or splines, depending on the value of the `solidType` slot in the object. Please see the methods in `taperInterpolate-methods` for more details.

Usage

```
taperInterpolate(object, ...)
```

Arguments

`object`	See methods for the possibilites of this signature argument.
`...`	Just gobbled for now.

Details

Please note that the routine will throw an error if the points you select for interpolation are not found in the stem.

If `solidType = NULL` then the user has supplied their own taper data, and splines are used for the interpolation. Otherwise, the built-in taper function is used since the taper data were generated from it.

Value

A numeric vector with the interpolated diameters or lengths/heights.

Author(s)

Jeffrey H. Gove

See Also

"`downLog`", "`standingTree`", *The Stem Class* vignette.

Examples

```
dl = downLog(buttDiam=12, topDiam=2, logLen=10, units="English")
dl@taper
taperInterpolate(dl, pts=c(2.6, 4.1, 7.2))
```

taperInterpolate-methods *Methods for Function* `taperInterpolate` *in Package* **sampSurf**

Description

There are three methods for the `taperInterpolate` generic function keyed off the one `object` signature. Each of them share the same argument list, but with one argument having slightly different meanings between the methods. The method associated with `list` objects should *not* be used unless you know what you are doing. Please use the wrappers in the other two methods below, which set the correct list structure up, and call the `list` version to do all the calculations.

Methods

`signature(object = "downLog")`

usage...

```
taperInterpolate(object,
                 whichSense = c('diameter', 'length'),
                 pts = NULL,
                 ...)
```

- `object`: An object of class “downLog”.
- `whichSense`: Either "diameter" to interpolate diameters, or "length" to interpolate lengths.
- `pts`: If `whichSense = "diameter"` then this is a vector of lengths at which to interpolate the diameters. Alternatively, if `whichSense = "length"` then this is a vector of diameters at which to interpolate the lengths.

`signature(object = "list")` Please do not use this method, use the other two which call it correctly...

usage...

```
taperInterpolate(object,
                 whichSense = c('diameter', 'length', 'height'),
                 pts = NULL,
                 ...)
```

- `object`: An object of class “list”.
- `whichSense`: Either “diameter” to interpolate diameters, “length” to interpolate lengths for “downLog” objects, or “height” to interpolate heights for “standingTree” objects.
- `pts`: If `whichSense = "diameter"` then this is a vector of lengths/heights at which to interpolate the diameters. Alternatively, if `whichSense = "length"` (or "height") then this is a vector of diameters at which to interpolate the lengths/heights.

`signature(object = "standingTree")`

usage...

```
taperInterpolate(object,
                 whichSense = c('diameter', 'height'),
                 pts = NULL,
                 ...)
```

- `object`: An object of class “standingTree”.
- `whichSense`: Either "diameter" to interpolate diameters, or "height" to interpolate heights.
- `pts`: If `whichSense = "diameter"` then this is a vector of heights at which to interpolate the diameters. Alternatively, if `whichSense = "height"` then this is a vector of diameters at which to interpolate the heights.

`Tract` *Generate Objects of Class "*`Tract`*"*

Description

This generic function has methods based on the signature formal argument `object`. It is used as a constructor function for objects that are of class "`Tract`". This method should be used in preference to `new` to insure a valid object.

Usage

```
Tract(object, ...)
```

Arguments

`object`	This is the signature formal argument, see the methods for more details.
`...`	Formal arguments that are different for each method, see those for details.

Details

There is no default generic function for `Tract` per se. It is defined completely with respect to the signature of its methods `Tract-methods`. Methods are available for several useful signatures, such as objects of class "`RasterLayer`", allowing for a large variety of ways to generate objects of this class. Other methods can obviously be added for other signatures as necessary.

Value

A valid object of class "`Tract`."

