Package 'SDraw'

July 3, 2020

Type Package

Title Spatially Balanced Samples of Spatial Objects

Version 2.1.13

Date 2020-07-03

Maintainer Trent McDonald <tmcdonald@west-inc.com>

Description Routines for drawing samples from spatial objects, focused on spatially
balanced algorithms. Draws Halton Iterative Partition (HIP)
(Robertson et al., 2018; <doi:10.1007/s10651-018-0406-6>),
Balanced Acceptance Samples (BAS) (Robertson et al., 2013; <doi:10.1111/biom.12059>),
Generalized Random
Tessellation Stratified (GRTS) (Stevens and Olsen, 2004; <doi:10.1198/016214504000000250>),
Simple Systematic Samples (SSS) and
Simple Random Samples (SRS) from point, line, and polygon resources.
Frames are 'SpatialPoints', 'SpatialLines', or 'SpatialPolygons'
objects from package 'sp'.

License GNU General Public License

URL https://github.com/tmcd82070/SDraw/wiki/SDraw

BugReports https://github.com/tmcd82070/SDraw/issues

Imports spsurvey, utils, rgeos, graphics, methods, deldir, stats, covr

Depends sp, R (>= 3.5.0)

LazyData true

Suggests knitr, testthat, devtools, roxygen2, markdown

VignetteBuilder knitr

RoxygenNote 7.1.0

Encoding UTF-8

NeedsCompilation no

Author Trent McDonald [cre, aut],
Aidan McDonald [aut] (HIP sampling methods),
Michael Kleinsausser [ctb],
Stefan Emmons [ctb] (Auto testing and Travis)

Repository CRAN

Date/Publication 2020-07-03 16:20:09 UTC

R topics documented:

SDraw-package . . . 3
bas.line . . . 4
bas.point . . . 6
bas.polygon . . . 7
extended.gcd . . . 9
getJ . . . 10
grts.line . . . 11
grts.point . . . 13
grts.polygon . . . 14
halton . . . 16
halton.coefficients . . . 18
halton.frame . . . 18
halton.indices . . . 20
halton.indices.CRT . . . 22
halton.indices.vector . . . 24
halton.lattice . . . 26
halton.lattice.polygon . . . 28
HI.coast . . . 30
hip.lattice.polygon . . . 30
hip.plot.lattice . . . 32
hip.point . . . 32
hip.polygon . . . 34
lineLength . . . 36
maxU . . . 37
plotLattice . . . 38
plotSample . . . 39
polygonArea . . . 40
primes . . . 41
sdraw.SpatialLines . . . 42
srs.line . . . 44
srs.point . . . 45
srs.polygon . . . 47
sss.line . . . 48
sss.point . . . 50
sss.polygon . . . 52
voronoi.polygons . . . 55
WA . . . 56
WA.cities . . . 57
WY . . . 59

Index **60**

`SDraw-package` *Selection of spatially balanced samples.*

Description

SDraw provides a set of R functions that draw Halton-Lattice samples (HAL), Balanced-Acceptance-Samples (BAS), Generalized-Random-Tessellation-Stratified (GRTS) samples, Simple-Systematic-Samples (SSS), and Simple-Random-Samples (SRS). The types of input frames accepted are points (0-D, finite), lines (1-D, infinite), or polygons (2-D, infinite).

Package:	SDraw
Type:	Package
License:	GNU General Public License
Imports:	spsurvey,utils,rgdal,rgeos,graphics, methods,deldir,OpenStreetMap,stats
Depends:	sp

The work-horse functions are named `???.point`, `???.line`, and `???.polygon`, where '???' is either `hal`, `bas`, `grts`, `sss`, or `srs`. For simplicity, an S4 generic, `sdraw`, is provided to handle all combinations of sample and frame types (see `sdraw`).

Author(s)

Trent McDonald `tmcdonald@west-inc.com`. The GRTS routine comes from package `spsurvey`.

References

Manly, B. F. J. and Navarro-Alberto, J. A., editors, (2015), "Introduction to Ecological Sampling", CRC Press, Boca Raton, FL.

Robertson, B.L., J. A. Brown, T. L. McDonald, and P. Jaksons (2013) "BAS: Balanced Acceptance Sampling of Natural Resources", Biometrics, v69, p. 776-784.

Stevens, D. L., Jr. and A. R. Olsen (2004) "Spatially balanced sampling of natural resources." Journal of the American Statistical Association 99, 262-278.

See Also

`sdraw`, `bas.polygon`, `bas.line`, `bas.point`, `hip.polygon`, `hip.point`, `sss.line`, `sss.polygon`, `grts.polygon`, `grts.line`, `grts.point` documentation for package `sp`.

bas.line	*Draws a Balanced Acceptance Sample (BAS) from a linear resource (line).*

Description

Draws a BAS sample from a `SpatialLines*` object.

Usage

```
bas.line(x, n, balance = "1D", init.n.factor = 10)
```

Arguments

`x` A `SpatialLines` or `SpatialLinesDataFrame` object. This object must contain at least 1 line. If it contains more than 1 line, the BAS sample is drawn from the union of all lines.

`n` Sample size. Number of locations to draw from the set of all lines contained in `x`.

`balance` Option specifying how spatial balance is maintained. The options are "1D" or "2D".

Under "1D" all lines in `x` are stretched straight and laid end-to-end in the order they appear in `x` and a 1-dimensional BAS sample is taken from the amalgamated line. 1D sample locations on the amalgamated line are mapped back to two dimensional space for output and appear on the original lines. This method maintains 1D spatial balance, but not necessarily 2D balance. Spatially balanced samples in 1D may not look spatially balanced when plotted in 2 dimensions.

Under "2D" a systematic sample of points along the union of all lines in `x` is drawn first, and a 2-dimensional BAS sample of the points is drawn (see `init.n.factor` below and `bas.point`). This maintains 2D spatial balance of sample locations on the lines. Depending on `init.n.factor`, the "2D" balance option can take significantly longer to run than the "1D" option.

`init.n.factor` If `balance == "2D"`, this is a scalar controlling the number of points to place on the lines before drawing the 2D BAS sample. Number of points created on the line is `n*init.n.factor`, so this number can grow quickly. On average, this is the number of un-selected points between each selected point. See Details.

If one desires an underlying grid spaced *w* meters apart, set `init.n.factor` to *L/(w*n)*, where *L* is total length of all lines in `x` and *n* is sample size.

Details

If a "1D" sample is requested, spatial balance is maintained on the lines when laid end-to-end in the order they appear. Points far apart in 1 dimension may be close together in 2 dimensions, and vice versa. Thus the sample may not look spatially balanced on a 2D map. This is a true infinite sample in that any of an infinite number of points along the lines could be selected.

If a "2D" BAS sample is requested, spatial balance is maintained in 2 dimensions. Points are well balance on a 2D map. This is done by discretization of lines with a dense systematic sample of points (with random start) where density of the systematic points is controlled by `init.n.factor`. After discretization of the line, points are selected using `bas.point`. The BAS method for points places a small square (pixel) around each and samples the set of squares using the BAS method for polygons (see `bas.polygon`). The BAS method of polygons computes Halton points until `n` fall inside the squares surrounding discretization points. When a Halton point falls in a square, the square is selected and the sample location is the center of the square (which falls somewhere on the original lines).

Value

A `SpatialPointsDataFrame` containing locations in the BAS sample, in BAS order. Attributes of the sample points are:

- `sampleID`: A unique identifier for every sample point. This encodes the BAS order. If BAS order is lost, `return[ order( return$sampleID ),]` will resort the returned object (i.e., `return`) into BAS order.
- `geometryID`: The ID of the line in `x` on which each sample point falls. The ID of lines in `x` are `row.names(x)`.
- Any attributes of the original lines (in `x`).

Additional attributes of the output object, beyond those which make it a `SpatialPointsDataFrame`, are:

- `frame`: Name of the input sampling frame.
- `frame.type`: Type of resource in sampling frame. (i.e., "line").
- `sample.type`: Type of sample drawn. (i.e., "BAS").
- `balance`: The type of balance ("1d" or "2d").
- `random.start`: The random seed for the random-start 1D or 2D Halton sequence that produced the sample. If `balance=="1D"`, this is a single uniform random integer between 0 and `maxU`. If `balance=="2D"`, this is a vector of two uniform random integers between 0 and `maxU`.
- `bas.bbox`: If `balance=="2D"`, this is the bounding box surrounding `x` used to scale Halton points. The scaled Halton sequence is `bas.bbox[,"min"]+ t(halton(n,2,random.start))* rep(max( diff(t(bas.bbox))), 2)`. If `balance=="1D"`, this is a vector containing the 1D bounding box. Lower limit of the 1D bounding box is 0. Upper limit of the 1D box is the total length of lines in `x`. In this case, Halton points are scaled as `bas.bbox[,"min"]+ halton(n,1,random.start)* diff(bas.bbox)` which is equivalent to `halton(n,1, random.start)* bas.bbox[2]` because `bas.bbox[1]` is zero.

Author(s)

Trent McDonald

See Also

`bas.line`, `bas.polygon`, `sdraw`

Examples

```
#   Draw sample of Hawaii coastline
#   This takes approximately 60 seconds to run
samp <- bas.line( HI.coast, 50 )
plot(HI.coast)
points( samp, pch=16, col="red" )
```

`bas.point` *Draws a Balanced Acceptance Sample (BAS) from a discrete resource (points).*

Description

Draws a BAS sample from a `SpatialPoints*` object.

Usage

```
bas.point(x, n)
```

Arguments

`x`	A SpatialPoints or SpatialPointsDataFrame object. This object must contain at least 1 point.
`n`	Sample size. Number of points to select from the set of points contained in `x`.

Details

The BAS method for points computes the minimum distance between any two points in `x` and places a small square (pixel) around each. Size of the square around each point is d/sqrt(2) on a side, where d is the minimum distance between points. The BAS method for points then selects a BAS sample from the set of polygons (i.e., squares) surrounding each point (see `bas.polygon`). The BAS method of polygons selects Halton points until `n` points are located inside the squares surrounding the points. When a square contains a Halton point, the official sample location is the the original point (center of the square), not the Halton point.

Value

A `SpatialPointsDataFrame` containing locations in the BAS sample, in BAS order. Attributes of the sample points are:

- `sampleID`: A unique identifier for every sample point. This encodes the BAS order. If BAS order is lost, `return[ order( return$sampleID ),]` will resort the returned object (i.e., `return`) into BAS order.
- `geometryID`: The ID of the point in `x` that has been selected. The ID of points in `x` are `row.names(x)`.

- Any attributes of the original lines (in x).

Additional attributes of the output object, beyond those which make it a SpatialPointsDataFrame, are:

- frame: Name of the input sampling frame.
- frame.type: Type of resource in sampling frame. (i.e., "point").
- sample.type: Type of sample drawn. (i.e., "BAS").
- random.start: The random seed of the random-start Halton sequence that produced the sample. This is a vector of length 2 whose elements are random integers between 0 and maxU. This routine ensures that the point associated with this index falls inside a polygon of interest. i.e., that halton(1,2,random.start) scaled by a square bounding box (see attribute bas.bbox below) lies inside a polygon of x.

  Note that halton(1,2,random.start+i), for i > 0, is not guaranteed to fall inside a polygon of x when scaled by bas.bbox. The sample consists of the point associated with random.start and the next n-1 Halton points in sequence that fall inside a polygon of x.
- bas.bbox: The square bounding box surrounding x used to scale Halton points. A scaled Halton sequence of n points is bas.bbox[,"min"] + t(halton(n,2,random.start)) * rep(max(diff(t(bas.bbox))),2).

Author(s)

Trent McDonald

See Also

bas.polygon, bas.line, spsample

Examples

```
## Not run:
bas.point( WA.cities, 100)

## End(Not run)
```

---

bas.polygon *Draws a Balanced Acceptance Sample (BAS) from an area resource (polygons).*

Description

Draws a BAS sample from a SpatialPolygons* object

Usage

```
bas.polygon(x, n)
```

Arguments

x	A SpatialPolygons or SpatialPolygonsDataFrame object. This object must contain at least 1 polygon. If it contains more than 1 polygon, the BAS sample is drawn from the union of all.
n	Sample size. Number of locations to draw from the union of all polygons contained in x.

Details

A BAS sample is drawn from the union of all polygons in x by enclosing all polygons in a bounding square and selecting a randomized Halton sequence of points from the bounding square. Points falling outside all polygons are discarded until exactly n locations are selected inside the polygons.

The sampling frame for routine is infinite and contains all (infinitesimally small) points in the union of polygons in x.

