Package 'IDSpatialStats'

July 28, 2024

Version 0.4.0

Date 2024-07-22

Title Estimate Global Clustering in Infectious Disease

License GPL (>= 2)

Description Implements various novel and standard clustering statistics and other analyses useful for understanding the spread of infectious disease.

RoxygenNote 7.3.1

Encoding UTF-8

LazyData true

Imports igraph, spatstat.explore, spatstat.geom

Depends doParallel, foreach, parallel, R (>= 2.10)

Suggests knitr, testthat (>= 3.0.0)

Config/testthat/edition 3

NeedsCompilation yes

Author John Giles [aut, ctb],
Henrik Salje [aut, ctb],
Justin Lessler [aut, cre],
Timothy Willard [ctb]

Maintainer Justin Lessler <jlessler@unc.edu>

Repository CRAN

Date/Publication 2024-07-28 20:50:07 UTC

Contents

DengueSimR01 2
DengueSimR02 3
DengueSimRepresentative 4
est.transdist 4
est.transdist.bootstrap.ci 6
est.transdist.temporal 8

est.transdist.temporal.bootstrap.ci . . . 11
est.transdist.theta.weights . . . 13
est.wt.matrix . . . 15
est.wt.matrix.weights . . . 16
get.cross.K . . . 17
get.cross.PCF . . . 19
get.pi . . . 20
get.pi.bootstrap . . . 21
get.pi.ci . . . 23
get.pi.permute . . . 24
get.pi.typed . . . 26
get.pi.typed.bootstrap . . . 27
get.pi.typed.permute . . . 28
get.tau . . . 29
get.tau.bootstrap . . . 32
get.tau.ci . . . 34
get.tau.permute . . . 35
get.tau.typed . . . 37
get.tau.typed.bootstrap . . . 38
get.tau.typed.permute . . . 40
get.theta . . . 42
get.theta.bootstrap . . . 43
get.theta.ci . . . 45
get.theta.permute . . . 46
get.theta.typed . . . 48
get.theta.typed.bootstrap . . . 49
get.theta.typed.permute . . . 50
get.transdist.theta . . . 52
sim.epidemic . . . 53
sim.plot . . . 55

Index **56**

`DengueSimR01` *Simulated dataset of dengue transmission with basic reproductive number of 1*

Description

Dataset simulated using an agent based model with a spatially heterogeneous population structure. Infectious agents were introduced resulting in agent to agent transmission. The distance between successive cases in a transmission chain were randomly drawn from a uniform distribution U(0,100). Each infectious agent resulted in a single transmission to another agent after a delay of 15 days, reflecting the generation time of dengue. There are 11 transmission chains, each with a different genotype. The genotypes are subdivided into four serotypes.

Usage

```
DengueSimR01
```

Format

Matrix with five columns representing the X and Y coordinates of infected individuals, the time of infection, the genotype of the infecting pathogen and the serotype of the infecting pathogen.

Author(s)

Justin Lessler and Henrik Salje

`DengueSimR02` *Simulated dataset of dengue cases with basic reproductive number of 2*

Description

Dataset simulated using an agent based model with a spatially heterogeneous population structure. Infectious agents were introduced resulting in agent to agent transmission. The distance between successive cases in a transmission chain were randomly drawn from a uniform distribution U(0,100). Each infectious agent resulted in transmissions to two other agents after a delay of 15 days, reflecting the generation time of dengue. There are 11 transmission chains, each with a different genotype. The genotypes are subdivided into four serotypes.

Usage

```
DengueSimR02
```

Format

Matrix with five columns representing the X and Y coordinates of infected individuals, the time of infection, the genotype of the infecting pathogen and the serotype of the infecting pathogen.

Author(s)

Justin Lessler and Henrik Salje

DengueSimRepresentative

Simulated dataset of dengue cases with representative underlying population

Description

Dataset simulated using an agent based model with a spatially heterogeneous population structure. Infectious agents were introduced resulting in agent to agent transmission. The distance between successive cases in a transmission chain were randomly drawn from a uniform distribution U(0,100). Each infectious agent resulted in transmissions to two other agents after a delay of 15 days, reflecting the generation time of dengue. There are 11 transmission chains, each with a different genotype. The genotypes are subdivided into four serotypes. 500 randomly selected individuals from the underlying population also included.

