The splancs Package

August 25, 2007

Version 2.01-23

Date 2007-08-23

Title Spatial and Space-Time Point Pattern Analysis

Author Barry Rowlingson <b.rowlingson@lancaster.ac.uk> and Peter Diggle, adapted and packaged for R by Roger Bivand, pcp functions by

Maintainer Roger Bivand <Roger.Bivand@nhh.no>

Depends R (>= 2.2.0), sp (>= 0.8-14)

Description Spatial and Space-Time Point Pattern Analysis Functions

License GPL version 2 or newer (see README).

URL http://www.maths.lancs.ac.uk/~rowlings/Splancs/

R topics documented:

Fhat ................................................................. 3
Fzero .............................................................. 4
Ghat ................................................................. 5
Kenv.csr .......................................................... 6
Kenv.label ....................................................... 7
Kenv.pcp ......................................................... 8
Kenv.tor ........................................................ 9
Kenv.tor1 ....................................................... 11
addpoints ....................................................... 12
amacrines ...................................................... 13
areapl ............................................................ 14
as.points ....................................................... 14
bboxx ............................................................. 15
bodmin ........................................................... 16
burkitt ........................................................... 16
cardiff ........................................................... 18
csr ................................................................. 18
<table>
<thead>
<tr>
<th>R topics documented:</th>
</tr>
</thead>
<tbody>
<tr>
<td>delpoints</td>
</tr>
<tr>
<td>dsquare</td>
</tr>
<tr>
<td>gen</td>
</tr>
<tr>
<td>getpoly</td>
</tr>
<tr>
<td>gridpts</td>
</tr>
<tr>
<td>inout</td>
</tr>
<tr>
<td>inpip</td>
</tr>
<tr>
<td>is.points</td>
</tr>
<tr>
<td>k12hat</td>
</tr>
<tr>
<td>kernel2d</td>
</tr>
<tr>
<td>kernel3d</td>
</tr>
<tr>
<td>kernrat</td>
</tr>
<tr>
<td>kerview</td>
</tr>
<tr>
<td>khat</td>
</tr>
<tr>
<td>khvc</td>
</tr>
<tr>
<td>khvmat</td>
</tr>
<tr>
<td>mpoint</td>
</tr>
<tr>
<td>mse2d</td>
</tr>
<tr>
<td>n2dist</td>
</tr>
<tr>
<td>nndistF</td>
</tr>
<tr>
<td>nndistG</td>
</tr>
<tr>
<td>npts</td>
</tr>
<tr>
<td>okblack</td>
</tr>
<tr>
<td>okwhite</td>
</tr>
<tr>
<td>pcp</td>
</tr>
<tr>
<td>pcp.sim</td>
</tr>
<tr>
<td>pdense</td>
</tr>
<tr>
<td>pip</td>
</tr>
<tr>
<td>plt</td>
</tr>
<tr>
<td>pointmap</td>
</tr>
<tr>
<td>polymap</td>
</tr>
<tr>
<td>print.ribfit</td>
</tr>
<tr>
<td>ranpts</td>
</tr>
<tr>
<td>rLabel</td>
</tr>
<tr>
<td>rtor.shift</td>
</tr>
<tr>
<td>sbox</td>
</tr>
<tr>
<td>secal</td>
</tr>
<tr>
<td>Shift</td>
</tr>
<tr>
<td>southlanes</td>
</tr>
<tr>
<td>splancs</td>
</tr>
<tr>
<td>spoints</td>
</tr>
<tr>
<td>stdiagn</td>
</tr>
<tr>
<td>stkhfat</td>
</tr>
<tr>
<td>stmctest</td>
</tr>
<tr>
<td>stsecal</td>
</tr>
<tr>
<td>stvmat</td>
</tr>
<tr>
<td>thin</td>
</tr>
<tr>
<td>tor.shift</td>
</tr>
</tbody>
</table>
Description

Calculates an estimate of the F nearest neighbour distribution function

Usage

Fhat(pts1,pts2,s)

Arguments

pts1 A points data set
pts2 A points data set
s A vector of distances at which to evaluate Fhat

Details

The function $Fhat(pts1,pts2,s)$ is defined as the proportion of members of a point set $pts2$ for which the distance to the nearest member of another points set $pts1$ is less than or equal to $s$.

Value

A vector of the same length as $s$, containing the value of Fhat at the distances in $s$.

References


See Also

nndistF,Fzero
**Fzero**

*Theoretical nearest neighbour distribution function*

**Description**

Calculate the theoretical nearest neighbour distribution function.

**Usage**

`Fzero(density, s)`

**Arguments**

- `density` The density of the point pattern, i.e. the number of points per unit area.
- `s` A vector of distances at which to evaluate Fzero

**Details**

Fzero returns the nearest neighbour distribution for a homogeneous planar Poisson process. In fortran notation, Fzero(s) is $FZERO = 1 - \exp(-\pi \cdot DENSITY \cdot (S^2))$.

**Value**

A vector of the same length as `s`, containing the value of Fzero at the distances in `s`.

**References**


**See Also**

`Fhat, Ghat, pdense`
**Ghat**

*G nearest neighbour distribution function*

**Description**

Calculates an estimate of the G nearest neighbour distribution function.

**Usage**

```r
Ghat(pts, s)
```

**Arguments**

- **pts** A points data set
- **s** A vector of distances at which to evaluate the G function

**Details**

The function `Ghat(pts, s)` is defined as the proportion of members of a point set for which the distance to the nearest other member of the set is less than or equal to `s`.

**Value**

A vector of the same length as `s`, containing the estimate of G at the distances in `s`.

**References**


**See Also**

`Fhat`, `nndistG`

**Examples**

```r
data(uganda)
plot(seq(20, 500, 20), Ghat(as.points(uganda), seq(20, 500, 20)), type="l", xlab="distance", ylab="Estimated G")
```

```r
plot(seq(20, 500, 20), Ghat(as.points(uganda), seq(20, 500, 20)), type="l", ylab="Theoretical G", xlab="Estimated G")
```
Envelopes of $K$ from simulations of complete spatial randomness

**Description**

Compute envelope of $K$ from simulations of complete spatial randomness.

**Usage**

```r
Kenv.csr(nptg, poly, nsim, s, quiet=FALSE)
```

**Arguments**

- `nptg`: Number of points to generate in each simulation.
- `poly`: Polygon in which to generate the points.
- `nsim`: Number of simulations to do.
- `s`: Vector of distances at which to calculate the envelope.
- `quiet`: If FALSE, print a message after every simulation for progress monitoring. If TRUE, print no messages.

**Value**

A list with two components, called `$upper` and `$lower`. Each component is a vector like `s`. The two components contain the upper and lower bound of the $K$ envelope.

**References**


**See Also**

csr, khat

**Examples**

```r
data(cardiff)
UL.khat <- Kenv.csr(length(cardiff$x), cardiff$poly, nsim=29, seq(2,30,2))
plot(seq(2,30,2), sqrt(khat(as.points(cardiff), cardiff$poly, seq(2,30,2))/pi)-seq(2,30,2), type="l", xlab="Splancs - polygon boundary", ylab="Estimated L", ylim=c(-1,1.5))
lines(seq(2,30,2), sqrt(UL.khat$upper/pi)-seq(2,30,2), lty=2)
lines(seq(2,30,2), sqrt(UL.khat$lower/pi)-seq(2,30,2), lty=2)
```
Kenv.label

Envelope of \( \hat{K}_1 - \hat{K}_2 \) from random labelling of two point patterns

Description

Compute envelope of \( \hat{K}_1 - \hat{K}_2 \) from random labelling of two point patterns

Usage

\[
\text{Kenv.label}(\text{pts1}, \text{pts2}, \text{poly}, \text{nsim}, \text{s}, \text{quiet} = \text{FALSE})
\]

Arguments

- `pts1`: First point data set.
- `pts2`: Second point data set.
- `poly`: Polygon containing the points.
- `nsim`: Number of random labellings to do.
- `s`: Vector of distances at which to calculate the envelope.
- `quiet`: If FALSE, print a message after every simulation for progress monitoring. If TRUE, print no messages.

Details

The two point data sets are randomly labelled using `rLabel`, then \( \hat{K} \) is called to estimate the K-function for each resulting set at the distances in \( s \). The difference between these two estimates is then calculated. The maximum and minimum values of this difference at each distance, over the \( n_{\text{lab}} \) labellings is returned.

Value

A list with two components, called \$upper and \$lower. Each component is a vector like \( s \).