Author(s)

Jeffrey H. Gove

See Also

`Tract-methods`

Examples

```
tract = Tract(c(x=20,y=20), cellSize=0.25)    #extents constructor
tr = Tract(,0.5, c(x=20,y=20),c(0.25,0.25),units='metric') #missing constructor
tr2 = Tract(cellSize=0.5, cellDims=c(x=20,y=20), cellCenter=c(0.25,0.25),
            units='metric') #same constructor as the previous
tr.ext = Tract(c(x=10,y=15), cellSize=0.5) #maximal extent constructor
#bounding box constructor...
bb = bbox(tr.ext)
bb[,1] = c(5,4)
tr3 = Tract(bb, 0.5)
#RasterLayer constructor...
ex = extent(tr3)
rl = raster(ex, nrow=10, ncols=10, crs=NA)
## Not run:
tr4 = Tract(rl)  #error: non-square grid cells

## End(Not run)
rl = raster(ex, nrow=22, ncol=10, crs=NA)
tr4 = Tract(rl)  #okay, square cells
```

Tract-class *Class "Tract"*

Description

The "Tract" class is the base class for creating objects corresponding to raster grids for use in samplng surface simulation. The class is a subclass of "`RasterLayer`" and so incorporates all of the slots from that class as well as the ones defined below.

Objects from the Class

Objects can be created by calls of the form `new("Tract",...)`; however, due to the complexity of the class, this is not recommended. Instead, one can use on of the class constructor methods of the same name, `Tract`, to create and object. There are several of these available, giving a variety of ways to create a raster grid for use in sampling surface simulation.

There is currently one subclass, "`bufferedTract`", which allows an internal buffer.

Slots

In addition to the slots found in the "`RasterLayer`" class, the following slots are defined. . .

`description`**:** Object of class `"character"`: A character vector describing the tract if desired.

`units`**:** Object of class `"character"`: The units of measure used, must be either "English" or "metric".

`area`**:** Object of class `"numeric"`: The area of the tract in appropriate units.

Extends

Class "`RasterLayer`", directly. Class "`Raster`", by class "RasterLayer", distance 2. Class "`BasicRaster`", by class "RasterLayer", distance 3.

Methods

Examples of methods include, but are not limited to the following...

bbox `signature(obj = "Tract")`: Return object bounding box

bufferedTract `signature(bufferWidth = "numeric", tract = "Tract")`: Object constructor

heapIZ `signature(izgObject = "InclusionZoneGrid", tract = "Tract")`: Heap into a sampling surface

izGrid `signature(izObject = "fullChainSawIZ", tract = "Tract")`: Full chainsaw InclusionZoneGrid

izGrid `signature(izObject = "chainSawIZ", tract = "Tract")`: “InclusionZoneGrid” generic constructor

izGrid `signature(izObject = "matrix", tract = "Tract")`: ...

izGrid `signature(izObject = "sausageIZ", tract = "Tract")`: ...

izGrid `signature(izObject = "standUpIZ", tract = "Tract")`: ...

perimeter `signature(object = "Tract")`: Returns object perimeter

plot `signature(x = "Tract", y = "missing")`: Plot the object

sampSurf `signature(object = "downLogIZs", tract = "Tract")`: Generate a sampling surface

sampSurf `signature(object = "numeric", tract = "Tract")`: ...

show `signature(object = "Tract")`: Show/print the object

summary `signature(object = "Tract")`: Object summary

Author(s)

Jeffrey H. Gove

See Also

`bufferedTract`, `mirageTract`

Examples

```
showClass("Tract")
```

`Tract-methods` *Methods for "Tract" Object Construction in package 'sampSurf'*

Description

There are several methods available for the `Tract` generic function, which will generate valid objects of class "`Tract`". These constructors are the preferred method for generating such objects instead of using `new`. Other constructors could be added by extention to these method as required.