Usage

```
DengueSimRepresentative
```

Format

Matrix with five columns representing the X and Y coordinates of infected individuals, the time of infection, the genotype of the infecting pathogen and the serotype of the infecting pathogen. Individuals representative from the underlying population have '-999'for time, genotype and serotype.

Author(s)

Justin Lessler and Henrik Salje

est.transdist *Estimate transmission distance*

Description

this function estimates the mean transmission distance of an epidemic when given the locations and times of symptomatic cases and the mean and standard deviation of the generation time of the infecting pathogen

Usage

```
est.transdist(
  epi.data,
  gen.t.mean,
  gen.t.sd,
  t1,
  max.sep,
```

```
  max.dist,
  n.transtree.reps = 100,
  theta.weights = NULL
)
```

Arguments

epi.data	a three-column matrix giving the coordinates (x and y) and time of infection (t for all cases in an epidemic (columns must be in x, y, t order)
gen.t.mean	mean generation time of the infecting pathogen
gen.t.sd	standard deviation of generation time of the infecting pathogen
t1	time step to begin estimation of transmission distance
max.sep	maximum number of time steps allowed between two cases (passed to the get.transdist.theta function)
max.dist	maximum spatial distance between two cases considered in calculation
n.transtree.reps	number of time to simulate transmission trees when estimating the weights of theta (passed to the est.transdist.theta.weights function, default = 10). Warning: higher values of this parameter cause significant increases in computation time.
theta.weights	use external matrix of theta weights. If NULL (default) the matrix of theta weights is automatically estimated by calling the est.transdist.theta.weights function

Value

a list containing the estimated mean distance of the transmission kernel (mu.est) and its standard deviation (sigma.est). Bounded estimates (bound.mu.est and bound.sigma.est) are also given for when the assumption of equal mean and standard deviation is violated.

Author(s)

John Giles, Justin Lessler, and Henrik Salje

References

Salje H, Cummings DAT and Lessler J (2016). "Estimating infectious disease transmission distances using the overall distribution of cases." Epidemics, 17, pp. 10–18. ISSN 1755-4365, doi: 10.1016/j.epidem.2016.10.001.

See Also

Other est.wt: est.wt.matrix(), est.wt.matrix.weights()

Other transdist: est.transdist.bootstrap.ci(), est.transdist.temporal(), est.transdist.temporal.bootstrap.
est.transdist.theta.weights(), get.transdist.theta()

Examples

```
set.seed(123)

# Exponentially distributed transmission kernel with mean and standard deviation = 100
dist.func <- alist(n=1, a=1/100, rexp(n, a))

# Simulate epidemic
a <- sim.epidemic(R=1.5,
                  gen.t.mean=7,
                  gen.t.sd=2,
                  min.cases=50,
                  tot.generations=12,
                  trans.kern.func=dist.func)

# Estimate mean and standara deviation of transmission kernel
b <- est.transdist(epi.data=a,
                   gen.t.mean=7,
                   gen.t.sd=2,
                   t1=0,
                   max.sep=1e10,
                   max.dist=1e10,
                   n.transtree.reps=10)
b
```

`est.transdist.bootstrap.ci`

Bootstrap mean transmission distance values

Description

Runs `est.trandsdist` on multiple bootstraps of the data and calculates confidence intervals for the mean transmission distance.

Usage

```
est.transdist.bootstrap.ci(
  epi.data,
  gen.t.mean,
  gen.t.sd,
  t1,
  max.sep,
  max.dist,
  n.transtree.reps = 100,
  mean.equals.sd = FALSE,
  theta.weights = NULL,
```

```
  boot.iter,
  ci.low = 0.025,
  ci.high = 0.975,
  parallel = FALSE,
  n.cores = NULL
)
```