References


See Also

`rLabel`, `ikhat`
Examples

```r
data(okwhite)
data(okblack)
okpoly <- list(x=c(okwhite$x, okblack$x), y=c(okwhite$y, okblack$y))
K1.hat <- khat(as.points(okwhite), bboxx(bbox(as.points(okpoly))), seq(5,80,5))
K2.hat <- khat(as.points(okblack), bboxx(bbox(as.points(okpoly))), seq(5,80,5))
K.diff <- K1.hat-K2.hat
plot(seq(5,80,5), K.diff, xlab="distance", ylab=expression(hat(K)[1]-hat(K)[2]),
ylim=c(-11000,7000), type="l", main="Simulation envelopes, random labelling")
env.lab <- Kenv.label(as.points(okwhite), as.points(okblack),
bboxx(bbox(as.points(okpoly))), nsim=29, s=seq(5,80,5))
lines(seq(5,80,5), env.lab$upper, lty=2)
lines(seq(5,80,5), env.lab$lower, lty=2)
```

Kenv.pcp

*Calculate simulation envelope for a Poisson Cluster Process*

Description

This function computes the envelope of Khat from simulations of a Poisson Cluster Process for a given polygon

Usage

```r
Kenv.pcp(rho, m, s2, region.poly, larger.region=NULL, nsim, r, vectorise.loop=TRUE)
```

Arguments

- `rho` intensity of the parent process
- `m` average number of offsprings per parent
- `s2` variance of location of offsprings relative to their parent
- `region.poly` a polygon defining the region in which the process is to be generated
- `larger.region` a rectangle containing the region of interest given in the form (xl,xu,yl,yu), defaults to `sbox()` around `region.poly`
- `nsim` number of simulations required
- `r` vector of distances at which the K function has to be estimated
- `vectorise.loop` if TRUE, use new vectorised code, if FALSE, use loop as before

Value

- `ave` mean of simulations
- `upper` upper bound of envelope
- `lower` lower bound of envelope
Kenv.tor

Author(s)

Giovanni Petris <GPetris@uark.edu>, Roger.Bivand@nhn.no

References


See Also

pcp, pcp.sim, khat

Examples

data(cardiff)
polymap(cardiff$poly)
pointmap(as.points(cardiff), add=TRUE)
title("Locations of homes of 168 juvenile offenders")
pcp.fit <- pcp(as.points(cardiff), cardiff$poly, h0=30, n.int=30)

m <- npts(as.points(cardiff))/(areapl(cardiff$poly)*pcp.fit$par[2])
r <- seq(2,30,by=2)
K.env <- Kenv.pcp(pcp.fit$par[2], m, pcp.fit$par[1], cardiff$poly,
                  nsim=20, r=r)
L.env <- lapply(K.env, FUN=function(x) sqrt(x/pi)-r)
limits <- range(unlist(L.env))
plot(r, sqrt(khat(as.points(cardiff),cardiff$poly,r)/pi)-r, ylim=limits,
     main="L function with simulation envelopes and average", type="l",
     xlab="distance", ylab="")
lines(r, L.env$lower, lty=5)
lines(r, L.env$upper, lty=5)
lines(r, L.env$ave, lty=6)
abline(h=0)

Kenv.tor

Envelope of K12hat from random toroidal shifts of two point patterns

Description

Compute envelope of K12hat from random toroidal shifts of two point patterns.

Usage

Kenv.tor(pts1,pts2,poly,nsim,s,quiet=FALSE)
Arguments

- `pts1`: First point data set.
- `pts2`: Second point data set.
- `poly`: Polygon containing the points.
- `nsim`: Number of random toroidal shifts to do.
- `s`: Vector of distances at which to calculate the envelope.
- `quiet`: If `FALSE`, print a message after every simulation for progress monitoring. If `true`, print no messages.

Details

The second point data set is randomly shifted using `rtor.shift` in the rectangle defined by `poly`. Then `k12hat` is called to compute \( K_{12}\hat{} \) for the two patterns. The upper and lower values of \( K_{12}\hat{} \) over the \( n_{tor} \) toroidal shifts are returned.

Value

A list with two components, called `$upper` and `$lower`. Each component is a vector like `s`.

References


See Also

- `rtor.shift`, `k12hat`

Examples

```r
data(okwhite)
data(okblack)
okpoly <- list(x=c(okwhite$x, okblack$x), y=c(okwhite$y, okblack$y))
plot(seq(5,80,5), sqrt(k12hat(as.points(okwhite), as.points(okblack), bboxx(bbox(as.points(okpoly))), seq(5,80,5))/pi) - seq(5,80,5), xlab="distance", ylab=expression(hat(L)[12]), ylim=c(-35,35), type="l", main="Simulation envelopes, random toroidal shifts")
env.ok <- Kenv.tor(as.points(okwhite), as.points(okblack), bboxx(bbox(as.points(okpoly))), nsim=29, s=seq(5,80,5))
lines(seq(5,80,5), sqrt(env.ok$upper/pi)-seq(5,80,5), lty=2)
lines(seq(5,80,5), sqrt(env.ok$lower/pi)-seq(5,80,5), lty=2)
```
**Kenv.tor1**

Modified envelope of \( K^{12hat} \) from random toroidal shifts of two point patterns

---

**Description**

Modification of \( \text{Kenv.tor}() \) to allow the assignment of a \( p \) value to the goodness of fit, following the method outlined in Peter Diggle’s 1986 paper (J Neurosci methods 18:115-125) and in his 2002 book.

**Usage**

\[
\text{Kenv.tor1}(\text{pts1}, \text{pts2}, \text{poly}, \text{nsim}, s, \text{quiet} = \text{FALSE})
\]

**Arguments**

- **pts1**: First point data set
- **pts2**: Second point data set
- **poly**: Polygon containing the points
- **nsim**: Number of random toroidal shifts to do
- **s**: Vector of distances at which to calculate the envelope
- **quiet**: If FALSE, print a message after every simulation for progress monitoring. If TRUE, print no messages

**Value**

A list with components: \$upper, \$lower, \$real, u, \$ksim, and \$rank. The first three components are vectors like \( s \), the next two contain results passed back from the simulations, and the final is a one-element vector with the rank of the observed data set.

**Author(s)**

Stephen Eglen <stephen@inf.ed.ac.uk>

**See Also**

Kenv.tor

**Examples**

```r
data(amacrines)
ama.a <- rbind(amacrines.on, amacrines.off)
ama.bb <- bboxx(bbox(as.points(ama.a)))
ama.t <- seq(from = 0.002, to=.250, by=0.002)
nsim=999
plot(amacrines.on, asp=1, pch=19,
    main="Data set, match figure 1.4 of Diggle(2002)?")
```
points(amacrines.off, pch=1)
#
k12 <- k12hat(amacrines.on, amacrines.off, ama.bb, ama.t)
#
k11 <- khat(amacrines.on, ama.bb, ama.t)
k22 <- khat(amacrines.off, ama.bb, ama.t)
k00 <- khat(ama.a, ama.bb, ama.t)
theor <- pi * (ama.t^2)
#
plot(ama.t, k12-theor, ylim=c(min( c(k12, k11, k22, k00) - theor),
  max( c(k12, k11, k22, k00) - theor)),
  main="2nd order properties, match figure 4.8 of Diggle (2002)", type="l")
lines(ama.t, -theor)
lines(ama.t, k11-theor, lty=2)
lines(ama.t, k22-theor, lty=3)
lines(ama.t, k00-theor, lty=5)
#
k12.tor <- Kenv.tor(amacrines.on, amacrines.off, ama.bb,
  nsim, ama.t, quiet=TRUE)
plot(ama.t, k12-theor, type="l", main="Output from Kenv.tor")
lines(ama.t, k12.tor$upper-theor, type="l", col="red")
lines(ama.t, k12.tor$lower-theor, type="l", col="red")
#
k12.sims <- Kenv.tor1(amacrines.on, amacrines.off, ama.bb,
  nsim, ama.t, quiet=TRUE)
plot(ama.t, sqrt(k12.sims$real/pi), type="l", asp=1, bty="n",
  main=paste("K12 versus toroidal sims; rank ", k12.sims$rank, "of",
    length(k12.sims$s))
lines(ama.t, sqrt(k12.sims$upper/pi), col="red")
lines(ama.t, sqrt(k12.sims$lower/pi), col="red")

---

addpoints

Add points interactively to a point data set

Description

Add points interactively to a point data set.

Usage

addpoints(pts, plot=FALSE, quiet=FALSE)

Arguments

pts         A points data set.
plot       if true, plot the pts data, using pointmap. If false, or if pts
            is missing, don’t plot the data.
quiet      if true, don’t print a prompt to enter points.
**Details**

The points entered are displayed on the current graphics device.

**Value**

A points data set consisting of `pts` and the points entered on the current graphics device.

**References**


**See Also**

delpoints

---

<table>
<thead>
<tr>
<th>amacrines</th>
<th>Amacrines on/off data set</th>
</tr>
</thead>
</table>

**Description**

Two two-column matrices of points marked on and off

**Usage**

data(amacrines)

**Format**

Two two-column matrices of points marked on and off

**Source**

areapl  

Calculate area of polygon

Description

Calculate area of polygon. If the polygon is self-intersecting, the area will not be correct.

Usage

areapl(poly)

Arguments

poly  

a polygon data set

Value

The area of the polygon is returned

References


Examples

x <- c(1,0,0,1,1,1,3,3,1)
y <- c(0,0,1,1,0,0,-1,-1,0,0)
m <- cbind(x, y)
plot(m, type="b")
areapl(m)
areapl(m[1:5,])
areapl(m[6:10,])

as.points  

Creates data in spatial point format

Description

Creates data in spatial point format.