Value

A SDrawSample object, which is a SpatialPointsDataFrame object containing points in the BAS sample, in BAS order. Attributes of the sample points are:

- sampleID: A unique identifier for every sample point. This encodes the BAS order. If BAS order is lost, return[order(return$sampleID),] will resort the returned object (i.e., return) into BAS order.
- geometryID: The ID of the polygon in x which each sample point falls. The ID of polygons in x are row.names(geometry(x)).
- Any attributes of the original polygons (in x).

Additional attributes of the output object, beyond those which make it a SpatialPointsDataFrame, are:

- frame: Name of the input sampling frame.
- frame.type: Type of resource in sampling frame. (i.e., "polygon").
- sample.type: Type of sample drawn. (i.e., "BAS").
- random.start: The random seed of the random-start Halton sequence that produced the sample. This is a vector of length 2 whose elements are random integers between 0 and maxU. This routine ensures that the point associated with this index falls inside a polygon of x. i.e., that halton(1,2,random.start) scaled by a square bounding box (see attribute bas.bbox below) lies inside a polygon of x.

  Note that halton(1,2,random.start+i), for i > 0, is not guaranteed to fall inside a polygon of x when scaled by bas.bbox. The sample consists of the point associated with random.start and the next n-1 Halton points in sequence that fall inside a polygon of x.
- bas.bbox: The square bounding box surrounding x used to scale Halton points. A scaled Halton sequence of n points is bas.bbox[,"min"] + t(halton(n,2,random.start)) * rep(max(diff(t(bas.bbox))),2).

Author(s)

Trent McDonald

References

Robertson, B.L., J. A. Brown, T. L. McDonald, and P. Jaksons (2013) "BAS: Balanced Acceptance Sampling of Natural Resources", Biometrics, v69, p. 776-784.

See Also

bas.line, bas.point, sdraw

Examples

```
#   Draw sample
WA_sample <- bas.polygon(WA, 100)

#   Plot
plot( WA )

# Plot first 100 sample locations
points( WA_sample[ WA_sample$siteID <= 100, ], pch=16 )

# Plot second 100 locations
points( WA_sample[ WA_sample$siteID >  100, ], pch=1 )
```

---

extended.gcd — *Extended Greatest Common Denominator (GCD) algorithm.*

Description

Implements the extended Euclidean algorithm which computes the greatest common divisor and solves Bezout's identity.

Usage

```
extended.gcd(a, b)
```

Arguments

a	A vector of integers
b	A vector of integers. `length(a)` must equal `length(b)`.

Details

This routine computes the element-wise gcd and coefficients s and t such that a*t + b*s = d. In other words, if `x = extended.gcd(a,b)`, then `x$a*x$t + x$b*x$s == x$gcd`

The Wikipedia page, from which this algorithm was stolen, has the following statement, 'The quotients of a and b by their greatest common divisor, which are output, may have an incorrect sign. This is easy to correct at the end of the computation, but has not been done here for simplifying the code.' I have absolutely no idea what that means, but include it as a warning. For purposes of `SDraw`, elements of a and b are always positive, and I have never observed "incorrect signs". But, there may be some pathological cases where "incorrect signs" occur, and the user should "correct" for this. This routine does check that the output gcd is positive, and corrects this and the signs of s and t if so.

Value

a data frame containing 5 columns; `a`, `t`, `b`, `s`, and `gcd`. Number of rows in output equals length of input `a`.

Author(s)

Trent McDonald

References

Code is based on the following Wikipedia pseudo-code: https://en.wikipedia.org/wiki/Extended_Euclidean_algorithm

Examples

```
x <- extended.gcd( c(16,27,27,46), c(9,16,9,240) )

#  Check
cbind(x$a*x$t + x$b*x$s, x$gcd)
```

`getJ` *getJ*

Description

Compute J, the depth of the Halton lattice, given a population size .

Usage

```
getJ(N, bases = c(2, 3))
```

Arguments

`N`	Population or frame size (a scalar)
`bases`	A vector of co-prime Halton bases of length D.

Details

This routine returns a largest Halton cycle (i.e., B) such that n <= B <= N. The first few Halton cycles are in Robertson et al. (2018) Web table 1.

Value

A vector of length 2 containing the exponents of bases that produce a Halton cycle of length B such that n <= B <= N.

Author(s)

Trent McDonald

Examples

```
getJ(62208, c(2,3))  # should equal c(8,5)
getJ(72, c(2,3))  # should equal c(3,2)
```

`grts.line` *Draw a equi-probable GRTS sample from a linear (line) resource.*

Description

Draws an equi-probable unstratified Generalized Random Tessellation Stratified (GRTS) sample from a `SpatialLines*` object

Usage

```
grts.line(x, n, over.n = 0)
```

Arguments

`x`	A `SpatialLines` or `SpatialLinesDataFrame` object.
`n`	Sample size. The number of sample points to draw from `x`
`over.n`	Over-sample size. The number of 'over-sample' points to draw from `x`. The actual number of points drawn from `x` is `n + over.n`.

Details

This is a wrapper for the `grts` function in package `spsurvey`. This simplifies calling `grts` when equi-probable samples are desired. It extends the allowable input frame types to `SpatialLines` objects (i.e., no attributes), rather than just `SpatialLinesDataFrame` objects. For more complicated designs (e.g., variable probability, stratification), call `grts` directly.

Value

A `SpatialPointsDataFrame` containing locations in the GRTS sample, in order they are to be visited. Attributes of the sample points (in the embedded data frame) are as follows:

- `sampleID`: A unique identifier for points in the sample. This encodes the GRTS ordering of the sample. The output object comes pre-sorted in GRTS order. If the sample becomes un-GRTS-ordered, resort by `sampleID` (i.e., `samp <-samp[order(samp$sampleID),]`).
- `pointType`: A string identifying regular sample points (`pointType=="Sample"`) and over-sample points (`pointType=="OverSample"`).
- `geometryID`: The ID of the line in `x` onto which sample points fall. The ID's of lines in `x` are `row.names(geometry(x))`.
- Any attributes of the original lines (in `x`) onto which sample points fall.

Author(s)

Trent McDonald

References

Stevens, D. L. and A. R. Olsen (1999). Spatially restricted surveys over time for aquatic resources. Journal of Agricultural, Biological, and Environmental Statistics 4 (4), 415-428.

Stevens, D. L. and A. R. Olsen (2004). Spatially balanced sampling of natural resources. Journal of the American Statistical Association 99, 262-278.

See Also

`grts.line`, `grts.polygon` , `spsample`

Examples

```
## Not run:
# Draw sample
## The following take > 5s
HI.sample <- grts.line(HI.coast,10,5)

# Plot
plot( HI.coast )

# Plot 'sample' locations
plot( HI.sample[ HI.sample$pointType == "Sample", ], pch=16, add=TRUE, col="red" )
```

```
# Plot 'over sample' locations
plot( HI.sample[ HI.sample$pointType == "OverSample", ], pch=1, add=TRUE, col="blue" )

## End(Not run)
```

`grts.point` *Draw a equi-probable GRTS sample from a discrete (point) resource.*

Description

Draws an equi-probable unstratified Generalized Random Tessellation Stratified (GRTS) sample from a `SpatialPoints*` object

Usage

```
grts.point(x, n, over.n = 0)
```

Arguments

`x`	A `SpatialPoints` or `SpatialPointsDataFrame` object.
`n`	Sample size. The number of sample points to draw from `x`
`over.n`	Over-sample size. The number of 'over-sample' points to draw from `x`. The actual number of points drawn from `x` is `n + over.n`.

Details

This is a wrapper for the `grts` function in package `spsurvey`. This simplifies calling `grts` when equi-probable samples are desired. It extends the valid input frame types to `SpatialPoints` objects (i.e., no attributes), rather than just `SpatialPointsDataFrame` objects. For more complicated designs (e.g., variable probability, stratification), call `grts` directly.

Value

A `SpatialPointsDataFrame` containing locations in the GRTS sample, in order they are to be visited. Attributes of the sample points (in the embedded data frame) are as follows:

- `sampleID`: Unique identifier for sample points. This encodes the GRTS ordering of the sample. The output object comes pre-sorted in GRTS order. If the sample becomes un-GRTS-ordered, resort by `sampleID` (i.e., `samp <-samp[order(samp$sampleID),]`).
- `pointType`: A string identifying regular sample points (`pointType=="Sample"`) and over-sample points (`pointType=="OverSample"`).
- `geometryID`: The ID of the point in `x` which was sampled. The ID of points in `x` are `row.names( geometry(x))`.
- Any attributes of the original points (in `x`).

Author(s)

Trent McDonald

References

Stevens, D. L. and A. R. Olsen (1999). Spatially restricted surveys over time for aquatic resources. Journal of Agricultural, Biological, and Environmental Statistics 4 (4), 415-428.

Stevens, D. L. and A. R. Olsen (2004). Spatially balanced sampling of natural resources. Journal of the American Statistical Association 99, 262-278.

See Also

grts.line, grts.polygon, hip.point, sdraw

Examples

```
#   Draw sample
WA.city.samp <- grts.point(WA.cities,100,50)

#   Plot
plot( WA.cities, pch=16, cex=.5 )

# Plot 'sample' locations
plot( WA.city.samp[ WA.city.samp$pointType == "Sample", ], pch=1, add=TRUE, col="red" )

# Plot 'over sample' locations
plot( WA.city.samp[ WA.city.samp$pointType == "OverSample", ], pch=2, add=TRUE, col="blue" )
```

---

grts.polygon *Draw a equi-probable GRTS sample from an area (polygon) resource.*

Description

Draws an equi-probable unstratified Generalized Random Tessellation Stratified (GRTS) sample from a SpatialPolygons* object

Usage

```
grts.polygon(x, n, over.n = 0)
```

Arguments

x	A SpatialPolygons or SpatialPolygonsDataFrame object.
n	Sample size. The number of 'sample' points to draw from x
over.n	Over-sample size. The number of 'over-sample' points to draw from x. The actual number of points drawn from x is n + over.n.

Details

This is a wrapper for the grts function in package spsurvey. This simplifies calling grts when equi-probable samples are desired. It extends the allowable input frame types to SpatialPolygons objects (i.e., no attributes), rather than just SpatialPolygonsDataFrame objects. For more complicated designs (e.g., variable probability, stratification), call grts directly.

Value

A SpatialPointsDataFrame containing locations in the GRTS sample, in order they are to be visited. Attributes of the sample points (in the embedded data frame) are as follows:

- sampleID: Unique identifier for points in the sample. This encodes the GRTS ordering of the sample. The output object comes pre-sorted in GRTS order. If the sample becomes un-GRTS-ordered, resort by sampleID (i.e., samp <-samp[order(samp$sampleID),]).
- pointType: A string identifying regular sample points (pointType=="Sample") and over-sample points (pointType=="OverSample").
- geometryID: The ID of the polygon in x which each sample points fall. The ID of polygons in x are row.names(geometry(x)).
- Any attributes of the original polygons (in x).

Additional attributes of the output object, beyond those which make it a SpatialPointsDataFrame, are:

- frame: Name of the input sampling frame.
- frame.type: Type of resource in sampling frame. (i.e., "polygon").
- sample.type: Type of sample drawn. (i.e., "GRTS").
- n: Regular sample size. (i.e., sum(out$pointType=="Sample"))
- over.n: Over-sample size. (i.e., sum(out$pointType=="OverSample"))

Author(s)

Trent McDonald

References

Stevens, D. L. and A. R. Olsen (1999). Spatially restricted surveys over time for aquatic resources. Journal of Agricultural, Biological, and Environmental Statistics 4 (4), 415-428.

Stevens, D. L. and A. R. Olsen (2004). Spatially balanced sampling of natural resources. Journal of the American Statistical Association 99, 262-278.

See Also

grts.line, grts.polygon, hip.polygon, bas.polygon, sdraw

Examples

```
## Not run:
## The following take > 5s to execute

#   Draw sample
WA.sample <- grts.polygon(WA,10,5)

#   Plot
plot( WA )

# Plot 'sample' locations
plot( WA.sample[ WA.sample$pointType == "Sample", ], pch=16, add=TRUE, col="red" )

# Plot 'over sample' locations
plot( WA.sample[ WA.sample$pointType == "OverSample", ], pch=1, add=TRUE, col="blue" )

## End(Not run)
```

---

halton — *Compute points in the Halton sequence.*

Description

Computes points in a multi-dimensional Halton sequence, beginning at specified indices and using specified co-prime bases.

Usage

```
halton(n, dim = 1, start = 0, bases = NULL)
```

Arguments

n	A scalar giving the number of values in the Halton points to produce.
dim	A scalar giving the number of dimensions, equal to the number of van der Corput sequences. Technically, `dim==1` produces a van der Corput sequence, `dim>=2` produces Halton sequences.
start	A scalar or a length `dim` vector giving the starting index (location) for each van der Corput sequence. Origin of each sequence is 0. `all(start>=0)` must be true.