Arguments

epi.data	a three-column matrix giving the coordinates (x and y) and time of infection (t for all cases in an epidemic (columns must be in x, y, t order)
gen.t.mean	mean generation time of the infecting pathogen
gen.t.sd	standard deviation of generation time of the infecting pathogen
t1	time step to begin estimation of transmission distance
max.sep	maximum number of time steps allowed between two cases (passed to the get.transdist.theta function)
max.dist	maximum spatial distance between two cases considered in calculation
n.transtree.reps	number of time to simulate transmission trees when estimating the weights of theta (passed to the est.transdist.theta.weights function, default = 10). Warning: higher values of this parameter cause significant increases in computation time.
mean.equals.sd	logical term indicating if the mean and standard deviation of the transmission kernel are expected to be equal (default = FALSE)
theta.weights	use external matrix of theta weights. If NULL (default) the matrix of theta weights is automatically estimated by calling the est.transdist.theta.weights function
boot.iter	the number of bootstrapped iterations to perform
ci.low	low end of the confidence interval (default = 0.025)
ci.high	high end of the confidence interval (default = 0.975)
parallel	run bootstraps in parallel (default = FALSE)
n.cores	number of cores to use when parallel = TRUE (default = NULL, which uses half the available cores)

Value

a list object containing the point estimate for mean transmission distance and low and high bootstrapped confidence intervals

Author(s)

John Giles, Justin Lessler, and Henrik Salje

References

Salje H, Cummings DAT and Lessler J (2016). "Estimating infectious disease transmission distances using the overall distribution of cases." Epidemics, 17, pp. 10–18. ISSN 1755-4365, doi: 10.1016/j.epidem.2016.10.001.

See Also

Other transdist: `est.transdist()`, `est.transdist.temporal()`, `est.transdist.temporal.bootstrap.ci()`, `est.transdist.theta.weights()`, `get.transdist.theta()`

Examples

```
set.seed(1)

# Exponentially distributed transmission kernel with mean and standard deviation = 100
dist.func <- alist(n=1, a=1/100, rexp(n, a))

# Simulate epidemic
a <- sim.epidemic(R=2.5,
                  gen.t.mean=7,
                  gen.t.sd=2,
                  min.cases=20,
                  tot.generations=5,
                  trans.kern.func=dist.func)

# Estimate mean transmission kernel and its bootstrapped confidence intervals
b <- est.transdist.bootstrap.ci(epi.data=a,
                                gen.t.mean=7,
                                gen.t.sd=2,
                                t1=0,
                                max.sep=1e10,
                                max.dist=1e10,
                                n.transtree.reps=10,
                                mean.equals.sd=TRUE,
                                boot.iter=10,
                                ci.low=0.025,
                                ci.high=0.975,
                                n.cores=2)
b
```

`est.transdist.temporal`

Change in mean transmission distance over time

Description

Estimates the change in mean transmission distance over the duration of the epidemic by running `est.trandsdist` on all cases occuring up to each time point.

Usage

```
est.transdist.temporal(
  epi.data,
  gen.t.mean,
  gen.t.sd,
  t1,
  max.sep,
  max.dist,
  n.transtree.reps = 10,
  mean.equals.sd = FALSE,
  theta.weights = NULL,
  parallel = FALSE,
  n.cores = NULL
)
```

Arguments

`epi.data`	a three-column matrix giving the coordinates (`x` and `y`) and time of infection (`t` for all cases in an epidemic (columns must be in `x`, `y`, `t` order)
`gen.t.mean`	mean generation time of the infecting pathogen
`gen.t.sd`	standard deviation of generation time of the infecting pathogen
`t1`	time step to begin estimation of transmission distance
`max.sep`	maximum number of time steps allowed between two cases (passed to the `get.transdist.theta` function)
`max.dist`	maximum spatial distance between two cases considered in calculation
`n.transtree.reps`	number of time to simulate transmission trees when estimating the weights of theta (passed to the `est.transdist.theta.weights` function, default = 10). Higher values of this parameter cause significant increases in computation time.
`mean.equals.sd`	logical term indicating if the mean and standard deviation of the transmission kernel are expected to be equal (default = FALSE)
`theta.weights`	use external matrix of theta weights. If NULL (default) the matrix of theta weights is automatically estimated by calling the `est.transdist.theta.weights` function
`parallel`	run time steps in parallel (default = FALSE)
`n.cores`	number of cores to use when `parallel` = TRUE (default = NULL, which uses half the available cores)

Value

a numeric matrix containing the point estimate for mean transmission distance for each unique time step of the epidemic and the sample size n used to make the estimate NAs are returned for time steps which contain fewer than three cases

Author(s)

John Giles, Justin Lessler, and Henrik Salje

References

Salje H, Cummings DAT and Lessler J (2016). "Estimating infectious disease transmission distances using the overall distribution of cases." Epidemics, 17, pp. 10–18. ISSN 1755-4365, doi: 10.1016/j.epidem.2016.10.001.