Usage

as.points(...)
bboxx

Arguments

... any object(s), such as x and y vectors of the same length, or a list or data frame containing x and y vectors. Valid options for ... are: a points object; returns it unaltered; a list with x and y elements of the same length — returns a points object with the x and y elements as the coordinates of the points; two vectors of equal length; returns a points object with the first vector as the x coordinates, the second vector as the y-coordinates.

Value

as.points tries to return the argument(s) as a points object.

References


bboxx

Generate a non-closed bounding polygon

Description

Generate a non-closed bounding polygon from the bounding box of an object

Usage

bboxx(obj)

Arguments

obj An object: either a Spatial object or a points data set

Value

A points data set of four points giving the non-closed coordinates of the bounding box

References

See Also

sbox

---

bodmin  
*Bodmin Moors granite tors*

**Description**

Locations of 35 granite tors on Bodmin Moor, taken from Infomap data set (northings multiplied by -1 to correspond to Figure 3.2, p. 82, Bailey and Gatrell.

**Usage**

```r
data(bodmin)
```

**Format**

A list corresponding to a Venables and Ripley point object with 35 observations

- `x` numeric grid eastings
- `y` numeric grid northings
- `area` list bounding box with `xl`, `xu`, `yl`, `yu`
- `poly` array polygon boundary with columns `x` and `y`

**Source**


**References**


---

burkitt  
*Burkitt’s lymphoma in Uganda*

**Description**

Locations of cases of Burkitt’s lymphoma in the Western Nile district of Uganda 1960-1975. The time variable is recorded as the number of days starting from an origin of 1 Jan 1960. The examples given below show how the `chron()` function and derived time structures may be used to analyse the data in the time dimension.

**Usage**

```r
data(burkitt)
```
The data is provided as a data table:

- **x**: numeric, grid eastings
- **y**: numeric, grid northings
- **t**: numeric, day number starting at 1/1/1960 of onset
- **age**: numeric, age of child patient
- **dates**: factor, day as string yy-mm-dd

as a points object `burpts` of `burkitt$x` and `burkitt$y`; and a point object of the area boundary `burbdy`.

**Source**


**References**


**Examples**

```r
data(burkitt)
burDates <- as.Date(as.character(burkitt$dates), "%y-%m-%d")
res <- aggregate(rep(1, length(burDates)), list(quarters(burDates), format(burDates, "%y")), sum)
plot(as.numeric(as.character(res$Group.2))+0.25*(as.numeric(substr(as.character(res$Group.1), 2, 2))-1), res$x, type="h", lwd=3, col=ifelse(as.character(res$Group.1)="Q3","grey","red"), xlab="year", ylab="count", xaxt="n")
axis(1, at=seq(61,75,4), labels=format(seq.Date(as.Date("1961/1/1"), as.Date("1975/1/1")), "%y"))
title("Plot of Burkitt's lymphoma in West Nile district, Q3 grey shaded")
op <- par(mfrow=c(3,5))
for (i in unique(format(burDates, "%y"))) {
  polymap(burbdy)
  pointmap(burpts[which(format(burDates, "%y") == i),], add=TRUE, pch=19)
  title(main=paste("19", i, sep=""))
}
par(op)
op <- par(mfrow=c(2,2))
for (i in c("Q1", "Q2", "Q3", "Q4")) {
  polymap(burbdy)
  pointmap(burpts[which(unclass(quarters(burDates)) == i),], add=TRUE, pch=19)
  title(main=i)
}
par(op)
op <- par(mfrow=c(3,4))
for (i in months(seq(as.Date("70-01-01", "%y-%m-%d"), len=12, by="1 month"))) {
  polymap(burbdy)
  pointmap(burpts[which(unclass(months(burDates)) == i),], add=TRUE, pch=19)
  title(main=i)
}
par(op)
```
cardiff  
Locations of homes of juvenile offenders

Description

Locations of homes of 168 juvenile offenders on a Cardiff housing estate

Usage

data(cardiff)

Format

A list corresponding to a Venables and Ripley point object with 168 observations

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>numeric</td>
<td>grid eastings</td>
</tr>
<tr>
<td>y</td>
<td>numeric</td>
<td>grid northings</td>
</tr>
<tr>
<td>area</td>
<td>list</td>
<td>bounding box with xl, xu, yl, yu</td>
</tr>
<tr>
<td>poly</td>
<td>array</td>
<td>polygon boundary with columns x and y</td>
</tr>
</tbody>
</table>

Source


References


csr  
Generate completely spatially random points on a polygon

Description

Generate completely spatially random points on a polygon.

Usage

csr(poly, npoints)

Arguments

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>poly</td>
<td>A polygon data set.</td>
</tr>
<tr>
<td>npoints</td>
<td>The number of points to generate.</td>
</tr>
</tbody>
</table>
Details

csr generates points randomly in the bounding box of poly, then uses pip to extract those in the polygon. If the number of points remaining is less than that required, csr generates some more points in the bounding box until at least npoints remain inside the polygon. If too many points are generated then the list of points is truncated.

Uses runif() to generate random numbers and so updates .Random.seed, the standard S random number generator seed.

Value

A point data set consisting of npoints points distributed randomly, i.e. as an independent random sample from the uniform distribution in the polygon defined by poly.

References


Examples

data(cardiff)
sim <- 29
emp.Ghat <- Ghat(as.points(cardiff), seq(0,30,1))
av.Ghat <- numeric(length(emp.Ghat))
U.Ghat <- numeric(length(emp.Ghat))
L.Ghat <- numeric(length(emp.Ghat))
U.Ghat <- -99999
L.Ghat <- 99999
for(i in 1:sim) {
  S.Ghat <- Ghat(csr(cardiff$poly, length(cardiff$x)), seq(0,30,1))
av.Ghat <- av.Ghat + S.Ghat
L.Ghat <- pmin(S.Ghat, L.Ghat)
U.Ghat <- pmax(S.Ghat, U.Ghat)
}
av.Ghat <- av.Ghat/sim
lines(c(0,1),c(0,1),lty=2)
lines(U.Ghat,emp.Ghat,lty=3)
lines(L.Ghat,emp.Ghat,lty=3)
**delpoints**  
*Select points to delete from a points data set*

**Description**  
Select points to delete from a points data set.

**Usage**  
delpoints(pts, add=FALSE)

**Arguments**  
- **pts** a points data set
- **add** if false, plot the points using `pointmap`.

**Details**  
Using the mouse, the user selects points on the current graphics device. These points are marked on the plot as they are selected. The function returns the remaining points as a points object. If `add` is false the points are plotted on the current plot device.

**Value**  
A points object containing the undeleted points.

**References**  

---

**dsquare**  
*Distance-squared from a number of points to a number of sources*

**Description**  
Computes the distance-squared from a number of points to a number of sources.

**Usage**  
dsquare(pts, srcs, namepref="d")
Arguments

- **pts**: A number of points representing the locations of cases and controls.
- **srcs**: A number of points representing source locations.
- **namepref**: A prefix given to the name of the results.

Value

A data frame with the same number of columns as **srcs**. The column names will be the value of **namepref** prefixing the numbers from 1 to the number of sources.

References


See Also

tribble, triblik

description

generate points in polygon

description

generates random points within a defined polygon, trying to reach **npoints** points - used in **csr**.

Usage

gen(poly, npoints)

Arguments

- **poly**: A polygon data set
- **npoints**: The number of points to generate

Value

returns a point object.
getpoly

Draw a polygon on the current graphics device

Description

Draw a polygon on the current graphics device

Usage

getpoly(quiet=FALSE)

Arguments

quiet if TRUE, don’t prompt for input of a polygon.

Details

The system prompts the user to enter points on the current graphics device using the mouse or other pointing device. The points are joined on the screen with the current line symbol. A polygon of the points entered is drawn on the current graphics device.

Value

A polygon data set consisting of the points entered. The current coordinate system is used.

References

gridpts

Generate a grid of points

Description
Generate a grid of points

Usage
gridpts(poly,npts,xs,ys)

Arguments
- poly: polygon in which to generate the points
- npts: approximate number of points to generate
- xs, ys: grid spacing in x and y

Either npts or xs and ys must be specified. If all three are given then xs and ys are ignored.

Value
A points object containing a grid of points inside the polygon. If npts is specified, then a grid spacing xs and ys will be calculated to give approximately npts in the polygon. If xs and ys are given then these will be used to generate a number of points in the polygon.

References

inout

Test points for inclusion in a polygon

Description
Test points for inclusion in a polygon.

Usage
inout(pts,poly,bound=NULL,quiet=TRUE)
Arguments

pts  A points data set
poly A polygon data set
bound If points fall exactly on polygon boundaries, the default NULL gives arbitrary assignments. If TRUE, then all points "on" boundaries are set as within the polygon, if FALSE, outside.
quiet Do not report which points are on boundary for non-NULL bound

Value

A vector of logical values. TRUE means the point was inside the polygon, FALSE means the point was outside. Note that "inside" is an arbitrary concept for points "on" the polygon boundary.