`bases` A length `dim` vector giving the base to use for each dimension. For a Halton sequence, bases must all be co-prime. No check for common prime factors is performed. If `bases` is `NULL`, the first `dim` primes starting at 2 are used as bases of the Halton sequence. For example, the 4-dimensional Halton sequence would use bases 2, 3, 5, and 7. The 6-dimensional Halton sequence would use 2, 3, 5, 7, 11, and 13. Etc.

Details

The Halton sequence is a sequence of `dim`-dimensional numbers where each dimension is a (1-dimensional) co-prime van der Corput sequence. Here, all van der Corput sequences use bases that are prime numbers. See references below.

Value

A matrix of size `n` X `dim`. Each column corresponds to a dimension. Each row is a `dim`-dimensional Halton point.

Author(s)

Trent McDonald

References

van der Corput sequences are described here: http://en.wikipedia.org/wiki/Van_der_Corput_sequence

Halton sequences are described here: http://en.wikipedia.org/wiki/Halton_sequence

Robertson, B.L., J. A. Brown, T. L. McDonald, and P. Jaksons (2013) BAS: "Balanced Acceptance Sampling of Natural Resources", Biometrics, v69, p. 776-784.

See Also

`halton.indices`

Examples

```
halton(10,2)
halton(10,2, floor(runif(2,max=100000))) # A random-start 2-D Halton sequence of length 10
```

`halton.coefficients` *halton.coefficients*

Description

Return the coefficients in the Halton equation for a list of Halton indices (boxes).

Usage

```
halton.coefficients(samp, J, bases = c(2, 3))
```

Arguments

`samp`	A vector of Halton indices.
`J`	A vector of powers of the bases. This determines the level of hierarchy in the Halton boxes and the number of boxes.
`bases`	The bases of the Halton sequence.

Details

Let `digits = halton.coefficients(samp,J,bases)`, `K = max(J)` and `places <-1/matrix(rep(bases,each=K)^(1:K),` The coordinate in [0,1) of the lower left corner of the Halton box with index `samp[i]` is `colSums(digits[i,,] * places,na.rm = T)`. This is how you get the Halton sequence from this routine. However, if you are interested in the Halton sequence alone, not the coefficients, call function `halton()`.

Value

An array of size `length(samp)` X `max(J)` X `length(J)` of coefficients. Row i, column j, page k of this array is the jth coefficient for the kth dimension of the ith index in `samp`.

Author(s)

Trent McDonald

`halton.frame` *Construct a Halton sampling frame.*

Description

Makes a Halton frame from a set of points that have their Halton indices attached. This function identifies points in the same Halton box, and randomly adds Halton cycles to geographically separate nearby points. The frame is then sorted by the new frame indices for sampling.

Usage

```
halton.frame(x, index.name = attr(x, "index.name"), order.name = "HaltonOrder")
```

Arguments

- `x` — Either a data frame or a `SpatialPointsDataFrame` object. In particular, the output of `halton.indices` is acceptable. The data frame, or data frame of the `SpatialPointsDataFrame`, must contain the Halton indices, which is assumed to be named `attr(x,"index.name")`. The default name for this column when using output from `halton.indices` is `HaltonIndex`. Each row of the data frame is a sampling unit to be included in the frame, and the Halton index of the unit is the Halton box the unit falls in. A `SpatialPoints` object without the data frame is not acceptable.
- `index.name` — Name of the Halton index column in the input object. This names the column of `x` containing the index of the Halton box that contains the point represented on that row of `x`. In reality, this could be any ordering for the rows in `x`. Duplicate values of this column are randomly moved to the end of the frame. If `index.name` is missing (the default), `x` is assumed to have an attribute names `index.name`
- `order.name` — Name of the Halton order column in the output object. See description of returned object. This parameter is saved as an attribute of the output object.

Value

A data frame or `SpatialPointsDataFrame` suitable for use as a sampling frame. The data frame, or attributes of the points, contain a new index column separating points in the same Halton box, and the output is sorted by this new index. Returned object has the following attributes,

- `index.name`: Name of the Halton index column used to construct the frame.
- `order.name`: The name of the Halton ordering column. This column is unique across rows of the output, and orders the output frame, but is not consecutive. This differs from column `index.name` because points with identical `index.name` indices have been randomly moved to the end of the frame by adding random Halton cycles.
- `J`: Halton base powers defining lattice of Halton boxes, if `x` has a `J` attribute.
- `bases`: Base powers defining lattice of Halton boxes, if `x` has a `bases` attribute.
- `hl.bbox`: Bounding box for lattice of Halton boxes, if `x` has a `hl.bbox` attribute.

Author(s)

Trent McDonald

See Also

`halton.indices`

Examples

```
# The following is equivalent to hal.point(WA.cities,20,J=c(6,3))

# Define Halton lattice
attr(WA.cities,"J") <- c(6,3)
attr(WA.cities,"bases") <- c(2,3)
```

```
# Add tiny amount to right and top of bounding box because Halton boxes are
# closed on the left and bottom.  This includes points exactly on top and right
# bounding lines.
attr(WA.cities,"hl.bbox") <- bbox(WA.cities) + c(0,0,1,1)

# Compute Halton indices
frame <- halton.indices( WA.cities )

# Separate points in frame that are in same box
frame <- halton.frame( frame )

# Draw sample of size 20
n <- 20
random.start <- floor( runif(1,0,nrow(frame)-1 ) )
samp <- frame[ ( ((0:(n-1))+random.start) %% nrow(frame) ) + 1, ]
```

`halton.indices` *Halton indices*

Description

Compute and attach "inverse" or indices of the Halton sequence to points. Points can be an arbitrary set or a Halton lattice.

Usage

```
halton.indices(
  x,
  J = NULL,
  hl.bbox,
  bases = c(2, 3),
  index.name = "HaltonIndex",
  use.CRT = FALSE
)
```

Arguments

`x` Either a data frame or a `SpatialPoints*` object. Suitable input objects are the output of functions `halton.lattice` (a data frame) and `halton.lattice.polygon` (a `SpatialPointsDataFrame` object).

If `x` is a data frame, it must either contain the names of coordinates columns as attribute "coordnames", or coordinates must be the first D columns of the data frame. I.e., coordinates are either `x[,attr(x,"coordnames")]` or `x[,1:length(bases)]`.

This function works for dimensions >2 if `x` is a data.frame. `SpatialPoints*` objects are not defined for D>2.

J
: A vector of length D containing base powers. J determines the size and shape of the smallest Halton boxes in D space. There are bases[i]^J[i] boxes over the i-th dimension of x's bounding box. Total number Halton boxes is prod(bases^J). The size of each box in the i-th dimension is delta[i]/ (bases[i]^J[i]), where delta[i] is the extent of x's bounding box along the i-th dimension. If J is NULL (the default), approximately length(x) boxes will be chosen (approx. one point per box) and boxes will be as square as possible.

hl.bbox
: DX2 matrix containing bounding box of the full set of Halton boxes. First column of this matrix is the lower-left coordinate (i.e., minimums) of the bounding box. Second column is the upper-right coordinate (i.e., maximums) of the bounding box. For example, if D = 2, hl.bbox = matrix(c(min(x), min(y), max(x), max(y)),2). If hl.bbox is missing (the default), the bounding box of x is used, but expanded on the top and right by 1 percent to include any points exactly on the top and right boundaries. If hl.bbox is supplied, keep in mind that all point outside the box, or on the maximums (i.e., hl.bbox[,2]), will not be assigned Halton indices.

bases
: A vector of length D containing Halton bases. These must be co-prime.

index.name
: A character string giving the name of the column in the output data frame or SpatialPoints object to contain the Halton indices. This name is saved as an attribute attached to the output object.

use.CRT
: A logical values specifying whether to invert the Halton sequence using the Chinese Remainder Theorem (CRT). The other method (use.CRT == FALSE) is a direct method, and is very fast, but requires multiple huge vectors be allocated (size of vectors is prod{bases^J}, see Details). As the number of points grows, eventually the direct method will not be able to allocate sufficient memory (tips: Make sure to run 64-bit R, and try increasing memory limit with memory.limit). The CRT method, while much (much) slower, does not require as much memory, and should eventually complete a much larger problem. Patience is required if your problem is big enough to require use.CRT == TRUE.

Details

Halton indices are the arguments to the Halton sequence. This routine is the inverse function for the Halton sequence. Given a point in D space, this routine computes the index (a non-negative integer) of the Halton sequence which maps to the Halton region containing the point.

For example, in 1D, with bases == 2, J == 3, and hl.bbox= matrix(c(0,1),1), all points in the interval [0,1/8) have Halton index equal to 0, all point in [1/8,2/8) have Halton index 4, points in [2/8,3/8) have index 2, etc. To check, note that the Halton sequence maps x (mod 8) = 4 to the interval [1/8,2/8), x (mod 8) = 2 are mapped to [2/8,3/8), etc. (i.e., check range(halton(200)[(0:199)%% 8 == 4]) and range(halton(200)[(0:199)%% 8 == 2]))

Value

If x is a data frame, x is returned with an addition column. The additional column is named index.name and stores the index of the Halton box containing the point represented on that line of x. If x is a SpatialPoints* object, a SpatialPointsDataFrame is returned containing the points in x. The attributes of the returned object have an additional column, the index of the Halton box containing the point. Name of the attribute is `index.name`. If multiple points fall in the same Halton box, their Halton indices are identical.

Author(s)

Trent McDonald

See Also

`halton.frame`, `hip.point`

Examples

```
# The following is equivalent to hip.point(WA.cities,25,J=c(3,2))
#
# Add tiny amount to right and top of bounding box because Halton boxes are
# closed on the left and bottom.  This includes points exactly on the bounding lines.
bb <- bbox(WA.cities) + c(0,0,1,1)

# Compute Halton indices
frame <- halton.indices( WA.cities, J=c(3,2), hl.bbox=bb  )

# Construct Halton frame
frame <- halton.frame( frame )

# Draw HAL sample
n <- 25
N.frame <- nrow(frame)
m <- floor(runif(1, 0, N.frame)) # Integer 0,...,N.frame-1
ind <- (((0:(n-1))+m) %% N.frame ) + 1  # Cycle around frame if necessary
samp <- frame[ind,]  # draw sample
```

`halton.indices.CRT` *Halton indices by the Chinese Remainder Theorem (CRT)*

Description

Computes Halton indices of D-dimensional points by solving the Chinese Remainder Theorem. This function is slightly slower than `halton.indices.vector`, but it works for large problems.

Usage

```
halton.indices.CRT(
  hl.coords,
  n.boxes,
  D = 2,
  b = c(2, 3),
```

```
    delta = c(1, 1),
    ll.corner = c(0, 0)
  )
```

Arguments

hl.coords	nXD vector of coordinates for points. No points can be outside the bounding box or exactly on the right or top boundary. See Details.
n.boxes	DX1 vector containing number of Halton boxes in each dimension.
D	Number of dimensions
b	DX1 vector of bases to use in the Halton sequence.
delta	DX1 vector of study area bounding box extents in each dimension. Study area is bounded by a cube in D space, which is delta[i] units wide in dimension i. Area of bounding cube is prod{delta} units to the D power.
ll.corner	DX1 vector containing minimum coordinates in all dimensions.

Details

The Halton sequence maps the non-negative integers (the Halton indices) to D-space. This routine does the inverse. Given a point in D-space and a grid of Halton boxes, the point's Halton index is any integer N which gets mapped to the Halton box containing the point. (i.e., any integer in the set ${x:N = x mod C}$, where C = prod(n.boxes)).

This routine solves the Chinese Remainder Theorem to find Halton indices. This routine loops over the points in hl.coords, and as such minimizes memory usage but sacrifices speed. For small problems, see halton.indices.vector, which computes indices by actually placing points in Halton boxes to find their indices.

No point can be less than it's corresponding ll.corner. No point can be equal to or greater than it's corresponding ll.corner + delta.

Note: n.boxes is checked for compatibility with b. That is, log(n.boxes,b) must all be integers.

Value

A nX1 vector of Halton indices corresponding to points in hl.coords.