See Also

Other transdist: `est.transdist()`, `est.transdist.bootstrap.ci()`, `est.transdist.temporal.bootstrap.ci()`, `est.transdist.theta.weights()`, `get.transdist.theta()`

Examples

```
set.seed(123)

# Exponentially distributed transmission kernel with mean and standard deviation = 100
dist.func <- alist(n=1, a=1/100, rexp(n, a))

# Simulate epidemic
a <- sim.epidemic(R=2,
                  gen.t.mean=7,
                  gen.t.sd=2,
                  tot.generations=7,
                  min.cases=30,
                  trans.kern.func=dist.func)

a <- a[sample(1:nrow(a), 50),] # subsample a to 50 observations

# Estimate mean transmission kernel over time
b <- est.transdist.temporal(epi.data=a,
                            gen.t.mean=7,
                            gen.t.sd=2,
                            t1=0,
                            max.sep=1e10,
                            max.dist=1e10,
                            n.transtree.reps=5,
                            mean.equals.sd=TRUE,
                            n.cores=2)
b

plot(b[,2], pch=19, col='grey', ylim=c(min(b[,2], na.rm=TRUE), max(b[,2], na.rm=TRUE)),
```

```
     xlab='Time step', ylab='Estimated mean of transmission kernel')
abline(h=100, col='red', lty=2)
axis(3, b[,2])

low <- loess(b[,2] ~ as.vector(1:length(b[,2])))
low <- predict(low, newdata=data.frame(as.vector(1:length(b[,2]))))
lines(low, lwd=3, col='blue')
```

`est.transdist.temporal.bootstrap.ci` *Bootstrapped confidence intervals for the change in mean transmission distance over time*

Description

Estimates bootstrapped confidence intervals for the mean transmission distance over the duration of the epidemic by running `est.trandsdist` on all cases occuring up to each time point.

Usage

```
est.transdist.temporal.bootstrap.ci(
  epi.data,
  gen.t.mean,
  gen.t.sd,
  t1,
  max.sep,
  max.dist,
  n.transtree.reps = 100,
  mean.equals.sd = FALSE,
  theta.weights = NULL,
  boot.iter,
  ci.low = 0.025,
  ci.high = 0.975,
  parallel = FALSE,
  n.cores = NULL
)
```

Arguments

`epi.data`	a three-column matrix giving the coordinates (`x` and `y`) and time of infection (`t` for all cases in an epidemic (columns must be in `x`, `y`, `t` order)
`gen.t.mean`	mean generation time of the infecting pathogen
`gen.t.sd`	standard deviation of generation time of the infecting pathogen
`t1`	time step to begin estimation of transmission distance

max.sep	maximum number of time steps allowed between two cases (passed to the get.transdist.theta function)
max.dist	maximum spatial distance between two cases considered in calculation
n.transtree.reps	number of time to simulate transmission trees when estimating the weights of theta (passed to the est.transdist.theta.weights function, default = 10). Warning: higher values of this parameter cause significant increases in computation time.
mean.equals.sd	logical term indicating if the mean and standard deviation of the transmission kernel are expected to be equal (default = FALSE)
theta.weights	use external matrix of theta weights. If NULL (default) the matrix of theta weights is automatically estimated by calling the est.transdist.theta.weights function
boot.iter	the number of bootstrapped iterations to perform
ci.low	low end of the confidence interval (default = 0.025)
ci.high	high end of the confidence interval (default = 0.975)
parallel	run bootstraps in parallel (default = FALSE)
n.cores	number of cores to use when parallel = TRUE (default = NULL, which uses half the available cores)

Value

a four-column numeric matrix containing the point estimate for mean transmission distance, low and high bootstrapped confidence intervals, and the sample size up to each time step

Author(s)

John Giles, Justin Lessler, and Henrik Salje

References

Salje H, Cummings DAT and Lessler J (2016). "Estimating infectious disease transmission distances using the overall distribution of cases." Epidemics, 17, pp. 10–18. ISSN 1755-4365, doi: 10.1016/j.epidem.2016.10.001.