Methods

`signature(object = "missing")` Several of the constructors listed below end up calling this one for final object construction, so it could be considered the base constructor. It is patterned after the construction of a "`GridTopology`" object in the '`sp`' package. Please note that it is best to either explicitly name the arguments when using this constructor, or to include an initial missing argument (see examples in the generic) to invoke it, since supplying an unnamed numeric scalar as the first argument will instead, invoke the second constructor below (with an error).

usage...

```
Tract(cellSize,
      cellDims = c(x=100, y=100),
      cellCenter = NULL,
      units = 'metric',
      data = 0,
      spUnits = CRS(projargs=as.character(NA)),
      description = 'object of class Tract',
      runQuiet = FALSE,
      ... )
```

- `cellSize`: The size of the individual grid cells in the appropriate units. Note particularly that the restriction has been made that grid cells must be square, so this is expecting a numeric scalar.
- `cellDims`: The dimensions in terms of number of cells in the x and y directions. The to match a desired extent in terms of the units of x and y, one must take the `cellSize` into account when determining these limits. For example, if the extent in the x and y directions are to be 10 units, and the `cellSize` is 0.5 units, then the specify this argument as `c(x=20,y=20)`.
- `cellCenter`: This is the center point for the grid cell at the origin of the tract. If the `cellSize` is 0.5 units and the minimum in x and y is (0,0), then this would be `c(x=0.25,y=0.25)`. Note that we can specify lower limts other than zero for the tract using this method.
- `units`: Either "English" or "metric". These must be conformable with the projection in `spUnits`.
- `data`: The default is to set all grid cells to zero. One can either specify a numeric scalar which will be replicated for all cells, or a numeric vector the same length as the number of grid cells; these will be assigned in sequence to the grid.

- spUnits: A valid CRS object specifying the Coordinate Reference System. This defaults to NA, which means you want to use your own user-defined system, say for a sample plot located in the field.
- description: A character vector description of the tract.
- runQuiet: TRUE: suppress any messages; FALSE: echo messages.
- ...: Other arguments to be passed along (not used currently).

signature(object = "numeric") In this second constructor, the object argument should be a numeric vector specifying the *maximum extents* of the tract. The tract minimum is always assumed to be at (0,0) for this method. If some other minima is desired, use one of the other constructors.

usage...

```
Tract(object = c(x=10, y=10),
      cellSize,
      units = 'metric',
      data = 0,
      spUnits = CRS(projargs=as.character(NA)),
      description = 'object of class Tract',
      runQuiet = FALSE,
      ... )
```

- object: A length two vector with names c("x","y") specifying the maximal extents of the tract in these coordinates.
- runQuiet: TRUE: suppress any messages; FALSE: echo messages.
- ...: Other arguments to be passed along.
- All other arguments as defined for the other constructors above.

signature(object = "matrix") This construtor allows creation of a “Tract” object using a bounding box specification.

usage...

```
Tract(object,
      cellSize,
      units = 'metric',
      data = 0,
      spUnits = CRS(projargs=as.character(NA)),
      description = 'object of class Tract',
      runQuiet = FALSE,
      ... )
```

- object: A 2x2 matrix with rownames c("x","y") and column names c("min","max"). The matrix therefore specifies the bounding box extents in the plane. The origin can be anywhere.
- All other arguments as defined for the other constructors above.

signature(object = "RasterLayer") Finally this constructor takes an object of class “RasterLayer”.

usage...

```
Tract(object,
      units = 'metric',
      description = 'object of class Tract',
      runQuiet = FALSE,
      ... )
```

- `object`: A valid object of type "`RasterLayer`".
- All other arguments as defined for the other constructors above.

`transparentColorBase` *Set Transparancey in Base Graphics*

Description

Setting transparency in base graphics is not as easy as in `Lattice`, so here's a little functon to help.

Usage

```
transparentColorBase(color, alphaTrans = alphaTrans)
```

Arguments

`color`	The color, or a vector of colors from `colors()`.
`alphaTrans`	The alpha transparency value between [0,1] with 0 opaque and 1 fully transparent.

Details

As above.

Value

The rgb value(s), which can be passed to any base graphics routine to get transparency.

Author(s)

Jeffrey H. Gove

See Also

`col2rgb`, `rgb`

Examples

```
## Not run:
cols = transparentColorBase('red', alphaTrans=c(0.3,0.6,0.9))
symbols(c(1,1.5,2), c(1,1.5,2), circles=rep(1,3), bg=cols, xlim=c(0,4), ylim=c(0,4))

## End(Not run)
```

Index