Author(s)

Trent McDonald

See Also

halton.indices.vector, halton.indices

Examples

```
pt <- data.frame(x=0.43, y=0.64)
n.boxes <- c(16,9)
halton.indices.vector(pt, n.boxes) # should equal 70

# Plot Halton boxes and indices to check.
# pt should plot in box labeled 70
b <- c(2,3)
J <- log(n.boxes,b)  # J must be integers
hl.ind <- halton( prod(n.boxes), 2,0 )
plot(c(0,1),c(0,1),type="n")
for( i in J[1]:1) abline(v=(0:b[1]^i)/b[1]^i, lwd=J[1]+1-i, col=i)
for( i in J[2]:1) abline(h=(0:b[2]^i)/b[2]^i, lwd=J[2]+1-i, col=i)
for( i in 1:prod(n.boxes)){
  box.center <- (floor(n.boxes*hl.ind[i,]+.Machine$double.eps*10) + 1-.5)/n.boxes
  text(box.center[1],box.center[2], i-1, adj=.5)
}
points(pt$x, pt$y, col=6, pch=16, cex=2)

# Longer vector
tmp <- data.frame(x=(0:100)/101,y=.2)
n.boxes <- c(16,9)
tmp.crt <- halton.indices.CRT(tmp, n.boxes)
```

`halton.indices.vector` *Halton indices for an entire vector of coordinates*

Description

Computes Halton indices of an entire vector of points by matching them with a vector of the Halton sequence. This function is relatively fast, but can only handle reasonably sized vectors.

Usage

```
halton.indices.vector(
  hl.coords,
  n.boxes,
  D = 2,
  b = c(2, 3),
  delta = c(1, 1),
  ll.corner = c(0, 0)
)
```

Arguments

`hl.coords`	nXD vector of coordinates for points
`n.boxes`	DX1 vector containing number of Halton boxes in each dimension.

`D`	Number of dimensions
`b`	DX1 vector of bases to use for each dimension
`delta`	DX1 vector of study area extents in each dimension. Study area is `delta[i]` units wide in dimension `i`.
`ll.corner`	DX1 vector containing minimum coordinates in all dimensions.

Details

The Halton sequence maps the non-negative integers (the Halton indices) to D-space. This routine does the inverse. Given a point in D-space and a grid of Halton boxes, the point's Halton index is any integer N which gets mapped to the Halton box containing the point. (i.e., any integer in the set ${x:N = x mod C}$, where C = `prod(n.boxes)`).

This routine uses the Halton sequence and modular arithmetic to find Halton indices. This means several vectors of size `nrow(hl.coords)` must be created. Depending on memory, this approach fails for a sufficiently large number of points. When this routine fails, see the slower `halton.indices.CRT`, which computes indices by solving the Chinese Remainder Theorem.

Value

A nX1 vector of Halton indices corresponding to points in `hl.coords`.

Author(s)

Trent McDonald

See Also

`halton.indices.CRT`, `halton.indices`

Examples

```
# Compute Halton box index for one value
pt <- data.frame(x=0.43, y=0.64)
n.boxes <- c(16,9)
halton.indices.vector(pt, n.boxes) # should equal 70

# The following should also equal 70
pt <- data.frame(x=143, y=164)
halton.indices.vector(pt, n.boxes, delta=c(100,100), ll.corner=c(100,100))

# Plot Halton boxes and indices to check
b <- c(2,3)
J <- c(4,2)  # or, J <- log(n.boxes) / log(b) # = (log base 2 of 16, log base 3 of 9)
hl.ind <- halton( prod(n.boxes), 2,0 )
plot(c(0,1),c(0,1),type="n")
for( i in J[1]:1) abline(v=(0:b[1]^i)/b[1]^i, lwd=J[1]+1-i, col=i)
for( i in J[2]:1) abline(h=(0:b[2]^i)/b[2]^i, lwd=J[2]+1-i, col=i)
for( i in 1:prod(n.boxes)){
  box.center <- (floor(n.boxes*hl.ind[i,]+.Machine$double.eps*10) + 1-.5)/n.boxes
  text(box.center[1],box.center[2], i-1, adj=.5)
```

```
}
points(pt$x, pt$y, col=6, pch=16, cex=2)
```

---

halton.lattice *Halton lattice inside a rectangle*

Description

Constructs a lattice of Halton boxes (a Halton lattice) inside a rectangular box.

Usage

```
halton.lattice(
  box = matrix(c(0, 0, 1, 1), 2),
  N = 10000,
  J = NULL,
  eta = rep(1, nrow(box)),
  triangular = FALSE,
  bases = NULL
)
```

Arguments

`box`	A DX2 matrix containing coordinates of the box. One row per dimension. Column 1 is the minimum, column 2 is the maximum. `box[1,]` contains `c(min,max)` coordinates of the box in dimension 1 (horizontal). `box[2,]` contains `c(min,max)` coordinates of the box in dimension 2 (vertical). Etc for higher dimensions. Default is the 2D unit box.
`N`	Approximate number of points to place in the whole box. If `J` is specified, it takes precedence. If `J` is NULL, the algorithm attempts to place `N` points in the bounding box using Halton boxes that are as close to square as possible. `N` is not exact, but is a target.
`J`	A DX1 vector of base powers which determines the size and shape of the Halton boxes. Elements of `J` less than or equal to 1 are re-set to 1. See additional description in help for `hip.polygon` function.
`eta`	A DX1 vector of the number of points to add inside each Halton box. e.g., if `eta = c(3,2)`, a small grid of 3 by 2 points is added inside each Halton box. `eta[1]` is for the horizontal dimension, `eta[2]` is for the vertical dimension, etc for higher dimensions.
`triangular`	boolean, if TRUE, construct a triangular grid. If FALSE, construct rectangular grid. See help for `hip.polygon`.
`bases`	A DX1 vector of Halton bases. These must be co-prime.

Details

This is designed to be called with the bounding box of a spatial object. See examples.

Definition of Halton lattice: A Halton lattice has the same number of points in every Halton box. Halton boxes are the `bases[1]^J[1]` X `bases[2]^J[2]` matrix of rectangles over a square. Each Halton box contains `prod(eta)` points.

Value

A data frame containing coordinates in the Halton lattice. Names of the coordinates are `dimnames(box)[1]`. If box does not have dimnames, names of the coordinates are `c("d1","d2",...)` (d1 is horizontal, d2 is vertical, etc).

In addition, return has following attributes:

- `J`: the `J` vector used to construct the lattice. This is either the input `J` or the computed `J` when only `N` is specified.
- `eta`: the `eta` vector used in the lattice.
- `bases`: Bases of the van der Corput sequences used in the lattice, one per dimension.
- `triangular`: Whether the lattice is triangular or square.
- `hl.bbox`: The input box. If box does not have dimnames, this attribute will be assigned dimnames of `list(c("d1","d2"),c("min","max"))`.

Author(s)

Trent McDonald

See Also

halton.lattice, hip.polygon

Examples

```
# Lattice of 2^3*3^2 = 72 points in unit box
hl <- halton.lattice( J=c(3,2) )

# Plot
hl.J <- attr(hl,"J")
hl.b <- attr(hl,"bases")
hl.bb <- attr(hl,"hl.bbox")

plot( hl.bb[1,], hl.bb[2,], type="n", pty="s")
points( hl[,1], hl[,2], pch=16, cex=.75, col="red")

for(d in 1:ncol(hl)){
  tmp2 <- hl.bb[d,1] + (0:(hl.b[d]^hl.J[d]))*(diff(hl.bb[d,]))/(hl.b[d]^hl.J[d])
  if( d == 1){
      abline(v=tmp2)
  } else{
      abline(h=tmp2)
```

```
  }
}

# Lattice of approx 1000 points over bounding box of spatial object
hl <- halton.lattice( bbox(HI.coast), N=1000 )
```

---

halton.lattice.polygon   *Halton lattice inside a* SpatialPolygon* *object.*

Description

Constructs a lattice of Halton boxes (a Halton lattice) inside a SpatialPolygons or SpatialPolygons DataFrame object. This is a wrapper for halton.lattice, which does all the hard work.

Usage

```
halton.lattice.polygon(
  x,
  N = 10000,
  J = NULL,
  eta = c(1, 1),
  triangular = FALSE,
  bases = c(2, 3)
)
```

Arguments

x	A SpatialPolygons or SpatialPolygonsDataFrame object.
N	Approximate number of points to place in the lattice. If J is specified, it takes precedence. If J is NULL, the algorithm attempts to place N points in the bounding box using Halton boxes that are as close to square as possible. This N is not exact, but is a target.
J	A 2X1 vector of base powers which determines the size and shape of the Halton boxes. See additional description in help for hip.polygon function.
eta	A 2X1 vector of the number of points to add inside each Halton box. e.g., if eta = c(3,2), a small grid of 3 by 2 points is added inside each Halton box. eta[1] is for the horizontal dimension, eta[2] is for the vertical dimension.
triangular	boolean, if TRUE, construct a triangular grid. If FALSE, construct rectangular grid. See help for hip.polygon.
bases	A 2X1 vector of Halton bases. These must be co-prime.

Details

This routine is called internally by hip.polygon, and is not normally called by the user.

Value

A `SpatialPointsDataFrame` containing locations in the Halton lattice

Attributes of the points are:

- `latticeID`: A unique identifier for every point. ID's are integers numbering points in row-major order from the south.
- `geometryID`: The ID of the polygon in `x` containing each point. The ID of polygons in `x` are `row.names(geometry(x))`.
- Any attributes of the original polygons (in `x`).

Additional attributes of the output object, beyond those which make it a `SpatialPointsDataFrame`, are:

- `J`: the `J` vector used to construct the lattice. This is either the input `J` or the computed `J` when only `N` is specified.
- `eta`: the `eta` vector used in the lattice.
- `bases`: the bases of the van der Corput sequences used in the lattice, one per dimension.
- `triangular`: Whether the lattice is triangular or square.
- `hl.bbox`: the bounding box surrounding the input `x` object. This is saved because bounding box of the return object is not the same as the bounding box of `x` (i.e., `bbox(return) != bbox(x)`).

Author(s)

Trent McDonald

See Also

`hip.polygon`, `halton.lattice`

Examples

```
# Take and plot Halton lattice to illustrate
WA.hgrid <- halton.lattice.polygon( WA, J=c(3,2), eta=c(3,2), triangular=TRUE )
plot(WA)
points(WA.hgrid, pch=16, cex=.5, col="red" )

# Plot the Halton boxes
tmp.J <- attr(WA.hgrid,"J")
tmp.b <- attr(WA.hgrid,"bases")
tmp.bb <- attr(WA.hgrid,"hl.bbox")

for(d in 1:2){
  tmp2 <- tmp.bb[d,1] + (0:(tmp.b[d]^tmp.J[d]))*(diff(tmp.bb[d,]))/(tmp.b[d]^tmp.J[d])
  if( d == 1){
      abline(v=tmp2, col="blue")
  } else{
      abline(h=tmp2, col="blue")
```

```
  }
}

# To explore, re-run the above changing J, eta, and triangular,
```

HI.coast *SpatialLinesDataFrame of the coastline of Hawaii, USA*

Description

A SpatialLinesDataFrame [package "sp"] containing lines outlining the coast of the Hawaiian Islands, USA.

Usage

```
data("HI.coast")
```

Format

A SpatialLinesDataFrame containing 12 lines outlining the coastline of Hawaii. The Shapefile from which this coastline was queried can be found at http://nationalmap.gov/small_scale/atlasftp.html (file 'coastll010g.shp.tar.gz').

From metadata of the shapefile, attributes of the points are as follows:

1. Coastln010 = An internal sequence number delineating lines.
2. Miles = The length of the coastline segment, in miles.

proj4string is `+proj=utm +zone=4 +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0`, meaning among other things that the coordinates are projected (UTM's).

The rectangular bounding box of the polygon is

	min	max
x	371155	940304.1
y	2094278	2458600.9

Examples

```
plot(HI.coast)
```

---

hip.lattice.polygon *Halton Iterative Partition lattice inside a* `bbox` *(bounding box) matrix object.*

Description

Constructs an iteratively partitioned lattice of Halton boxes (a Halton lattice) inside a bounding box bbox of the sample space. This method does the hard work of partitioning the boxes to sample from. It is meant to be used internally by `hip.polygon` only.

Usage

```
hip.lattice.polygon(box, J, bases = c(2, 3))
```

Arguments

`box`	A bbox bounding box for the sample space.
`J`	A 2X1 vector of base powers which determines the size and shape of the Halton boxes. See additional description in help for `hip.polygon` function.
`bases`	A 2X1 vector of Halton bases. These must be co-prime.