See Also

Other transdist: est.transdist(), est.transdist.bootstrap.ci(), est.transdist.temporal(), est.transdist.theta.weights(), get.transdist.theta()

Examples

```
set.seed(123)

# Exponentially distributed transmission kernel with mean and standard deviation = 100
dist.func <- alist(n=1, a=1/100, rexp(n, a))
```

```
# Simulate epidemic
a <- sim.epidemic(R=2,
                  gen.t.mean=7,
                  gen.t.sd=2,
                  tot.generations=8,
                  min.cases=30,
                  trans.kern.func=dist.func)

a <- a[sample(1:nrow(a), 70),] # subsample a to 70 observations

# Estimate change in mean transmission kernel over time with confidence intervals
b <- est.transdist.temporal.bootstrap.ci(epi.data=a,
                                         gen.t.mean=7,
                                         gen.t.sd=2,
                                         t1=0,
                                         max.sep=1e10,
                                         max.dist=1e10,
                                         n.transtree.reps=10,
                                         mean.equals.sd=TRUE,
                                         boot.iter=10,
                                         ci.low=0.025,
                                         ci.high=0.975,
                                         n.cores=2)

plot(b[,2], pch=19, col='grey', ylim=c(min(b[,1:3], na.rm=TRUE), max(b[,2:4], na.rm=TRUE)),
     xlab='Time step', ylab='Estimated mean of transmission kernel')
abline(h=100, col='red', lty=2)
axis(3, 1:nrow(b), b[,5])

low <- loess(b[,2] ~ as.vector(1:nrow(b)), span=1)
low <- predict(low, newdata=data.frame(as.vector(1:nrow(b))))
lines(low, lwd=3, col='blue')

for(i in 3:4) {
     low <- loess(b[,i] ~ as.vector(1:nrow(b)), span=1)
     low <- predict(low, newdata=data.frame(as.vector(1:nrow(b))))
     lines(low, lty=2, lwd=3, col='blue')
}
```

---

est.transdist.theta.weights

Estimate transmission distance theta values by replication

Description

This function estimates the weight of each theta value by performing a user defined number of replications with the `get.transdist.theta` function. The weights of each theta are calculated as

the number of simulations in which a case at time t1 and t2 are separated by theta transmission events.

Usage

```
est.transdist.theta.weights(case.times, gen.t.mean, t.density, t1, n.rep = 100)
```

Arguments

case.times	a vector giving the occurrence time for each case
gen.t.mean	the mean generation time of the infecting pathogen
t.density	a vector giving the generation time density of the infecting pathogen
t1	time step to begin simulation
n.rep	number of replications in the simulation (default = 100)

Value

a three-dimensional array containing the mean normalized theta weights estimated across all replications

Author(s)

John Giles, Justin Lessler, and Henrik Salje

References

Salje H, Cummings DAT and Lessler J (2016). "Estimating infectious disease transmission distances using the overall distribution of cases." Epidemics, 17, pp. 10–18. ISSN 1755-4365, doi: 10.1016/j.epidem.2016.10.001.

See Also

Other transdist: est.transdist(), est.transdist.bootstrap.ci(), est.transdist.temporal(), est.transdist.temporal.bootstrap.ci(), get.transdist.theta()

Examples

```
set.seed(1)

gen.t.mean <- 7
gen.t.sd <- 2
t1 <- 0

# Normally distributed transmission kernel with mean and standard deviation = 100
dist.func <- alist(n=1, a=1/100, rexp(n, a))

# Simulate epidemic
a <- sim.epidemic(R=5,
                  gen.t.mean=gen.t.mean,
```

```
                  gen.t.sd=gen.t.sd,
                  min.cases=5,
                  tot.generations=3,
                  trans.kern.func=dist.func)

# Get case times
a <- a[order(a[,3]),]
case.times <- round(a[,3])
unique.times <- unique(case.times)
ntimes <- length(unique.times)


# Generation time distribution
max.t <- round(max(unique.times) - t1) - 1
n.step <- round(max.t/gen.t.mean)
gen <- rep(0, max.t*2)
for (i in 1:n.step){gen <- gen + dnorm(1:(max.t*2), gen.t.mean*i, gen.t.sd*i)}
gen[1] <- 0 # No instantaneous infections
t.density <- gen/sum(gen)

# Estimation of theta weights matrix
b <- est.transdist.theta.weights(case.times=case.times,
                                 n.rep=3,
                                 gen.t.mean=gen.t.mean,
                                 t1=t1,
                                 t.density=t.density)
```