References


See Also

inpip,pip

Examples

data(uganda)
suganda <- sbox(uganda$poly)
ruganda <- csr(suganda, 1000)
polymap(suganda)
polymap(uganda$poly, add=TRUE)
def <- inout(ruganda, uganda$poly, bound=NULL)
pointmap(as.points(ruganda[def,1], ruganda[def,2]), add=TRUE, col="black")
pointmap(as.points(ruganda[!def,1], ruganda[!def,2]), add=TRUE, col="red")
tru <- inout(ruganda, uganda$poly, bound=TRUE, quiet=FALSE)
which(tru & !def)
ds1 <- as.points(expand.grid(x=seq(-1.5,1.5,0.5), y=seq(-1.5,1.5,0.5)))
d1.poly <- d1[chull(d1),]
ds2 <- as.points(rnorm(300),rnorm(300))
plot(ds2, type="n", asp=1)
polymap(d1.poly, add=TRUE, border="lightblue", col="lightblue", lwd=1)
points(ds2[inout(ds2,d1.poly),], col="green", pch=20)
points(ds2[!inout(ds2,d1.poly),], col="orange", pch=20)
points(ds1[inout(ds1,d1.poly),], col="black", pch=20)
points(ds1[!inout(ds1,d1.poly),], col="red", pch=20)
plot(ds2, type="n", asp=1)
polymap(d1.poly, add=TRUE, border="lightblue", col="lightblue", lwd=1)
points(ds2[inout(ds2,d1.poly,bound=TRUE),], col="green", pch=20)
points(ds2[!inout(ds2,d1.poly,bound=TRUE),], col="orange", pch=20)
inpip

Select points inside a polygon

Description

Select points inside a polygon

Usage

inpip(pts, poly, bound=NULL, quiet=TRUE)

Arguments

pts       A points data set
poly      A polygon data set
bound     If points fall exactly on polygon boundaries, the default NULL gives arbitrary
          assignments. If TRUE, then all points "on" boundaries are set as within the
          polygon, if FALSE, outside.
quiet     Do not report which points are on boundary for non-NUL bound

Value

inpip returns a vector of indices of the points in pts that are located in the polygon. Note that
"in" is an arbitrary concept for points "on" the polygon boundary.

References

Rowlingson, B. and Diggle, P. 1993 Splancs: spatial point pattern analysis code in S-Plus. Com-

See Also

pip
is.points  

**Point Objects**

**Description**

Tests for data in spatial point format.

**Usage**

```r
is.points(p)
```

**Arguments**

- **p** any object.

**Value**

`is.points` returns TRUE if `p` is a points object, FALSE otherwise.

**References**


---

**k12hat**  

**Bivariate K-function**

**Description**

Calculates an estimate of the bivariate K-function

**Usage**

```r
k12hat(pts1, pts2, poly, s)
```

**Arguments**

- **pts1, pts2** Two points data sets
- **poly** A polygon containing the points
- **s** A vector of distances at which to estimate the K12 function
Details

The bivariate K function is defined as the expected number of points of pattern 1 within a distance $s$ of an arbitrary point of pattern 2, divided by the overall density of the points in pattern 1. To estimate this function, the approximately unbiased estimator given by Lotwick and Silverman (1982) is used.

Value

A vector like $s$ containing the value of $K_{12}\hat{\text{hat}}$ at the points in $s$.

References


Examples

data(okwhite)
data(okblack)
okpoly <- list(x=c(okwhite$x, okblack$x), y=c(okwhite$y, okblack$y))
plot(seq(5,80,5), sqrt(12hat(as.points(okwhite), as.points(okblack), bboxx(bbox(as.points(okpoly))), seq(5,80,5))/pi) - seq(5,80,5), xlab="distance", ylab=expression(hat(L)[12]), ylim=c(-20,20), type="l")

kernel2d

Kernel smoothing of a point pattern

Description

Perform kernel smoothing of a point pattern

Usage

kernel2d(pts,poly,h0,nx=20,ny=20,kernel='quartic')
spkernel2d(pts, poly, h0, grd, kernel = "quartic")

Arguments

pts A points data set, or in function spkernel2d an object with a coordinates method from the sp package
poly A splancs polygon data set
h0 The kernel width parameter
nx Number of points along the x-axis of the returned grid.
kernel2d

ny

Number of points along the y-axis of the returned grid.

kernel

Type of kernel function to use. Currently only the quartic kernel is implemented.

grd

a GridTopology object from the sp package

Details

The kernel estimate, with a correction for edge effects, is computed for a grid of points that span the input polygon. The kernel function for points in the grid that are outside the polygon are returned as NA’s. The output list is in a format that can be read into `image()` directly, for display and superposition onto other plots.

Value

`kernel2d` returns a list with the following components:

- `x` List of x-coordinates at which the kernel function has been evaluated.
- `y` List of y-coordinates at which the kernel function has been evaluated.
- `z` A matrix of dimension `nx` by `ny` containing the value of the kernel function.
- `h0, kernel` containing the values input to `kernel2d`

`spkernel2d` returns a numeric vector with the value of the kernel function stored in the order required by `sp` package `SpatialGridDataFrame` objects

References


Examples

data(bodmin)
plot(bodmin$poly, asp=1, type="n")
image(kernel2d(as.points(bodmin), bodmin$poly, h0=2, nx=100, ny=100),
add=TRUE, col=terrain.colors(20))
pointmap(as.points(bodmin), add=TRUE)
polymap(bodmin$poly, add=TRUE)
bodmin.xy <- coordinates(bodmin[1:2])
apply(bodmin$poly, 2, range)
grd1 <- GridTopology(cellcentre.offset=c(-5.2, -11.5), cellsize=c(0.2, 0.2), cells.dim=c(75, 100))
k100 <- spkernel2d(bodmin.xy, bodmin$poly, h0=1, grd1)
k150 <- spkernel2d(bodmin.xy, bodmin$poly, h0=1.5, grd1)
k200 <- spkernel2d(bodmin.xy, bodmin$poly, h0=2, grd1)
k250 <- spkernel2d(bodmin.xy, bodmin$poly, h0=2.5, grd1)
if (.sp_lt_0.9()) {
  df <- AttributeList(list(k100=k100, k150=k150, k200=k200, k250=k250))
} else {
df <- data.frame(k100=k100, k150=k150, k200=k200, k250=k250)
kernels <- SpatialGridDataFrame(grd1, data=df)
plot(kernels, checkEmptyRC=FALSE, col.regions=terrain.colors(16), cuts=15)

kernel3d

Space-time kernel

Description

Compute the space-time kernel

Usage

kernel3d(pts, times, xgr, ygr, zgr, hxy, hz)

Arguments

pts A matrix of event coordinates x,y.
times A vector of event times, t.
xgr The values of x at which to compute the kernel function.
ygr The values of y at which to compute the kernel function.
zgr The values of time at which to compute the kernel function.
hxy The quartic kernel width in the x and y direction.
hz The quartic kernel width in the temporal direction.

Value

A list is returned. Most of the components are just copies of the input parameters, except for the $v$ parameter. This is a three dimensional array containing the kernel-smoothed values. Its dimension is [length(xgr), length(ygr), length(tgr)].

References


See Also

kerview
Examples

```r
data(burkitt)
b3d <- kernel3d(burpts, burkitt$t, seq(250,350,10), seq(250, 400, 10),
                seq(365,5800,365), 30, 200)
brks <- quantile(b3d$v, seq(0,1,0.05))
cols <- heat.colors(length(brks)-1)
oldpar <- par(mfrow=c(3,5))
for (i in 1:15) image(seq(250,350,10), seq(250, 400, 10), b3d$v[,i],
                   asp=1, xlab="", ylab="", main=1960+i, breaks=brks, col=cols)
par(oldpar)
```

kernrat  

**Ratio of two kernel smoothings**

Description

Return the ratio of two kernel smoothings

Usage

```r
kernrat(pts1,pts2,poly,h1,h2,nx=20,ny=20,kernel='quartic')
```

Arguments

- `pts1,pts2`: Point data sets
- `poly`: A polygon data set
- `h1,h2`: The kernel width parameters, `h1` for `pts1`, and `h2` for `pts2`
- `nx`: Number of points along the x-axis of the returned grid.
- `ny`: Number of points along the y-axis of the returned grid.
- `kernel`: Type of kernel function to use. Currently only the quartic kernel is implemented.