Details

This routine is called internally by `hip.polygon`, and is not normally called by the user. This should be avoided

Value

A `list` of `matrices` containing locations in the Halton lattice of the partitioned boxes

Author(s)

Michael J Kleinsasser

See Also

`hip.polygon`, `hip.point`

Examples

```
# Take a simple HIP lattice for illustration
# nboxes = 2^3 * 3^2 = 72
lat1 <- hip.lattice.polygon(box = matrix(data = c(0,1,0,1), nrow = 2, byrow = TRUE),
                   J = c(3,2),
                   bases = c(2,3))

# legth lat1, should be 72
length(lat1)
# prep points for plotting
trans <- list()
i=1
for(mat in lat1) {
  trans[[i]] <-   t(mat)
  i=i+1
```

```
}
# plot points
plot(c(0,1),c(0,1))

for(mat in trans) {
  points(mat[1,1],mat[1,2])
  points(mat[2,1],mat[2,2])
}
```

`hip.plot.lattice` *Plot a Halton Lattice over a polygon resource*

Description

Plots a halton lattice over a polygon resource. Primarily for demonstration of HIP sampling theory.

Usage

```
hip.plot.lattice(resource, bases = c(2, 3), J = c(8, 5), sample = NULL)
```

Arguments

`resource`	`SpatialPolygons` object. The resource the halton lattice will be plotted over.
`bases`	The bases of the halton lattice. Should be equal to those used to draw the sample.
`J`	The orders of the halton lattice. Should be equal to those used to draw the sample.
`sample`	The HIP sample to be plotted with the lattice.

`hip.point` *hip.point - Halton Iterative Partition (HIP) of point resources.*

Description

Draws a Halton Iterative Partition (HIP) sample from a `SpatialPoints*` object.

Usage

```
hip.point(x, n, J = NULL, plot.lattice = FALSE)
```

Arguments

- `x` — A `SpatialPoints` or `SpatialPointsDataFrame` object representing the 2-dimensional point resource from which samples are taken. This object must contain at least 1 point.
- `n` — Sample size. The number locations to draw from the set of points contained in `x`. If the sample size returned is less than the desired sample size, increase `n` until the desired sample size is reached.
- `J` — A 2X1 vector of base powers. `J[1]` is for horizontal, `J[2]` for vertical dimension. `J` determines the size and shape of the smallest Halton boxes. There are `bases[1]^J[1]` vertical columns of Halton boxes over `x`'s bounding box, and `bases[2]^J[2]` horizontal rows of Halton boxes over the bounding box, for a total of `prod(bases^J)` boxes. The dimension of each box is `c(dx,dy)/(bases^J)`, where `c(dx,dy)` are the horizontal and vertical extents of `x`'s bounding box. If `J=NULL` (the default), `J` is chosen so that Halton boxes are as square as possible.
- `plot.lattice` — Boolean. If TRUE, plots the sample drawn with corresponding halton lattice.

Details

A brief description of Halton Iterative Partition (HIP) sampling for points: Given a set of Halton Iterative Partition parameters `x` (SpatialPoints* object) and `n` (sample size), a lattice of Halton boxes is constructed iteratively over the bounding box of `x`. This results in enough Halton boxes on the bounding box to uniquely cover the point resource. That is, one and only one point per box. The Halton index (the inverse of the Halton sequence) of all boxes is computed and assigned to points that lie in each box. Finally, a random number between 0 and the largest Halton index is drawn, and the next `n` points associated with the next `n` Halton boxes are taken as the sample, wrapping to the beginning if necessary.

Value

A `SpatialPoints` objects containing locations in the HIP sample, in HIP order.

Additional attributes of the output object, beyond those which make it a `SpatialPoints`, are:

- `frame`: Name of the input sampling frame.
- `frame.type`: Type of resource in sampling frame. (i.e., "point").
- `sample.type`: Type of sample drawn. (i.e., "HIP").
- `J`: Exponents of the bases used to form the lattice of Halton boxes. This is either the input `J`, or the `J` vector computed by `halton.indices`.
- `bases`: Bases of the Halton sequence used to draw the sample.
- `hl.bbox`: The bounding box around points in `x` used to draw the sample. See `halton.indices`.

Author(s)

Michael Kleinsasser
Aidan McDonald

See Also

hip.polygon, SDraw, bas.point

Examples

```
#   Draw sample of cities in the state of Washington
data(WA.cities)
samp <- hip.point( WA.cities, 100 )
```

---

hip.polygon *Draws a Halton Iterative Partition (HIP) sample from a continuous 2-dimensional (polygon) resource.*

Description

Draws a Halton Iterative Partition (HIP) sample from a SpatialPoints* object.

Usage

```
hip.polygon(x, n, bases = c(2, 3), J = c(8, 5))
```

Arguments

x	A SpatialPoints or SpatialPointsDataFrame object. This object must contain at least 1 point. x is the 2-dimensional point resource from which samples are taken.
n	Target sample size. Target number of locations to draw from the set of points contained in x. If the sample size returned is less than the desired sample size, increase n until the desired sample size is reached.
bases	2X1 vector of Halton bases. These must be co-prime.
J	A 2X1 vector of base powers. J[1] is for horizontal, J[2] for vertical dimension. J determines the size and shape of the smallest Halton boxes. There are bases[1]^J[1] vertical columns of Halton boxes over x's bounding box, and bases[2]^J[2] horizontal rows of Halton boxes over the bounding box, for a total of prod(bases^J) boxes. The dimension of each box is c(dx,dy)/(bases^J), where c(dx,dy) are the horizontal and vertical extents of x's bounding box. If J=NULL (the default), J is chosen so that Halton boxes are as square as possible.

Details

A brief description of Halton Iterative Partition (HIP) sampling for polygons: Given a set of Halton Iterative Partition parameters `x` (SpatialPoints* object), `n` (sample size), `bases`, and `J`, a lattice of Halton boxes is constructed iteratively over the bounding box of the input points. This results in `prod(bases^J)` Halton boxes on the bounding box to cover all points in the point resource. The target should be one point per box, or `prod(bases^J) == n`. The Halton index of all boxes is computed and assigned to points that lie in each box. Finally, a random number between 0 and the largest Halton index is drawn, and the next `n` coordinates in the mapped real numbers are taken as the sample.

Value

A `SpatialPoints*` object containing locations in the HIP sample.

Additional attributes of the output object, beyond those which make it a `SpatialPoints*`, are:

- `frame`: Name of the input sampling frame.
- `frame.type`: Type of resource in sampling frame. (i.e., "polygon").
- `sample.type`: Type of sample drawn. (i.e., "HIP").
- `J`: Exponents of the bases used to form the lattice of Halton boxes. This is either the input `J`, or the `J` vector computed by `halton.indices`.
- `bases`: Bases of the Halton sequence used to draw the sample.
- `hl.bbox`: The bounding box around points in `x` used to draw the sample. See `halton.indices`.

Author(s)

Michael Kleinsasser, Aidan McDonald

See Also

`hip.point`, `SDraw`, `bas.point`

Examples

```
# Draw sample of cities in the state of Washington
data(WA)
samp <- hip.polygon( WA, 100, J = c(3,2))
```

lineLength — *Line length*

Description

An all-R routine that computes total length of all lines in a `SpatialLines*` object.

Usage

```
lineLength(x, byid = FALSE)
```

Arguments

`x`	A spatial object inheriting from `SpatialLines`, `SpatialPolygons`, or `SpatialPoints`.
`byid`	Whether to return lengths of individual spatial objects (TRUE) or the sum of all length (FALSE).

Details

Provides the same answer as `rgeos::gLength`, but is all-R (does not require rgeos Java library) and does not fire a warning if `x` is un-projected (i.e., lat-long).

Value

If `byid==TRUE`, a vector containing the lengths of individual spatial objects (the points, lines, or polygons) is returned. If `byid=FALSE`, the total length of all spatial objects is returned (a single number).

If `x` inherits from `SpatialPoints`, returned value is 0. If `x` inherits from `SpatialLines`, returned value contains line lengths or the sum of line lengths in `x`. If `x` inherits from `SpatialPolygons`, returned value contains lengths of the perimeter of all polygons, or the sum of perimeters, in `x`. When `x` contains polygons with holes, the perimeter of the holes is included (i.e., perimeter of holes is positive, not negative).

Units of the returned value are same as units of coordinates in `x`. E.g., meters if coordinates in `x` are UTM meters, decimal degrees if coordinates in `x` are lat-long decimal degrees.

Author(s)

Trent McDonald

See Also

`sp::SpatialLines-class`

Examples

```
# Length of Hawaii coastline, in kilometers
l <- lineLength( HI.coast ) / 1000
```

`maxU` *Maximum integer used in the BAS routines*

Description

A function that returns the maximum integer used to construct the random-start Halton sequences. By redefining this function and placing in the .GlobalEnv environment, the user can change the maximum integer and hence the number of possible BAS samples.

Usage

```
maxU()
```

Details

CAUTION: The following comment is intended for those who wish to simulate or study statistical properties of BAS, and want to completely enumerate the sample space. Don't do this if you are actually drawing a sample.

To change maxU, redefine maxU() in .GlobalEnv. For example, `maxU <-function() 4`. There are only 25 possible 2D Halton starts in this case. Random starts are = (0,1,2,3,4) X (0,1,2,3,4).

In general, `all.possible.starts = expand.grid(x=0:maxU(),y=0:maxU()))`

Number of possible BAS samples is less than or equal to `(maxU()+1)^2` because the first sample point is required to land in a valid polygon. So, starts that do not land in polygon are discarded.

Value

10e7 or 100,000,000

Author(s)

Trent McDonald

References

Robertson, B.L., J. A. Brown, T. L. McDonald, and P. Jaksons (2013) "BAS: Balanced Acceptance Sampling of Natural Resources", Biometrics, v69, p. 776-784.

See Also

`bas.line`, `bas.point`, `bas.polygon`

Examples

```
# A 2D random-start Halton sequence, length 10, bases c(2,3).
u <- c( floor((maxU()+1)*runif(1)), floor((maxU()+1)*runif(1)))
halt.pts <- halton(10,dim=2,start=u,bases=c(2,3))
```

---

plotLattice *plotLattice*

Description

Plot a Halton lattice

Usage

```
plotLattice(latt, indices = NULL, J = NULL, bases = c(2, 3), box = NULL, ...)
```

Arguments

latt	A list containing the Halton boxes in the lattice. One item in the list for each Halton box. Each list item is a 2 X 2 matrix where first row is min and max of dimension 1, second row is min and max of dimension 2.
indices	The halton indices of all boxes. `length(latt)` must equal `length(indices)` and `indices[i]` is the index of box `latt[[i]]`. If missing or NULL, indices are not printed.
J	A 2X1 vector of base powers. `J[1]` is for horizontal, `J[2]` for vertical dimension. `J` determines the size and shape of the smallest Halton boxes. There are `bases[1]^J[1]` vertical columns of Halton boxes over `x`'s bounding box, and `bases[2]^J[2]` horizontal rows of Halton boxes over the bounding box, for a total of `prod(bases^J)` boxes. The dimension of each box is `c(dx,dy)/(bases^J)`, where `c(dx,dy)` are the horizontal and vertical extents of `x`'s bounding box. If `J=NULL` (the default), `J` is chosen so that Halton boxes are as square as possible.
bases	2X1 vector of Halton bases. These must be co-prime.
box	The bounding box of all the halton boxes in `latt`. If missing or NULL, the min and max extent of `latt` are used.
...	Options of `graphics::lines` that control appearance of the boxes. For example, `lty`, `col`, etc.

Details

Lines on the current plot are produced.

Value

NULL is returned invisibly.

Author(s)

Trent McDonald

@examples lattice <- hip.lattice.polygon(box = matrix(c(0,0,1,1),2), J = c(2,2) bases = c(2,3)) plotLattice(lattice)

---

plotSample *Plot sample and frame*

Description

Plot the sample and optionally the frame, background image (terrain), and lattice (if HAL sample).

Usage

```
plotSample(
  x,
  frame,
  lattice = FALSE,
  bbox = FALSE,
  add = FALSE,
  poly.fill = TRUE
)
```

Arguments

`x`	A `SpatialPointsDataFrame` produced by an SDraw sampling function. For example, as produced by `sdraw(frame,n)`. This object is a standard `SpatialPolygonsDataFrame` object with additional attributes that record the sampling design.
`frame`	The sample frame used to draw the sample contained in `x`. This is either a `SpatialPoints*`, `SpatialLines*`, or `SpatialPolygons*` object.
`lattice`	Logical. Whether to plot the Halton Lattice if `x` is a HAL sample.
`bbox`	Logical. Whether to plot the bounding box if the sample has a bounding box attribute. This generally means `x` is a HAL or BAS sample. `lattice==TRUE` sets bbox == `TRUE`.
`add`	Logical. Whether to add to an existing plot. See Examples.
`poly.fill`	Logical. Whether to fill polygons (TRUE) or leave them transparent (FALSE). Only applies to `SpatialPolygon*` frames.