`est.wt.matrix` *Calculate the Infector-Infectee Wallinga-Teunis matrix*

Description

A function which takes the time of each case occurrence, the generation time distribution of the infecting pathogen, and the matrix of basic Wallinga-Teunis weights and estimates the probability that an infectee occurring time step j (columns) was infected by a case occurring at time i (rows).

Usage

```
est.wt.matrix(case.times, gen.t.dist, basic.wt.weights = NULL)
```

Arguments

`case.times`	a vector giving the occurrence time for each case
`gen.t.dist`	a vector giving the generation time distribution for the infecting pathogen
`basic.wt.weights`	a matrix giving the basic normalized Wallinga-Teunis weights for each time step (output from the `est.wt.matrix.weights` function). If this argument is `NULL` (the default), the basic Wallinga-Teunis matrix will be calculated automatically.

Value

a numerical matrix with the number of columns and rows equal to the number of cases in the epidemic

Author(s)

John Giles, Justin Lessler, and Henrik Salje

References

Salje H, Cummings DAT and Lessler J (2016). "Estimating infectious disease transmission distances using the overall distribution of cases." Epidemics, 17, pp. 10–18. ISSN 1755-4365, doi: 10.1016/j.epidem.2016.10.001.

See Also

Other est.wt: `est.transdist()`, `est.wt.matrix.weights()`

Examples

```
case.times <- c(1,2,2,3,3)
gen <- c(0, 2/3, 1/3, 0, 0)
t.density <- gen/sum(gen)

a <- est.wt.matrix(case.times=case.times, gen.t.dist=t.density)
```

`est.wt.matrix.weights` *Estimate matrix of basic Wallinga-Teunis weights*

Description

A function called by `est.wt.matrix`, which calculates the basic weights in the Wallinga-Teunis matrix given the time of each case occurrence and the generation time distribution of the pathogen. Code adapted from the **R0** package.

Usage

```
est.wt.matrix.weights(case.times, gen.t.dist)
```

Arguments

`case.times`	a vector giving the occurrence time for each case
`gen.t.dist`	a vector giving the generation time distribution for the infecting pathogen

Value

a numerical matrix with the number of columns and rows equal to the number of time steps in the epidemic

Author(s)

John Giles, Justin Lessler, and Henrik Salje

References

Boelle P and Obadia T (2015). R0: Estimation of R0 and Real-Time Reproduction Number from Epidemics. R package version 1.2-6, https://CRAN.R-project.org/package=R0.

Salje H, Cummings DAT and Lessler J (2016). "Estimating infectious disease transmission distances using the overall distribution of cases." Epidemics, 17, pp. 10–18. ISSN 1755-4365, doi: 10.1016/j.epidem.2016.10.001.

See Also

Other est.wt: `est.transdist()`, `est.wt.matrix()`

Examples

```
case.times <- c(1,2,2,3,3)
gen <- c(0, 2/3, 1/3, 0, 0)
t.density <- gen/sum(gen)

a <- est.wt.matrix.weights(case.times=case.times, gen.t.dist=t.density)
```

---

get.cross.K *Cross type K function using homotypic and heterotypic case types*

Description

A wrapper function of the Kcross function from the **spatstat.explore** package (Baddeley et al. 2016) that takes epidemiological data used by **IDSpatialStats** functions and calculates the cross type K-function based on user defined case type homology

Usage

```
get.cross.K(epi.data, type, hom, het = NULL, r = NULL, correction = "none")
```