Value

A list with the following components:

- `x`: List of x-coordinates at which the kernel function has been evaluated.
- `y`: List of y-coordinates at which the kernel function has been evaluated.
- `z`: A matrix of dimension `nx` by `ny` containing the ratio of the kernel functions.
- `h`: A vector of length 2 containing `h1` and `h2`
- `kernel`: A character string containing the kernel name.
References


See Also

`kernel2d, mse2d`

Examples

```r
data(okwhite)
data(okblack)
okpoly <- list(x=c(okwhite$x, okblack$x), y=c(okwhite$y, okblack$y))
kr <- kernrat(as.points(okwhite), as.points(okblack), bboxx(bbox(as.points(okpoly))),
h1=50, h2=50)
image(kr, asp=1)
brks <- quantile(c(kr$z), seq(0,1,1/10), na.rm=TRUE)
lbrks <- formatC(brks, 3, 6, "g", " ")
cols <- heat.colors(length(brks)-1)
def.par <- par(no.readonly = TRUE)
layout(matrix(c(1,0,1,2), 2, 2, byrow = TRUE), c(2.5,1.5), c(1,3), TRUE)
image(kr, breaks=brks, col=cols, asp=1)
plot.new()
legend(c(0,1), c(0,1), legend=paste(lbrks[-length(lbrks)], lbrks[-1], sep=":"), fill=cols, bty="n")
par(def.par)
```

`kerview`  
*A linked-window system for browsing space-time data*

Description

A linked-window system for browsing space-time data.

Usage

`kerview(pts, times, k3, map=TRUE, addimg=TRUE, ncol=1)`

Arguments

- `pts`  
  A matrix of event x,y coordinates.
- `times`  
  A vector of event times.
- `k3`  
  An object returned from `kernel3d`, the space-time kernel smoothing function.
- `map`  
  If false, don’t plot the map display.
If true, overwrite successive images in the image display, else make a fresh image plot each time.

Number of columns and rows for multiple images and maps.

### Details

This function displays three linked views of the data. In the current graphics window a temporal slice from the kernel smoothing is displayed. Another graphics device is started to display a map of the data that contributed to that time-slice. A third graphics device shows a histogram of the times of the events. Clicking with the mouse in this window with button 1 sets the time for the other displays to the time on the x-axis of the histogram at the clicked point.

In this way the 3-dimensional kernel smoothed function can be browsed, and the corresponding map of the data compared.

### References


### See Also

- kernel3d

---

**khat**

*K-function*

---

**Description**

Calculates an estimate of the K-function

**Usage**

```r
khat(pts,poly,s,newstyle=FALSE,checkpoly=TRUE)  
## S3 method for class 'khat': print(x, ...)  
## S3 method for class 'khat': plot(x, ...)
```

**Arguments**

- `pts` A points data set
- `poly` A polygon containing the points - must be a perimeter ring of points
- `s` A vector of distances at which to calculate the K function
newstyle  if TRUE, the function returns a khat object
checkpoly  if TRUE compare polygon area and polygon bounding box and convex hull areas
to see whether the polygon object is malformed; may be set to FALSE if the
polygon is known to be a ring of points
x  a khat object
...  other arguments passed to plot and print functions

Details

The K function is defined as the expected number of further points within a distance s of an arbitrary
point, divided by the overall density of the points. In practice an edge-correction is required to avoid
biasing the estimation due to non-recording of points outside the polygon.

The newstyle argument and khat object were introduced in collaboration with Thomas de Cor-
nulier to permit the mapping of counts or khats for chosen distance values, as in ftp://pbil.

Value

If newstyle is FALSE, a vector like s containing the value of K at the points in s. else a khat
object list with:

khat  the value of K at the points in s
counts  integer matrix of counts of points within the vector of distances s for each point
khats  matrix of values of K within the vector of distances s for each point
s  s

References

Ripley, B.D. 1976 The second-order analysis of stationary point processes, J. Appl. Prob, 13
255-266; Rowlingson, B. and Diggle, P. 1993 Splancs: spatial point pattern analysis code in S-
Plus. Computers and Geosciences, 19, 627-655; the original sources can be accessed at: http:
//www.maths.lancs.ac.uk/~rowlings/Splancs/ . See also Bivand, R. and Gebhardt,
A. 2000 Implementing functions for spatial statistical analysis using the R language. Journal of

See Also

Kenv.csr

Examples

data(cardiff)
s <- seq(2,30,2)
plot(s, sqrt(khat(as.points(cardiff), cardiff$poly, s)/pi) - s,
type="l", xlab="Splancs - polygon boundary", ylab="Estimated L",
ylim=c(-1,1.5))
newstyle <- khat(as.points(cardiff), cardiff$poly, s, newstyle=TRUE)
str(newstyle)
newstyle
apply(newstyle$khat, 2, sum)
plot(newstyle)

khvc

**Covariance matrix for the difference between two K-functions**

**Description**

Calculate the covariance matrix for the difference between two K-functions. Also return the contribution to the variance for each of the two point patterns.

**Usage**

`khvc(pts1, pts2, poly, s)`

**Arguments**

- `pts1`: An object containing the case locations.
- `pts2`: An object containing the control locations.
- `poly`: A polygon enclosing the locations in `pts1` and `pts2`.
- `s`: A vector of distances at which the calculation is to be made.

**Value**

A list with four components:

- `varmat`: The upper triangle of the covariance matrix.
- `k11`: The variance of Khat for the cases.
- `k22`: The variance of Khat for the controls.
- `k12`: The covariance of Khat for the cases and Khat for controls.

**Note**

Note that the diagonal of the covariance matrix is `k11 - 2 * k12 + k22`.

**References**


**See Also**

`khat`, `khvmat`, `secal`
### khvmat

**Covariance matrix for the difference between two K-functions**

#### Description

Calculate the covariance matrix for the difference between two K-functions under random labelling of the corresponding two sets of points.

#### Usage

```
khvmat(pts1, pts2, poly, s)
```

#### Arguments

- **pts1**: An object containing the case locations.
- **pts2**: An object containing the control locations.
- **poly**: Polygon enclosing the points in pts1 and pts2.
- **s**: A vector of distances at which the calculation is to be made.

#### Value

A matrix containing the covariances, with the variances on the diagonal.

#### References


#### See Also

- [secal](#)
mpoint

Overlay a number of point patterns

Description

Overlay a number of point patterns.

Usage

mpoint(..., cpch, add=FALSE, type="p")

Arguments

... At least one argument consisting of a points data set must be specified.

cpch A vector of characters for plotting symbols

add if add is TRUE then overlay on an existing plot

type plot data as points if type="p", lines if type="l"

Details

mpoint enables several point or polygon datasets to be overlayed. The plot region is calculated so that all the specified datasets fit in the region. The parameter cpch specifies the characters to use for each set of points. The default cpch consists of the numbers 1 to 9 followed by the uppercase letters A to Z. If cpch is shorter than the number of point sets to plot, then it is repeated.

References


See Also

pointmap
 mse2d

### Description

Estimate the Mean Square Error for a Kernel Smoothing.

### Usage

```r
mse2d(pts, poly, nsmse, range)
```

### Arguments

- `pts`: A set of points.
- `poly`: A polygon containing the points.
- `nsmse`: Number of steps of \( h \) at which to calculate the mean square error.
- `range`: Maximum value of \( h \) for calculating the mean square error.

### Value

A list with two components, \( \text{h} \) and \( \text{mse} \). These vectors store corresponding values of the mean square error at values of the kernel smoothing parameter, \( h \). The value of \( h \) corresponding to the minimum value of \( \text{mse} \) can be passed to `kernel2d` as the optimum smoothing parameter.

### References


### See Also

`kernel2d`

### Examples

```r
data(bodmin)
Mse2d <- mse2d(as.points(bodmin), bodmin$poly, nsmse=50, range=8)
plot(Mse2d$h[5:50], Mse2d$mse[5:50], type="l")
```
Description

Calculate nearest neighbours for two point patterns

Usage

n2dist(pts1,pts2)

Arguments

pts1, pts2 Point data sets

Value

Returns a list with components $dists$ and $neighs$. $dists[i]$ is the distance of the nearest neighbour of point pts2[i,] in pts1 and $neighs[i]$ is the index in pts1 of the point nearest to pts2[i,]. Documentation and example by Alun Pope, 2007-08-23.

References


See Also

nndistF, Fhat, Ghat, Fzero

Examples

(test1 <- matrix(c(151.1791, -33.86056, 151.1599, -33.88729, 151.1528, -33.90685, 151.1811, -33.85937), nrow=4, byrow=TRUE))
(test2 <- as.points(151.15, -33.9))
n2dist(test1,test2)
n2dist(test2,test1)
nndistF

Nearest neighbour distances as used by Fhat()

Description

Calculate nearest neighbour distances as used by Fhat().

Usage

nndistF(pts1, pts2)

Arguments

pts1 A points data set
pts2 A points data set

Value

The set of distances from each of the points in pts2 to the nearest point in pts1 is returned as a vector.

References


See Also

nndistG, Fhat, Ghat, Fzero

Examples

data(uganda)
boxplot(nndistF(as.points(uganda), as.points(csr(uganda$poly, length(uganda$x)))))
plot(ecdf(nndistF(as.points(uganda),
               as.points(csr(uganda$poly, length(uganda$x))))),
     main="Fhat ecdf Uganda volcano data")
Description

Calculate nearest neighbour distances as used by Ghat().

Usage

nndistG(pts)

Arguments

pts A points data set

Value

Returns a list with components $dists$ and $neighs$. $dists[i]$ is the distance to the nearest neighbour of point $i$ in $pts$, and $neighs[i]$ is the index of the neighbour of point $i$.