Value

Nothing (NULL is invisibly returned)

Author(s)

Trent McDonald

See Also

sdraw

Examples

```
data(WY)
samp <- sdraw(WY, 100, type="HIP", J=c(4,3))
plotSample( samp, WY )
plotSample( samp, WY, lattice=TRUE )

# A map-like background under frame and sample ----
# Requires 'OpenStreetMap' package and internet connection
## Not run:
library(OpenStreetMap)
# 1:convert to Lat-Long
WY.ll <- spTransform(WY, CRS("+init=epsg:4326"))
# 2:Specify bounding box for OpenStreetMap
bb.openmap <- bbox(WY.ll)
ULcoords <- c(bb.openmap[2,2], bb.openmap[1,1])
BRcoords <- c(bb.openmap[2,1], bb.openmap[1,2])
# 3:Fetch image (see 'openmap' help for 'type' parameter)
openMap <- OpenStreetMap::openmap(ULcoords, BRcoords, type = "esri")
# 4:Project background image to original coordinate system
openMap <- OpenStreetMap::openproj(openMap, projection = CRS(proj4string(WY)))
# 5:plot background
plot(openMap)
# 6:plot frame and sample
plotSample(samp, WY, add=TRUE, poly.fill=FALSE)

## End(Not run)
```

---

polygonArea *Polygon area*

Description

An all-R routine that computes area of all polygons in a SpatialPolygons* object, taking account of holes.

Usage

```
polygonArea(x)
```

Arguments

x	A spatial object inheriting from SpatialPolygons

Details

Provides the same answer as `rgeos::gArea`, but is all-R (does not require rgeos Java library) and does not fire a warning if `x` is un-projected (i.e., lat-long).

Value

Area of all polygons in `x`, taking account of holes. Units of area are squared units of coordinates in `x`. E.g., square meters if coordinates in `x` are UTM meters, square decimal degrees if coordinates in `x` are lat-long decimal degrees.

Author(s)

Trent McDonald

See Also

`sp::SpatialPolygons-class`

Examples

```
# Area of Washington state, in hectares
a <- polygonArea( WA ) / (100*100)
```

`primes` *Prime numbers*

Description

Returns the first `n` prime numbers (starting at 2)

Usage

```
primes(n)
```

Arguments

`n`	Number prime numbers requested, starting at 2. Maximum is 1e8 or 100,000,000.

Details

This routine is brute-force and works well for the low primes, i.e., for `n` less than a couple hundred thousand. It is not particularly efficient for large `n`. For example, `primes(2000)` on a Windows laptop takes approximately 4 seconds, while `primes(5000)` takes approximately 30 seconds.

Value

A vector of length n containing prime numbers, in order, starting at 2. Note that 1 is prime, but is never included here. I.e., primes(1) equals c(2).

Author(s)

Trent McDonald

Examples

```
primes(4) #  c(2,3,5,7)

# Prime pairs in the first 100
p <- primes(100)
p.diff <- diff(p)
cbind(p[-length(p)][p.diff==2], p[-1][p.diff==2])
```

---

sdraw.SpatialLines *Sample draws from spatial objects.*

Description

Draw samples (point locations) from SpatialPoints, SpatialLines, SpatialPolygons, and the *DataFrame varieties of each.

Usage

```
## S3 method for class 'SpatialLines'
sdraw(x, n, type, ...)

## S3 method for class 'SpatialPoints'
sdraw(x, n, type, ...)

## S3 method for class 'SpatialPolygons'
sdraw(x, n, type, ...)

sdraw(x, n, type = "BAS", ...)
```

Arguments

x	A spatial object. Methods are implemented for SpatialPoints, SpatialPoints DataFrame, SpatialLines, SpatialLinesDataFrame, SpatialPolygons, and Spatial PolygonsDataFrame objects.
n	Desired sample size. Some type's of samples are fixed-size (see DETAILS), in which case exactly n points are returned. Other type's are variable-size, and this number is the expected sample size (i.e., average over many repetitions).
type	Character, naming the type of sample to draw. Valid type's are:

- "BAS" : Balanced Acceptance Sampling (Robertson et al., 2013)
- "SSS" : Simple Systematic (grid) Sampling, with random start and orientation
- "GRTS" : Generalized Random Tessellation Stratified sampling (Stevens and Olsen, 2004)
- "SRS" : Simple Random Sampling
- "HIP" : Halton Iterative Partitioning (Robertson et al., 2017)

... Optional arguments passed to underlying sample type method. See DETAILS.

Details

This is a S4 generic method for types SpatialPoints*, SpatialLines*, and SpatialPolygons* objects.

BAS,GRTS,SRS,HIP are fixed-size designs (return exactly n points). The SSS algorithm applied to Line and Point is fixed-sized. The SSS method applied to Polygon frames is variable-sized.

Options which determine characteristics of each sample time are passed via .... For example, spacing and "shape" of the grid in sss.* are controlled via spacing= and triangular=, while the J parameter (which determine box sizes) is passed to hip.*. See documentation for hip.*, bas.*, sss.*, grts.*, and sss.* for the full list of parameters which determine sample characteristics.

Value

A SpatialPointsDataFrame object. At a minimum, the data frame embedded in the SpatialPoints object contains a column named siteID which numbers the points, and geometryID which contains the ID of the spatial object from which the point was drawn. If x is a Spatial*DataFrame, the return's data frame contains all attributes of x evaluated at the locations of the sample points.

Certain sampling routine add attributes that are pertinent to the design. For example, the grts.* routines add a pointType attribute. See documentation for the underlying sampling routine to interpret extra output point attributes.

Author(s)

Trent McDonald

References

Robertson, B.L., J. A. Brown, T. L. McDonald, and P. Jaksons (2013) "BAS: Balanced Acceptance Sampling of Natural Resources", Biometrics, v69, p. 776-784.

Stevens D. L. Jr. and A. R. Olsen (2004) "Spatially Balanced Sampling of Natural Resources", Journal of the American Statistical Association, v99, p. 262-278.

See Also

bas.polygon, bas.line, bas.point, hip.polygon, hip.point, sss.polygon, sss.line, sss.point, grts.polygon, grts.line, grts.point

Examples

```
 ## Not run:
 WA.sample <- sdraw(WA, 50, "BAS")
 WA.sample <- sdraw(WA, 50, "HIP", J=c(4,3))
 WA.sample <- sdraw(WA, 50, "SSS", spacing=c(1,2))

## End(Not run)
```

`srs.line` *Draw a Simple Random Sample (SRS) from a linear resource.*

Description

Draws a simple random sample from a `SpatialLines*` object. The `SpatialLines*` object represents a 2-dimensional line resource, such as a river, highway, or coastline.

Usage

```
srs.line(x, n)
```

Arguments

`x`	A `SpatialLines` or `SpatialLinesDataFrame` object. This object must contain at least 1 line.
`n`	Sample size. Number of points to draw from the set of all lines contained in `x`.

Details

If `x` contains multiple lines, the lines are amalgamated before sampling. Conceptually, under amalgamation the lines in `x` are "stretched" straight and laid end-to-end in order of appearance in `x`. The simple random sample is then drawn from the amalgamated line. Once drawn from the 1-D amalgamated line, sample points are mapped back to 2-dimensional space to fall on the lines in `x`.

Note that the line is not discretized prior to sampling. The sample points are selected from the set of continuous lines that contain an infinite number of points (up to machine precision anyway).

Value

A `SpatialPointsDataFrame` containing locations in the SRS sample, in order along the amalgamated line. Those on line 1 appear first, those on line 2 second, etc. Attributes of the sample points (in the embedded data frame) are as follows:

- `sampleID`: A unique identifier for every sample point. `sampleID` starts with 1 at the first point and increments by one for each. `sampleID` orders sample points along the amalgamated line.
- `geometryID`: The ID of the lines object in `x` on which each sample point falls. The ID of lines in `x` are `row.names(geometry(x))`.
- Any attributes of the original lines (in `x`) on which each sample point falls.

Additional attributes of the output object, beyond those which make it a SpatialPointsDataFrame, are:

- frame: Name of the input sampling frame.
- frame.type: Type of resource in sampling frame. (i.e., "line").
- sample.type: Type of sample drawn. (i.e., "SRS").

Author(s)

Trent McDonald

See Also

srs.polygon, srs.point, sdraw

Examples

```
# Draw fixed number of equi-distant points
HI.samp <- srs.line( HI.coast, 100 )
plot( HI.coast, col=rainbow(length(HI.coast)) )
points( HI.samp, col="red", pch=16 )

# Inspect attributes of points with HI.samp@data
```

---

srs.point — *Draw a Simple Random Sample (SRS) from a point resource or finite population frame.*

Description

Draw a systematic sample from a SpatialPoints* object or a data.frame. SpatialPoints* objects can represent point resources in 2-dimensional space, such as towns, event locations, or grid cell centers.

Usage

```
srs.point(x, n)
```

Arguments

x	A SpatialLines, SpatialLinesDataFrame, or data.frame object.
n	Sample size. Number of points or rows to draw from x. If n exceeds the number of units (= number of rows in data.frame(x)), a census is taken (i.e., x is returned).

Details

When x is a data frame, the simple random sample is drawn from the rows. That is, each row is viewed as a sample unit.

This draws equi-probable sample. First order inclusion probabilities are n/N for all units.

Value

If input x inherits from a the SpatialPoints class, a SpatialPointsDataFrame object containing locations and attributes in the sample is returned. If input x is a data.frame, a data.frame is returned. Attributes of the returned sample points are:

- sampleID: A unique identifier for every sample point. sampleID starts with 1 at the first point and increments by one for each.
- If x inherits from SpatialPoints, returned points have attribute geometryID – the ID (= row.names(x)) of the sampled point.
- Any attributes (columns) associated with the input points (rows).

Additional attributes of the output object are:

- frame: Name of the input sampling frame (i.e., x).
- frame.type: Type of resource in sampling frame. (i.e., "point").
- sample.type: Type of sample drawn. (i.e., "SRS").

Author(s)

Trent McDonald

See Also

srs.polygon, srs.line, sdraw

Examples

```
# Draw systematic sample across range of population
WA.samp <- srs.point( WA.cities, 100 )
plot( WA.cities )
points( WA.samp, col="red", pch=16 )

# Draw systematic sample from data frame
df <- data.frame( a=1:100, b=runif(100) )
samp <- srs.point( df, 5 )
```

`srs.polygon` *Draws a Simple Random Sample (SRS) from an area resource (polygons).*

Description

Draws a simple random sample from a `SpatialPolygons` or `SpatialPolygonsDataFrame` object.

Usage

```
srs.polygon(x, n)
```

Arguments

`x`	A `SpatialPolygons` or `SpatialPolygonsDataFrame` object. This object must contain at least 1 polygon. If it contains more than 1 polygon, the SRS sample is drawn from the union of all polygons. Holes are respected.
`n`	Sample size. Number of locations to draw from the union of all polygons contained in `x`.

Details

The SRS sample is drawn by generating uniform random deviates for coordinates in the bounding box surrounding polygons (e.g., `c(xmin,ymin) + c(dx,dy)*runif(2)`), tossing locations outside polygons until the required number is achieved.

Value

A `SpatialPointsDataFrame` containing locations in the SRS sample, in arbitrary order. Attributes of the sample points (in the embedded data frame) are as follows:

- `sampleID`: A unique identifier for every sample point.
- `geometryID`: The ID of the polygon in `x` which each sample point falls. The ID of polygons in `x` are `row.names(geometry(x))`.
- Any attributes of the original polygons (in `x`).

Additional attributes of the output object, beyond those which make it a `SpatialPointsDataFrame`, are:

- `frame`: Name of the input sampling frame.
- `frame.type`: Type of resource in sampling frame. (i.e., "polygon").
- `sample.type`: Type of sample drawn. (i.e., "SRS").

Author(s)

Trent McDonald

See Also

bas.polygon, sss.polygon, hip.polygon, sdraw

Examples

```
# A square grid oriented east-west
WA.samp <- srs.polygon( WA, 100 )
plot( WA )
points( WA.samp, pch=16 )
```

---

sss.line *Draw a Simple Systematic Sample (SSS) from a linear resource.*

Description

Draws a systematic sample from a SpatialLines* object. The SpatialLines* object represents a 2-dimensional line resource, such as a river, highway, or coastline.