References


See Also

nndistF, Fhat, Ghat, Fzero

Examples

data(uganda)
boxplot(nndistG(as.points(uganda))$dists)
plot(ecdf(nndistG(as.points(uganda))$dists))
### npts

*Number of points in data set*

**Description**

return number of points in data set

**Usage**

\[
npts(pts)\]

**Arguments**

- `pts`: A points data set

**Value**

The number of points in the data set.

**References**


---

### okblack

*Oklahoma black offenders*

**Description**

Locations of theft from property offences committed by black offenders in Oklahoma City

**Usage**

\[
data(okblack)\]

**Format**

A list corresponding to a Venables and Ripley point object with 147 observations

- `x`: numeric grid eastings
- `y`: numeric grid northings
- `area`: list bounding box with `xl`, `xu`, `yl`, `yu`
Source

References

okwhite Oklahoma white offenders

Description
Locations of theft from property offences committed by white offenders in Oklahoma City

Usage
data(okwhite)

Format
A list corresponding to a Venables and Ripley point object with 104 observations

  x numeric grid eastings
  y numeric grid northings
  area list bounding box with xl, xu, yl, yu

Source

References

pcp Fit a Poisson cluster process

Description
The function fits a Poisson cluster process to point data for a given enclosing polygon and fit parameters

Usage
pcp(point.data, poly.data, h0=NULL, expo=0.25, n.int=20)
Arguments

point.data  a points object
poly.data  a polygon enclosing the study region
h0  upper bound of integration in the criterion function
expo  exponent in the criterion function
n.int  number of intervals used to approximate the integral in the criterion function with a sum

Value

The function returns an object as returned by \texttt{optim}, including:

par  The best set of parameters s2 and rho found
value  The value of the fit corresponding to \textquote{par}
convergence  '0' indicates successful convergence

Author(s)

Giovanni Petris <GPetris@uark.edu>, Roger.Bivand@nhh.no

References


See Also

\texttt{optim,pcp.sim,Kenv.pcp,khat}

Examples

data(cardiff)
polymap(cardiff$poly)
pointmap(as.points(cardiff), add=TRUE)
title("Locations of homes of 168 juvenile offenders")
pcp.fit <- pcp(as.points(cardiff), cardiff$poly, h0=30, n.int=30)
pcp.fit
Generate a Poisson Cluster Process

Description

The function generates a Poisson cluster process for a given polygon within a larger bounding region and given process parameters.

Usage

pcp.sim(rho, m, s2, region.poly, larger.region=NULL, vectorise.loop=TRUE)

Arguments

rho
  intensity of the parent process
m
  average number of offsprings per parent
s2
  variance of location of offsprings relative to their parent
region.poly
  a polygon defining the region in which the process is to be generated
larger.region
  a rectangle containing the region of interest given in the form (xl,xu,yl,yu), defaults to sbox() around region.poly
vectorise.loop
  if TRUE, use new vectorised code, if FALSE, use loop as before

Details

The function generates the parents in the larger bounding region, generates their children also in the larger bounding region, and then returns those inside the given polygon.

Value

A point object with the simulated pattern

Author(s)

Giovanni Petris <GPetris@uark.edu>, Roger.Bivand@nhh.no

References


See Also

pcp, Kenv.pcp, khat
pdense

Examples

```r
data(cardiff)
polymap(cardiff$poly)
pointmap(as.points(cardiff), add=TRUE)
title("Locations of homes of 168 juvenile offenders")
pcp.fit <- pcp(as.points(cardiff), cardiff$poly, h0=30, n.int=30)
pcp.fit
m <- npts(as.points(cardiff))/(areapl(cardiff$poly)*pcp.fit$par[2])
sims <- pcp.sim(pcp.fit$par[2], m, pcp.fit$par[1], cardiff$poly)
pointmap(as.points(sims), add=TRUE, col="red")
```

---

**pdense**

*Overall density for a point pattern*

Description

Calculate overall density for a point pattern.

Usage

```r
pdense(pts,poly)
```

Arguments

- **pts**: A points data set
- **poly**: A polygon data set

Value

The density of the points in the polygon. i.e. the number of points per unit area.

References


See Also

Fzero
**pip**

*Points inside or outside a polygon*

**Description**

Return points inside or outside a polygon.

**Usage**

```r
pip(pts, poly, out=FALSE, bound=NULL, quiet=TRUE)
```

**Arguments**

- **pts**: A points data set
- **poly**: A polygon data set
- **out**: If `out=TRUE`, return the points outside the polygon, else the points inside.
- **bound**: If points fall exactly on polygon boundaries, the default NULL gives arbitrary assignments. If TRUE, then all points "on" boundaries are set as within the polygon, if FALSE, outside.
- **quiet**: Do not report which points are on boundary for non-NULL bound

**Details**

`pip` calls `inout`, then selects the appropriate sub-set of points.

**Value**

`pip` returns the points of `pts` that lie inside (or outside with `out=TRUE`) the polygon `poly`. Compare this with `inpip`, which returns the indices of the points in the polygon, and `inout` which returns a logical vector indicating whether points are inside or outside.

**References**


**See Also**

`inpip`, `inout`
**plt**

*bins nearest neighbour distances*

**Description**

bins nearest neighbour distances

**Usage**

plt(data, value)

**Arguments**

data nearest neighbour distances
value breaks for binning distances

**Value**

binned values

**References**


**See Also**

Fhat, Ghat

---

**pointmap**

*Graphics*

**Description**

Plots point and polygon data sets on the current graphics device.

**Usage**

pointmap(pts, add=FALSE, axes=TRUE, xlab="", ylab="", asp, ...)

---
Arguments

- **pts**: a points data set.
- **add**: if FALSE, start a new plot. If TRUE, superimpose on current plot.
- **axes**: if true, display axes with labelling. If false, do not display any axes on the plot.
- **xlab, ylab**: Label strings for x and y axes.
- **asp**: aspect parameter for plot.
- **...**: Graphical arguments may be entered, and these are passed to the standard S `points` and `polygon` routines.

Details

The specified data set is plotted on the current graphics device, either as points or polygons. For `polymap`, the last point in the data set is drawn connected to the first point. `pointmap` and `polymap` preserve the aspect ratio in the data by using the `asp=1` plot argument. Graphical parameters can also be supplied to these routines, and are passed through to `plot`. Some useful parameters include `pch` to change the plotting character for points, `lty` to change the line type for polygons, and `type="n"` to set up axes for the plot without plotting anything.

References


See Also

`mpoint`

Examples

```r
data(bodmin)
plot(bodmin$poly, asp=1, type="n")
pointmap(as.points(bodmin), add=TRUE)
polymap(bodmin$poly, add=TRUE)
```
print.ribfit

Arguments

poly      a polygon.
add       if FALSE, start a new plot. If TRUE, superimpose on current plot.
xlab, ylab Label strings for x and y axes.
axes      if true, display axes with labelling. If false, do not display any axes on the plot.
asp       aspect parameter for plot
...       Graphical arguments may be entered, and these are passed to the standard S points and polygon routines.

Details

The specified data set is plotted on the current graphics device, either as points or polygons. For polymap, the last point in the data set is drawn connected to the first point. pointmap and polymap preserve the aspect ratio in the data by using the asp=1 plot argument. Graphical parameters can also be supplied to these routines, and are passed through to plot. Some useful parameters include pch to change the plotting character for points, lty to change the line type for polygons, and type="n" to just set up axes for the plot without plotting anything.

References


See Also

par, mpoint

Examples

data(bodmin)
plot(bodmin$poly, asp=1, type="n")
pointmap(as.points(bodmin), add=TRUE)
polymap(bodmin$poly, add=TRUE)

print.ribfit          Display the fit from tribble()

Description

Display the fit from tribble
Usage

```r
## S3 method for class 'ribfit':
print(x, ...)
```

Arguments

- `x`: An object returned from `tribble`
- `...`: optional arguments to pass through to `print()`

Details

The parameter estimates and log-likelihood for the raised incidence model are displayed. The likelihood ratio, $D = 2*(L-Lo)$, is also given. This function is called whenever `print` operates on an object with class `ribfit`.

References


See Also

- `tribble`

---

### ranpts

- **adjust number of random points in polygon**

Description

adjust number of random points in polygon

Usage

```r
ranpts(pts, poly, nprq)
```

Arguments

- `pts`: points object
- `poly`: polygon object
- `nprq`: required number of points

Value

points object with required number of random points
rLabel

References


See Also
csr

rLabel

Randomly label two or more point sets

Description

Randomly label two or more point sets. (function name changed from rlabel to rLabel to avoid collision with spatstat)

Usage

rLabel(...)

Arguments

... Any number of points data sets

Details

The output data sets are a random labelling of the input data sets, i.e. all the points in the input data sets are randomly assigned to the output sets. The number of points in each output set is the same as its corresponding input set.

Value

A list of points data sets. There are as many elements in the list as arguments.

References

rtor.shift

Random toroidal shift on a point data set

Description

Perform a random toroidal shift on a point data set

Usage

rtor.shift(pts, rect)

Arguments

pts The point data set to shift
rect A rectangle defining the region for the toroidal map. If not given, the bounding box of pts is used.

Details

The planar region defined by rect is assumed connected at its top and bottom edges, and at its left and right sides. A random shift is applied to the points and the resulting set of points returned.

Value

A point data set like pts, but after application of a random toroidal shift along the x and y axes.

References


See Also

Shift
sbox

Generate a box surrounding a point object

Description

Generate a box surrounding a point object

Usage

sbox(pts, xfrac = .1, yfrac = .1)

Arguments

- **pts**: A points data set
- **xfrac**: The fraction of the width of the point pattern by which the box will surround the point pattern to the left and right.
- **yfrac**: The fraction of the height of the point pattern by which the box will surround the point pattern to the top and bottom.

Value

A points data set of four points giving the coordinates of the surrounding box

References


See Also

bboxx

secal

Standard errors for the difference between two K-functions

Description

Calculate standard errors for the difference between two K-functions under random labelling of the corresponding two sets of points.

Usage

secal(pts1, pts2, poly, s)
Arguments

pts1, pts2  Two point data sets
poly  Polygon enclosing the points in pts1 and pts2
s  A vector of distances at which to calculate the standard error.

Details

To compare two point patterns, one can calculate the difference between their K-functions. The function seCal gives the pointwise standard errors for the estimated differences, under the random labelling hypothesis.

Value

A vector like s containing the value of the standard error at each of the distances in s

References


See Also

khat

Shift

Shift a point data set

Description

Shift a point data set (function name changed from shift to Shift to avoid collision with spatstat)

Usage

Shift(pts, xsh=0.0, ysh=0.0)

Arguments

pts  The point data set to shift
xsh  Amount to shift along the x-axis
ysh  Amount to shift along the y-axis
A point data set like `pts`, but with `xsh` added to its x-coordinates, and `ysh` added to its y-coordinates.

**References**


**See Also**

tor.shift

---

*Cancer cases in Chorley-Ribble*

**Description**

Locations of cases of cancer of lung and larynx in Chorley-Ribble, Lancashire. The data set is split into a points object `southlancs.pts` and a case/control 0/1 vector `southlancs.cc`. There are 917 controls and 57 cases in this data set - these numbers differ from 978 and 58 in Diggle (1990) and Diggle and Rowlingson (1994). The data set also includes the approximate location of an old incinerator `old.incinerator`, as well as `southlancs.bdy`, the study area boundary.

**Usage**

data(southlancs)

**Format**

A data frame with 974 observations

- `x` numeric grid eastings (metres)
- `y` numeric grid northings (metres)
- `cc` numeric case/control, lung=0, larynx=1

**Source**


**References**


Examples

data(southlancs)
opt <- par(mfrow=c(2,1))
pointmap(southlancs.pts[southlancs.cc == 0,])
pointmap(old.incinerator, add=TRUE, col="red", pch=19)
title("Lung cancer controls")
pointmap(southlancs.pts[southlancs.cc == 1,])
pointmap(old.incinerator, add=TRUE, col="red", pch=19)
title("Larynx cancer cases")
par(op)
polymap(southlancs.bdy,border="grey")
contour(kernel2d(southlancs.pts[southlancs.cc == 0],
    southlancs.bdy, h=500, nx=100, ny=100), nlevels=20,
    add=TRUE,drawlabels=FALSE)
pointmap(southlancs.pts[southlancs.cc == 1], add=TRUE, pch=19,
    col="green")
pointmap(old.incinerator, add=TRUE, pch=19, col="red")
title(xlab="h=500, quartic kernel")
title("Density map of control, green case points, red old incinerator")

splanccoli

Return version number and author information

Description

Return version number and author information

Usage

splanccoli()

Value

The version string is returned. This is a number of the format x.yy, where x is the major version number and yy is the minor version number.

References

**spoints**

*Point Objects*

**Description**

Creates and tests for data in spatial point format.

**Usage**

```r
spoints(data, npoints)
```

**Arguments**

- `data`: vector containing the data values for the points in order (x1,y1),(x2,y2),..., if missing, set to length(data)/2.
- `npoints`: number of points to generate, if missing, set to length(data)/2.

**Value**

`spoints` returns an object suitable for use as a point data object. If `npoints` is given, the vector data is either truncated or repeated until sufficient data values are generated. The returned object is a two-column matrix, where the first column stores the x-coordinate, and the second column stores the y-coordinate.

**References**


**See Also**

- `npts`

---

**stdiagn**

*Summary plots for clustering analysis*

**Description**

Produces some summary plots for clustering analysis

**Usage**

```r
stdiagn(pts, stkh, stse, stmc=0, Dzero=FALSE)
```
Arguments

- **pts**: A set of points, as used in Splancs
- **stkh**: An object returned from `stkhat`
- **stse**: An object returned from `stsecal`
- **stmcd**: An object returned from `stmctest`
- **Dzero**: FALSE - default D plot, TRUE Dzero plot

Details

Four plots are produced on the current graphics device. The first plot is simply a map of the data. The second is a perspective plot of the difference between space-time K-function and the product of spatial and temporal K-functions. The third plot is of the standardised residuals against the product of spatial and temporal K-functions. If the Monte-Carlo data is given the fourth plot is a histogram of the test statistics, with the value for the data indicated with a vertical line. See Diggle, Chetwynd, Haggkvist, and Morris (1995) for details.

References


See Also

- `stkhat`, `stsecal`, `stvmat`, `stmctest`

Examples

```r
example(stkhat)
example(stsecal)
example(stmctest)
stdiagn(burpts, bur1, bur1se, bur1mc)
```

Description

Compute the space-time K-functions

Usage

```
stkhat(pts, times, poly, tlimits, s, tm)
```
Arguments

pts  A set of points as defined in Splancs

times  A vector of times, the same length as the number of points in pts

poly  A polygon enclosing the points

tlimits  A vector of length 2 specifying the upper and lower temporal domain.

s  A vector of spatial distances for the analysis.

tm  A vector of times for the analysis

Value

A list with the following components is returned:

s, t  The spatial and temporal scales

ks  The spatial K-function

kt  The temporal K-function

kst  The space-time K-function

For details see Diggle, Chetwynd, Hagggkvist, and Morris (1995)

References


See Also

stsecal, stvmat, stmctest, stdiagn

Examples

data(burkitt)
burl <- stkhat(burpts, burkitt$t, burbdy, c(400, 5800),
    seq(1,40,2), seq(100, 1500, 100))
oldpar <- par(mfrow=c(2,1))
plot(burl$s, burl$ks, type="l", xlab="distance", ylab="Estimated K",
    main="spatial K function")
plot(burl$t, burl$kt, type="l", xlab="time", ylab="Estimated K",
    main="temporal K function")
par(oldpar)
stmctest          Monte-Carlo test of space-time clustering

Description

Perform a Monte-Carlo test of space-time clustering.

Usage

stmctest(pts, times, poly, tlimits, s, tt, nsim, quiet=FALSE)

Arguments

pts          A set of points as used by Splancs.
times        A vector of times, the same length as the number of points in pts.
poly         A polygon enclosing the points.
tlimits      A vector of length 2, specifying the upper and lower temporal domain.
s            A vector of spatial distances for the analysis.
tt            A vector of times for the analysis.
nsim          The number of simulations to do.
quiet         If quiet=TRUE then no output is produced, otherwise the function prints the number of simulations completed so far, and also how the test statistic for the data ranks with the simulations.

Details

The function uses a sum of residuals as a test statistic, randomly permutes the times of the set of points and recomputes the test statistic for a number of simulations. See Diggle, Chetwynd, Haggkvist and Morris (1995) for details.

Value

A vector of length nsim+1. The first element is the test statistic for the data, and the remaining elements are those for the simulations.

References

stsecal

See Also

stkhat, stsecal, stvmat, stdiagn

Examples

e.example(stkhat)
bur1mc <- stmctest(burpts, burkitt$t, burbdy, c(400, 5800),
                   seq(1, 40, 2), seq(100, 1500, 100), nsim=49, quiet=TRUE)

stsecal

Standard error for space-time clustering

Description

Computes the standard error for space-time clustering.

Usage

stsecal(pts, times, poly, tlim, s, tm)

Arguments

pts A set of points, as defined in Splancs.
times A vector of times, the same length as the number of points in pts
poly A polygon enclosing the points
tlim A vector of length 2 specifying the upper and lower temporal domain.
s A vector of spatial distances for the analysis
tm A vector of times for the analysis

Value

A matrix of dimension \[ \text{length(s), length(t)} \] is returned. Element \[ i, j \] is the standard error at \[ s[i], t[j] \]. See Diggle Chetwynd Haggkvist and Morris (1995) for details.