Usage

```
sss.line(x, n, spacing, random.start = TRUE)
```

Arguments

x	A SpatialLines or SpatialLinesDataFrame object. This object must contain at least 1 line.
n	Sample size. Number of points to draw from the set of all lines contained in x. Specification of n takes precedence over specification of spacing.
spacing	Assuming, n is not given, this is the distance between sample points on the amalgamated line in x. For example, if x is projected in UTM coordinates and spacing=100, the returned sample has one point every 100 meters along the amalgamated line in x. Keep in mind that the start of line i+1 in x may not coincide with the end of line i in x, and that lines in x may not be straight. Thus, 2-dimensional distances between sample points will not, in general, equal spacing.
random.start	Whether to start the sequence of points at a random place. If TRUE, a random uniform variate is selected between 0 and either spacing or (length/n) and the first location is placed at that location along the line. Subsequent points occur every spacing units along the lines. If random.start==FALSE, the first sample point occurs at 0 (first vertex of the lines).

Details

If `x` contains multiple lines, the lines are amalgamated before sampling. Conceptually, under amalgamation the lines in `x` are "stretched" straight and laid end-to-end in order of appearance in `x`. The simple systematic sample is then drawn from the amalgamated line. Finally, sample points on the amalgamated line are mapped back to 2-dimensional space to fall on the lines in `x`.

Note that spacing between sample points is enforced on the amalgamated line, and may not look correct if the lines loop back on themselves. For example, consider a line tracing a circle. The spacing between the first and last sample point along the circle will not be the prescribed `spacing` because the circle starts between them. Spacing of all other points (2 to n-1) will be as requested.

Value

A `SpatialPointsDataFrame` containing locations in the SSS sample, in order along the amalgamated line. Those on line 1 appear first, those on line 2 second, etc. Attributes of the sample points (in the embedded data frame) are as follows:

- `sampleID`: A unique identifier for every sample point. `sampleID` starts with 1 at the first point and increments by one for each. `sampleID` orders sample points along the amalgamated line.
- `geometryID`: The ID of the lines object in `x` on which each sample point falls. The ID of lines in `x` are `row.names(geometry(x))`.
- Any attributes of the original lines (in `x`) on which each sample point falls.

Additional attributes of the output object, beyond those which make it a `SpatialPointsDataFrame`, are:

- `frame`: Name of the input sampling frame.
- `frame.type`: Type of resource in sampling frame. (i.e., "line").
- `sample.type`: Type of sample drawn. (i.e., "SSS").
- `sample.spacing`: The spacing between sample points along the amalgamated line. This is the input `spacing` parameter if specified, or is computed as (length/n) if `n` is specified.
- `random.start`: The random start of the systematic sample. NA corresponds to no random start.

Author(s)

Trent McDonald

See Also

`sss.polygon`, `sss.point`, `sdraw`

Examples

```
# Draw fixed number of equi-distant points
HI.samp <- sss.line( HI.coast, 100 )
plot( HI.coast, col=rainbow(length(HI.coast)) )
points( HI.samp, col="red", pch=16 )
```

```
# Draw points every 20 km along Hawaii's coastline
HI.samp <- sss.line( HI.coast, spacing=20000 )
plot( HI.coast, col=rainbow(length(HI.coast)) )
points( HI.samp, col="red", pch=16 )

# Inspect attributes of points with HI.samp@data
```

`sss.point` *Draw a Simple Systematic Sample (SSS) from a point resource or finite population frame.*

Description

Draw a systematic sample from a `SpatialPoints*` object or a `data.frame`. `SpatialPoints*` objects can represent point resources in 2-dimensional space, such as towns, event locations, or grid cell centers.

Usage

```
sss.point(x, n)
```

Arguments

`x`	A `SpatialLines`, `SpatialLinesDataFrame`, or `data.frame` object.
`n`	Sample size. Number of points to draw from the set of all points in `x`. If `n` exceeds the number of units (= number of rows in `data.frame(x)`), a census is taken (i.e., `x` is returned).

Details

The points in `x` are systematically sampled in the order they appear. That is, the sampling frame (i.e., `data.frame(x)`) is *not* re-ordered prior to sampling. Each row in the frame represents a point or sample unit, and rows are sampled systematically starting with row 1. To draw a systematic sample across the range of an attribute, say attribute y, sort `x` by y prior to calling this routine (e.g,. `sss.point( x[order(x$y),],n ))`.

This routine draws *fixed size* systematic samples. Many systematic sampling procedure produce variable size samples. Conceptually, the sample procedure is:

1. Each sample unit (= row of sample frame) is associated with a line segment. Assuming there are N units in the frame (N = `nrow(x)`), each line segment has length n/N, where n is the input desired sample size.
2. Line segments are placed end-to-end, starting at 0, in the order in which their associated unit appears in the frame.
3. To start the systematic sample, the routine chooses a random number between 0 and 1. Let this random number be m.
4. The sample units associated with the line segments containing the numbers $m + i$ for i = 0,1,...,$(n - 1)$, are selected for the sample.

Value

If input x inherits from a the SpatialPointsDataFrame class, a SpatialPointsDataFrame object containing locations in the sample is returned. If input x is a data.frame, a data.frame is returned. Attributes of the returned sample points are:

- sampleID: A unique identifier for every sample point. sampleID starts with 1 at the first point and increments by one for each.
- If x inherits from SpatialPoints, returned points have attribute geometryID – the ID (= row.names(x)) of the sampled point.
- Any attributes (columns) associated with the input points (rows).

Additional attributes of the output object are:

- frame: Name of the input sampling frame (i.e., x).
- frame.type: Type of resource in sampling frame. (i.e., "point").
- sample.type: Type of sample drawn. (i.e., "SSS").
- random.start: The random start for the systematic sample.

Using these additional attributes, one could reconstruct the sample.

Author(s)

Trent McDonald

See Also

sss.polygon, sss.line, sdraw

Examples

```
# Draw systematic sample across range of population
WA.samp <- sss.point( WA.cities[order(WA.cities$POP_2010),], 100 )
plot( WA.cities )
points( WA.samp, col="red", pch=16 )

# Draw systematic sample from data frame
df <- data.frame( a=1:100, b=runif(100) )
samp <- sss.point( df, 5 )

# Equivalent to simple random sample: randomly sort frame.
samp <- sss.point( df[order(df$b),], 5 )
```

sss.polygon *Draws a Simple Systematic Sample (SSS) from an area resource (polygons).*

Description

Draws a systematic, or grid, sample from a `SpatialPolygons` or `SpatialPolygonsDataFrame` object. Optional parameters control control the relative spacing in horizontal and vertical directions, whether a square or triangular grid is produced, and whether the grid baseline has random orientation.

Usage

```
sss.polygon(x, n, spacing = c(1, 1), triangular = FALSE, rand.dir = FALSE)
```

Arguments

x	A `SpatialPolygons` or `SpatialPolygonsDataFrame` object. This object must contain at least 1 polygon. If it contains more than 1 polygon, the SSS sample is drawn from the union of all polygons. Holes are respected.
n	Sample size. Number of locations to draw from the union of all polygons contained in `x`.
spacing	A vector of length 2 containing the RELATIVE spacing of grid points in the horizontal (X) and vertical (Y) directions. See details.
triangular	Boolean scalar specifying whether to produce a rectangular (`triangular==FALSE`) or triangular (`triangular==TRUE`) grid. See Details.
rand.dir	Either a boolean scalar specifying whether to randomly orient the grid's horizontal axis (`rand.dir==TRUE`) or not (`rand.dir==FALSE`), or a user-specified fixed direction for the horizontal axis. If FALSE, orientation of the grid is parallel to the X and Y axes. If TRUE, the X axis of the grid is randomly rotated by an angle between -pi/4 (-45 degrees) and pi/4 (45 degrees). If `rand.dir` is a number, the grid is rotated by that many radians. No range check is performed on user-specified `rand.dir`, so for example, rotation by `pi/8` is equivalent to rotation by `pi/8 + 2*pi`. User-specified, but random, direction of the grid can be specified by `rand.dir = runif(1,0,pi)`. Note, relative spacing of the grid cells is computed prior to rotation.

Details

The projection system of the input shape object (`x`) is not considered. But, a projected coordinate system is necessary to obtain correct spacing on the ground. The author STRONGLY recommends converting `x` to a UTM coordinate system prior to calling this function.

Spacing (size and shape of grid cells) is determined by `n` and `spacing`. If `spacing` is not given, grid spacing is equal in X and Y directions, which produces square grid cells. In this case, grid spacing is `delta` (= `sqrt(A/n)`, where `A` = area of union of all polygons in `x`.

Relative shape of grid cells is controlled by the spacing vector. If spacing = c(rx,ry), spacing in X and Y directions is spacing*delta/rev(spacing), where delta = sqrt(A/n). Conceptually, a square cell of size delta^2 is "stretched" multiplicatively by rx in the X direction and ry in the Y direction. After stretching, the area of each cell remains delta^2 while the relative lengths of the (rectangular) cell sides is 1 to (ry/rx)^2. That is, vertical dimension of each cell is (ry/rx)^2 times the horizontal dimension. Vice versa, the horizontal dimension is (rx/ry)^2 times the vertical.

In general, realized sample size is not fixed. Across multiple calls, realized sample size will not always equal n. Across an infinite number of calls, the average sample size will be n

In all cases, the grid is randomly shifted in the X and Y directions, before rotation (if called for). The amount of the random shift is less than the X and Y extent of cells, and is returned as an attribute of the sample.

Value

A SpatialPointsDataFrame containing locations in the SSS sample, in row-wise order starting in the south (see sampleID, row, col in returned data frame). Attributes of the sample points (in the embedded data frame) are as follows:

- sampleID: A unique identifier for every sample point. For rectangular grids, sampleID is incremented west to east by row from the south. For triangular grids, sampleID is assigned west to east to points in every other row from the south. Then, starts over in the southwest and assigns ID's to previously-skipped rows.
- row: Row number of the sampled point in the grid. Row numbers are the vertical indices of the grid in a direction perpendicular to the (potentially rotated) main horizontal axis. Cell (1,1) is in the lower left (southwest) corner of the shape's bounding box. Thus, row 1 is defined along the lower (southern) boundary of the shape's bounding box. Points in row 1 may not be inside the shape and therefore may not appear in the sample. Consequently, the lowest row appearing in the sample may not be 1. Visualize row i with points(samp[samp$row==i,]).
- col: Column number of the sampled point in the grid. Column numbers are the horizontal indices of the grid in a direction parallel to the (potentially rotated) main horizontal axis. Cell (1,1) is in the lower left (southwest) corner of the shape's bounding box. Thus, column 1 is defined along the left (western) boundary of the shape's bounding box. Points in column 1 may not be inside the shape and therefore may not appear in the sample. Consequently, the lowest column appearing in the sample may not be 1. Visualize column i with points(samp[samp$col==i,]).
- geometryID: The ID of the polygon in x which each sample point falls. The ID of polygons in x are row.names(geometry(x)).
- Any attributes of the original polygons (in x).

Additional attributes of the output object, beyond those which make it a SpatialPointsDataFrame, are:

- frame: Name of the input sampling frame.
- frame.type: Type of resource in sampling frame. (i.e., "polygon").
- sample.type: Type of sample drawn. (i.e., "SSS").

- `spacing.m`: A vector of length 2 giving the dimensions of cells in units of the coordinates of `x`. (e.g., meters). This is the final `delta` computed above. Each cell has size `prod(spacing.m)` = Area / `n`.
- `rand.dir`: The (potentially randomly chosen) direction for the grid's horizontal axis. This is in radians between -pi/4 and pi/4. `rand.dir` = 0 corresponds to no rotation (i.e., `rand.dir` = `FALSE`).
- `rand.shift`: The random shift of the grid. This is a vector of length 2 containing the random shifts in the horizontal and vertical directions before rotation. The random shift in both directions is chosen between 0 and the corresponding element of the `spacing.m` attribute (described above).
- `triangular`: TRUE or FALSE depending on whether the output grid is triangular or rectangular, respectively.

Author(s)

Trent McDonald

See Also

`bas.polygon`, `sdraw`

Examples

```
# A square grid oriented east-west
WA.samp <- sss.polygon( WA, 100 )
plot( WA )
points( WA.samp )

# A rectangular grid oriented east-west, with relative spacing c(0.667, 1.5),
# or 1 to 2.25.
WA.samp <- sss.polygon( WA, 100, spacing=c(2,3) )
plot( WA )
points( WA.samp )

# A rectangular grid oriented east-west, with x spacing = 2*(y spacing).
WA.samp <- sss.polygon( WA, 100, spacing=c(sqrt(2),1) )

# A rectangular grid, random orientation, with y spacing = 3*(x spacing)
WA.samp <- sss.polygon( WA, 100, spacing=c(1,sqrt(3)), rand.dir=TRUE )

# A triangular grid oriented east-west
WA.samp <- sss.polygon( WA, 100, triangular=TRUE )

# A triangular grid oriented east-west, with relative spacing c(.667,1.5)
WA.samp <- sss.polygon( WA, 100, spacing=c(2,3), triangular=TRUE )
```

voronoi.polygons *Calculate Voronoi polygons for a set of points*

Description

Calculate Voronoi polygons (or tessellations) from a `SpatialPoints*` object

Usage

```
voronoi.polygons(x, bounding.polygon = NULL, range.expand = 0.1)
```

Arguments

`x` A `SpatialPoints` or `SpatialPointsDataFrame` object

`bounding.polygon` If present, this is a `SpatialPolygons*` object specifying the bounding polygon(s) for the Voronoi polygons. If present, the Voronoi polygons are clipped to the outside bounding polygon of `bounding.polygon`. The outside bounding polygon is the union of all polygons in `bounding.polygon`. If this is not present, the Voronoi polygons extend to a rectangle that is `range.expand` beyond the bounding box of input points in all directions.