References


See Also

stkhat, stsecal, stvmat, stdiagn
Examples

```r
example(stkhat)
burlse <- stsecal(burpts, burkitt$t, burbdy, c(400, 5800),
seq(1,40,2), seq(100, 1500, 100))
```

---

**stvmat**

*Variance matrix for space-time clustering*

### Description

Compute the variance matrix for space-time clustering

### Usage

```r
stvmat(pts, times, poly, tlim, s, tm)
```

### Arguments

- **pts**: A set of points.
- **times**: A vector of times, the same length as the number of points in `pts`.
- **poly**: A polygon that encloses the points.
- **tlim**: A vector of length 2 specifying the upper and lower temporal domain.
- **s**: A vector of spatial distances for the analysis.
- **tm**: A vector of times for the analysis.

### Value

A four-dimensional matrix is returned. The covariance between space-time \( t_1, s_1 \) and \( t_2, s_2 \) is given by the corresponding element \([t_1, s_1, t_2, s_2]\). For full details, see Diggle, Chetwynd, Haggkvist and Morris (1995).

### References


### See Also

`stkhat`, `stsecal`, `stmctest`, `stdiagn`
thin

Randomly thin a point data set

Description
Randomly thin a point data set.

Usage
thin(pts,n)

Arguments
pts a points data set.
n the number of points to return

Value
Returns a point data set consisting of n points selected randomly from the set pts.

References

tor.shift

Toroidal shift on a point data set

Description
Perform a toroidal shift on a point data set

Usage
tor.shift(pts,xsh=0.0,ysh=0.0,rect)

Arguments
pts The point data set to shift
xsh Amount to shift along the x-axis
ysh Amount to shift along the y-axis
rect A rectangle defining the region for the toroidal map. If not given, the bounding box of pts is used.
Details

The planar region defined by rect is assumed connected at its top and bottom edges, and at its left and right sides. A shift of xsh and ysh is applied to the points and the resulting set of points returned.

Value

A point data set like pts, but after application of a toroidal shift along the x and y axes.

References


See Also

Shift

tribble  Diggle-Rowlingson Raised Incidence Model

Description

Fits the Diggle-Rowlingson Raised Incidence Model.

Usage

tribble(ccflag, vars=NULL, alphas=NULL, betas=NULL, rho, which=1:length(alphas), covars=NULL, thetas=NULL, steps=NULL, reqmin=0.001, icount=50, hessian=NULL)

Arguments

ccflag  Case-control flag : a vector of ones and zeroes.
vars  A matrix where vars[i, j] is the distance squared from point i to source j.
alphas  Initial value of the alpha parameters.
betas  Initial value of the beta parameters.
rho  Initial value of the rho parameter.
which  Defines the mapping from sources to parameters.
covars  A matrix of covariates to be modelled as log-linear terms. The element covars[i, j] is the value of covariate j for case/control i.
thetas  Initial values of covariate parameters.
triblik

steps Step sizes for the Nelder-Mead simplex algorithm.
reqmin Tolerance for simplex algorithm
icount Iteration count for simplex algorithm
hessian by default NULL, any other value causes hessian to be computed and returned

Value

The return value is a list with many components, and class ribfit.

alphas A vector of the alpha parameters at the maximum
betas A vector of the beta values at the maximum
rho The value of rho at the maximum
logl The maximised log-likelihood
null.logl The null log-likelihood
call The function call to tribble

For further information see Diggle and Rowlingson (1993).

References


See Also

triblik, dsquare

triblik Log-likelihood for the Diggle-Rowlingson raised incidence model

Description

Calculates the log-likelihood for the Diggle-Rowlingson raised incidence model.

Usage

triblik(ccflag, vars=NULL, alphas=NULL, betas=NULL, rho, which=1:length(alphas), covars=NULL, thetas=NULL)
Arguments

ccflag  Case-control flag: a vector of ones and zeroes.
vars    A matrix where \( vars[i,j] \) is the distance squared from point \( i \) to source \( j \).
alphas  The alpha parameters.
betas   The beta parameters.
rho     The rho parameter.
which   Defines the mapping from sources to parameters.
covars  A matrix of covariates to be modelled as log-linear terms. The element \( covars[i,j] \) is the value of covariate \( j \) for case/control \( i \).
thetas  The covariate parameters.

Value

The log-likelihood for the given parameters and the given distances and optional covariates is returned.

References


See Also

\texttt{tribble}, \texttt{dsquare}

uganda

\textit{Craters in Uganda}

Description

Locations of craters in a volcanic field in Uganda

Usage

\texttt{data(uganda)}

Format

A list corresponding to a Venables and Ripley point object with 120 observations

\begin{verbatim}
x   numeric  grid eastings
y   numeric  grid northings
area list  bounding box with xl, xu, yl, yu
poly array  polygon boundary with columns x and y
\end{verbatim}
**zoom**

*Source*


**References**


---

**Description**

Interactively specify a region of a plot for expansion

**Usage**

```r
zoom(quiet=FALSE,out=FALSE,...)
```

**Arguments**

- `quiet` If false, prompt the user to enter two coordinates. If true, say nothing.
- `out` If true, expand the limits of the current plot by a factor of three, centred on the current plot.
- `...` Other arguments are passed through to pointmap.

**Details**

A prompt is optionally displayed, and the user selects two points forming the diagonal of a rectangle. A new, empty plot is created that has its axis limits set to the bounding square of the selected rectangle. If `out=TRUE`, no prompt is displayed, and a new blank plot is created with its limits in x and y set to span an area three times the height and width centred on the current centre.

**Value**

None

**References**


**See Also**

`pointmap`
Index

* Topic datasets
  amacrines, 11
  bodmin, 14
  burkitt, 15
  cardiff, 16
  okblack, 39
  okwhite, 40
  southlancs, 53
  uganda, 64

* Topic spatial
  addpoints, 10
  areapl, 12
  as.points, 13
  bboxx, 13
  csr, 17
  delpoints, 18
  dsquare, 19
  Fhat, 1
  Fzero, 2
  gen, 19
  getpoly, 20
  Ghat, 3
  gridpts, 21
  inout, 22
  inpip, 23
  is.points, 24
  k12hat, 25
  Kenv.csr, 4
  Kenv.label, 5
  Kenv.pcp, 6
  Kenv.tor, 8
  Kenv.tor1, 9
  kernel2d, 26
  kernel3d, 27
  kernrat, 28
  kerview, 30
  khat, 31
  khvc, 32
  khvmat, 33

  mpoint, 34
  mse2d, 35
  n2dist, 36
  nndistF, 37
  nndistG, 38
  npts, 39
  pcp, 40
  pcp.sim, 42
  pdense, 43
  pip, 44
  plt, 45
  pointmap, 45
  polymap, 46
  print.ribfit, 47
  ranpts, 48
  rLabel, 49
  rtor.shift, 50
  sbox, 51
  secal, 51
  Shift, 52
  splancs, 54
  spoints, 55
  stdiagn, 55
  stkhat, 56
  stmctest, 58
  stsecal, 59
  stvmat, 60
  thin, 61
  tor.shift, 61
  tribble, 62
  triblik, 63
  zoom, 65

  addpoints, 10
  amacrines, 11
  areapl, 12
  as.points, 13

  bboxx, 13, 51
  bodmin, 14
burbdy (burkitt), 15
burkitt, 15
burpts (burkitt), 15
cardiff, 16
csr, 5, 17, 20, 49
delpoints, 11, 18
dsquare, 19, 63, 64
Fhat, 1, 3, 4, 36–38, 45
Fzero, 2, 2, 36–38, 43
gen, 19
getpoly, 20
Ghat, 3, 3, 36–38, 45
gridpts, 21
inout, 22, 44
inpip, 22, 23, 44
is.points, 24
k12hat, 8, 25
Kenv.csr, 4, 32
Kenv.label, 5
Kenv.pcp, 6, 41, 42
Kenv.tor, 8, 10
Kenv.tor1, 9
kernel2d, 26, 29, 35
kernel3d, 27, 30
kernrat, 28
kerview, 28, 30
khat, 5–7, 31, 33, 41, 42, 52
khvc, 32
khvmat, 33, 33
mpoint, 34, 46, 47
mse2d, 29, 35
n2dist, 36
nndistF, 2, 36, 37, 38
nndistG, 4, 37, 38
npts, 39, 55
okblack, 39
okwhite, 40
old.incinerator (southlancs), 53
optim, 41
par, 47
pcp, 7, 40, 42
pcp.sim, 7, 41, 42
pdense, 3, 43
pip, 22, 24, 44
plot.khat (khat), 31
plt, 45
pointmap, 34, 45, 65
polymap, 46
print.khat (khat), 31
print.ribfit, 47
ranpts, 48
rLabel, 6, 49
rtor.shift, 8, 50
sbox, 14, 51
secal, 33, 34, 51
Shift, 50, 52, 62
southlancs, 53
spkernel2d (kernel2d), 26
splancs, 54
spoints, 55
stdiagn, 55, 57, 59, 60
stkh, 56, 56, 59, 60
stmctest, 56, 57, 58, 60
stsecal, 56, 57, 59, 60, 60
stvmat, 56, 57, 59, 60
thin, 61
tor.shift, 53, 61
tribble, 19, 48, 62, 64
triblik, 19, 63, 63
uganda, 64
zoom, 65