`range.expand` A length-one or length-two vector of expansion factors for the bounding box of points in `x` in the horizontal and vertical directions. If length one, it is replicated to length two. Element one is the fraction of the bounding box's horizontal width that is added and subtracted to the horizontal extent of the output polygons. Element two is the fraction of the bounding box's vertical height that is added and subtracted to the vertical extent of the output polygons. Only this parameter's absolute value is used (i.e., all values are made positive). If `bounding.polygon` is present, this parameter is ignored.

Details

This is a convenience routine for the `deldir::deldir` function. The hard work, computing the Voronoi polygons, is done by the `deldir::deldir` and `deldir::tile.list` functions. See documentation for those functions for details of computations.

This function is convenient because it takes a `SpatialPoints*` object and returns a `SpatialPolygonsDataFrame` object.

Value

A `SpatialPolygonsDataFrame` containing the Voronoi polygons (or tessellations) surrounding the points in `x`. Attributes of the output polygons are:

- x : the horizontal coordinate of the tessellation's defining point
- y : the vertical coordinate of the tessellation's defining point
- area : area of tessellation, in units of `x`'s projection.

Examples

```
# Triangular grid inside a set of polygons
WA.samp <- sss.polygon(WA,100,triangular=TRUE)

# Voronoi polygons of triangular grid
WA.tess <- voronoi.polygons(WA.samp)

# Plot
plot(WA)
plot(WA.tess, add=TRUE, col=rainbow(length(WA.samp)))
plot(WA.samp, add=TRUE, pch=16)

# One way to measure spatial balance:
# Compare variance of Voronoi polygons to same sized
# SRS sample.
WA.bas <- bas.polygon(WA, 100)
WA.srs <- srs.polygon(WA, 100)
WA.bas.tess <- voronoi.polygons(WA.bas)
WA.srs.tess <- voronoi.polygons(WA.srs)
rel.balance <- var(WA.bas.tess$area)/var(WA.srs.tess$area)

# Example clipping to fixed polygon (from @paul-vdb)
## Not run:
set.seed(101)
pts <- SpatialPoints(cbind(runif(1000), runif(1000)))
smp <- pts[sample(1:length(pts), 10),]
bound.pts <- cbind(c(0.2503111693,  0.5215198166,  0.8074680642,
                    0.9312807075,  0.9047494268,  0.7750409433,
                    0.3033737308,  0.0000000000,  0.0321650835,
                    0.0321650835),
                  c(0.03098592, 0.14595480, 0.03688176,
                    0.25502784, 0.89472650, 1.00000000,
                    0.80334098, 0.52918441, 0.14005896,
                    0.14005896))
bounding.poly <- SpatialPolygons(list(Polygons(list(Polygon(bound.pts)), "b")), as.integer(1))
vor <- SDraw::voronoi.polygons(smp, bounding.poly)
plot(vor)
points(pts, pch = 20)
points(smp, col = "red", pch = 20, cex=2)
plot(bounding.poly, border="blue", lwd=2, add=T)

## End(Not run)
```

WA *SpatialPolygonsDataFrame for the state of Washington, USA*

Description

A SpatialPolygonsDataFrame [package "sp"] containing polygons that comprise the state of Washington.

Usage

```
data("WA")
```

Format

A SpatialPolygonsDataFrame containing 50 polygons whose union outline boundaries of the state of Washington. Source of the Shapefile from which these polygons were queried is `http://nationalmap.gov/small_scale/atlasftp.html` (file 'statesp020.tar.gz').

Attributes of the polygons are:

1. AREA = Size of the polygon in square kilometers.
2. PERIMETER = The perimeter of polygon in kilometers.
3. STATESP020 = Internal feature number
4. STATE = The name of the State or State equivalent.
5. STATE_FIPS = The 2-digit FIPS code of the State or State equivalent.
6. ORDER_ADM = An ordinal value indicating the State's order of admission to the United States.
7. MONTH_ADM = The month when the State was admitted to the United States.
8. DAY_ADM = The day when the State was admitted to the United States.
9. YEAR_ADM = The year when the State was admitted to the United States.
10. LAND_TYPE = Type of the polygon. Types are "ISLAND", "MAINLAND", "OCEAN"

The proj4string is `+proj=utm +zone=10 +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0`, meaning among other things that the coordinates are projected zone 10 UTM's in meters. The rectangular bounding box of all polygons is

	min	max
x	369439	971361.3
y	5044642	5444677.5

Examples

```
plot(WA[WA$LAND_TYPE == "MAINLAND",], col="red")
plot(WA[WA$LAND_TYPE == "ISLAND",], col="blue",add=TRUE)
plot(WA[WA$LAND_TYPE == "OCEAN",], col="turquoise",add=TRUE)
```

`WA.cities` *SpatialPointsDataFrame of cities in Washington state, USA*

Description

A SpatialPointsDataFrame [package "sp"] containing the locations of cities in Washington state, USA.

Usage

```
data("WA.cities")
```

Format

A SpatialPointsDataFrame containing one point for each of 815 cities in Washington state. Source of the Shapefile from which these cities were queried is http://nationalmap.gov/small_scale/atlasftp.html (file 'citiesx010g.shp.tar.gz').

The attributes of each point are:

1. GNIS_ID = A unique identification number assigned by the Geographic Names Information System (GNIS). This number can be used to link places in this data set with GNIS.
2. ANSICODE = A unique identification number assigned by the U.S. Census Bureau. This number can be used to link places in this data set with the Census Gazetteer data.
3. FEATURE = The type of feature, as assigned by GNIS. Values are 'Census', 'Civil', and 'Populated Place'.
4. FEATURE2 = The status of the city or town. Values are -999 (missing), 'County Seat', and 'State Capital County Seat'.
5. NAME = The name of the city or town.
6. POP_2010 = The 2010 population of the city or town. Locations with a population of 0 are listed as such in the Census source data.
7. COUNTY = The name of the county or county equivalent where the city or town is located.
8. COUNTYFIPS = The 3-digit FIPS code of the county or county equivalent.
9. STATE = The 2-character abbreviation for the State in which the city or town is located. Values are 'WA'.
10. STATE_FIPS = The 2-digit FIPS code for the State in which the city or town is located.
11. LATITUDE = The latitude of the city or town as it appears in this data set.
12. LONGITUDE = The longitude of the city or town as it appears in this data set.
13. PopPlLat = The latitude of the city or town as it appears in the source data.
14. PopPlLong = The longitude of the city or town as it appears in the source data.
15. ELEV_IN_M = The elevation, in meters, of the city or town. Determined from GNIS or from topographic map sources.
16. ELEV_IN_FT = The elevation, in feet, of the city or town. Determined from GNIS or from topographic map sources.

proj4string is `+proj=utm +zone=10 +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0`, meaning among other things that the coordinates are projected zone 10 UTM's in meters.

The rectangular bounding box containing all points is

	min	max
x	377703.1	957996.8
y	5047878.6	5438319.2

Examples

```
max.popln <- max(WA.cities$POP_2010)
plot(WA.cities, pch=16, cex=5*WA.cities$POP_2010/max.popln, col="red" )
```

WY *SpatialPolygonsDataFrame of counties in the state of Wyoming, USA*

Description

A `SpatialPolygonsDataFrame` containing polygons for the 23 counties in the state of Wyoming.

Usage

```
data("WY")
```

Format

A `SpatialPolygonsDataFrame` containing 23 polygons whose union outline boundaries of the state of Wyoming. Source shapefile was from the Census Bureau in 2015; but, that shapefile has been taken down. A secondary source of the Wyoming counties shapefile, with different attributes, is here: https://www.arcgis.com/home/item.html?id=65b84d3d0c59441596c900d24196d4fd

Attributes of the polygons are:

1. STATEFP = State identifier (56 = Wyoming)
2. COUNTYFP = Unique identifier for county
3. NAME = Name of the county

The proj4string is `"+init=epsg:26912} \code{+proj=utm} \code{+zone=12} \code{+datum=NAD83} \code{+units=m} \code{+no_defs} \code{+ellps=GRS80} \code{+towgs84=0,0,0"`, meaning among other things that the coordinates are projected zone 12 UTM's in meters.

The rectangular bounding box of all polygons is

	min	max
x	495506	1084419
y	4538294	5006162

Examples

```
plot(WY, col=rainbow(length(WY)))
```

Index

∗ **design**

bas.line, 4
bas.point, 6
bas.polygon, 7
grts.line, 11
grts.point, 13
grts.polygon, 14
halton, 16
halton.lattice.polygon, 28
hip.lattice.polygon, 30
hip.point, 32
hip.polygon, 34
srs.line, 44
srs.point, 45
srs.polygon, 47
sss.line, 48
sss.point, 50
sss.polygon, 52

∗ **package**

SDraw-package, 3

∗ **survey**

bas.line, 4
bas.point, 6
bas.polygon, 7
grts.line, 11
grts.point, 13
grts.polygon, 14
halton, 16
halton.lattice.polygon, 28
hip.lattice.polygon, 30
hip.point, 32
hip.polygon, 34
srs.line, 44
srs.point, 45
srs.polygon, 47
sss.line, 48
sss.point, 50
sss.polygon, 52

_PACKAGE *(SDraw-package)*, 3

BAS-package *(SDraw-package)*, 3
bas.line, *3*, 4, *5*, *7*, *9*, *37*, *43*
bas.point, *3–5*, 6, *9*, *34, 35*, *37*, *43*
bas.polygon, *3*, *5–7*, 7, *16*, *37*, *43*, *48*, *54*

extended.gcd, 9

getJ, 10
GRTS *(SDraw-package)*, 3
grts.line, *3*, 11, *12*, *14*, *16*, *43*
grts.point, *3*, 13, *43*
grts.polygon, *3*, *12*, *14*, 14, *16*, *43*

HAL-package *(SDraw-package)*, 3
halton, 16
halton.coefficients, 18
halton.frame, 18, *22*
halton.indices, *17*, *19*, 20, *23*, *25*, *33*, *35*
halton.indices.CRT, 22, *25*
halton.indices.vector, *23*, 24
halton.lattice, 26, *27*, *29*
halton.lattice.polygon, 28
HI.coast, 30
hip.lattice.polygon, 30
hip.plot.lattice, 32
hip.point, *3*, *14*, *22*, *31*, 32, *35*, *43*
hip.polygon, *3*, *16*, *26–29*, *31*, *34*, 34, *43*, *48*

lineLength, 36

maxU, *5*, *7, 8*, 37
memory.limit, *21*

plotLattice, 38
plotSample, 39
polygonArea, 40
primes, 41

SDraw, *34, 35*
SDraw *(SDraw-package)*, 3
sdraw, *3*, *5*, *9*, *14*, *16*, *40*, *45, 46*, *48, 49*, *51*, *54*

sdraw *(*sdraw.SpatialLines*)*, 42
sdraw,SpatialLines-method
*(*sdraw.SpatialLines*)*, 42
sdraw,SpatialPoints-method
*(*sdraw.SpatialLines*)*, 42
sdraw,SpatialPolygons-method
*(*sdraw.SpatialLines*)*, 42
SDraw-package, 3
sdraw.SpatialLines, 42
sdraw.SpatialPoints
*(*sdraw.SpatialLines*)*, 42
sdraw.SpatialPolygons
*(*sdraw.SpatialLines*)*, 42
spsample, *7*, *12*
SRS *(*SDraw-package*)*, 3
srs.line, 44, *46*
srs.point, *45*, 45
srs.polygon, *45*, *46*, 47
SSS-package *(*SDraw-package*)*, 3
sss.line, *3*, *43*, 48, *51*
sss.point, *43*, *49*, 50
sss.polygon, *3*, *43*, *48*, *49*, *51*, 52

voronoi.polygons, 55

WA, 56
WA.cities, 57
WY, 59