Arguments

epi.data	a three-column numerical matrix that contains coordinates (x and y) for each case and information on case type (e.g. genotype or serotype). First two columns must be x and y
type	an integer giving the column that contains information on case type. Must be an integer or a character
hom	a scalar or vector giving the homotypic case type(s). Equivalent to the 'j' point type used in the cross K function. Must be an integer or character
het	a scalar or vector giving the heterotypic case type(s). Equivalent to the 'i' point type used in the cross K function. The default is NULL, which uses any case type not defined in the hom argument as heterotypic. Must be an integer or a character
r	a numeric vector giving the spatial distances
correction	type of edge correction to be applied (default set to 'none'). See the Kcross function in the **spatstat.explore** package for more details

Value

a data frame with a minimum of three columns giving the radius (r), the theoretical value of the K function for a Poisson process (theo), and value of the K function evaluated at radius r. The column name gives the type of edge correction used

Author(s)

John Giles

References

Baddeley A, Rubak E, and Turner R. (2016). "Spatial Point Patterns: Methodology and Applications with R". CRC Press.

Examples

```
data(DengueSimR01)

k <- get.cross.K(epi.data=DengueSimR01, type=5, hom=2, het=NULL, r=NULL, correction='border')

plot(k[,2], type='l', col='red', lty=2, xlab='r', ylab='cross K function')
lines(k$border)
```

`get.cross.PCF` *Cross type Pair Correlation Function using homotypic and heterotypic case types*

Description

A wrapper function of the pcf function from the **spatstat.explore** package (Baddeley et al. 2016) that takes epidemiological data used by **IDSpatialStats** functions and calculates the cross type Pair Correlation Function based on user defined case type homology

Usage

```
get.cross.PCF(epi.data, type, hom, het = NULL, r = NULL, correction = "none")
```

Arguments

`epi.data`	a three-column numerical matrix that contains coordinates (`x` and `y`) for each case and information on case type (e.g. genotype or serotype). First two columns must be `x` and `y`
`type`	an integer giving the column that contains information on case type. Must be an integer or a character
`hom`	a scalar or vector giving the homotypic case type(s). Equivalent to the 'j' point type used in the cross K function. Must be an integer or character
`het`	a scalar or vector giving the heterotypic case type(s). Equivalent to the 'i' point type used in the cross K function. The default is `NULL`, which uses any case type not defined in the `hom` argument as heterotypic. Must be an integer or a character
`r`	a numeric vector giving the spatial distances
`correction`	type of edge correction to be applied (default set to 'none'). See the pcf function in the **spatstat.explore** package for more details

Value

a data frame with two columns giving the radius `r`, the theoretical value of the Pair Correlation Function for a Poisson process (`theo`), and value of the Pair Correlation Function `pcf`

Author(s)

John Giles

References

Baddeley A, Rubak E, and Turner R. (2016). "Spatial Point Patterns: Methodology and Applications with R". CRC Press.

Examples

```
data(DengueSimR01)

g <- get.cross.PCF(epi.data=DengueSimR01, type=5, hom=2, het=NULL, r=NULL, correction='none')

plot(g$pcf, type='l', xlab='r', ylab='cross PCF')
abline(h=1, col='red', lty=2)
```

---

get.pi *Generalized version of* get.pi

Description

Generalized version of the get.pi function that takes in an arbitrary function and returns the probability that a point within a particular range of a point of interest shares the relationship specified by the passed in function with that point.

Usage

```
get.pi(posmat, fun, r = 1, r.low = rep(0, length(r)), data.frame = TRUE)
```

Arguments

posmat	a matrix with columns x, y and any other named columns columns needed by fun
fun	a function that takes in two rows of posmat and returns: 1. for pairs included in the numerator and denominator 2. for pairs that should only be included in the denominator 3. for pairs that should be ignored all together Note that names from posmat are not preserved in calls to fun, so the columns of the matrix should be referenced numerically so this is not available to the fun
r	the series of spatial distances (or there maximums) we are interested in
r.low	the low end of each range, 0 by default
data.frame	logical indicating whether to return results as a data frame (default = TRUE)

Value

pi value for each distance range that we look at. Where:

$$\pi(d_1, d_2) = \frac{\sum \mathbf{1}(d_{ij} \in [d_1, d_2))\mathbf{1}(f(i,j) = 1)}{\sum\sum \mathbf{1}[d_{ij} \in (d_1, d_2))\mathbf{1}(f(i,j) \in \{1, 2\})}$$

Author(s)

Justin Lessler and Henrik Salje

See Also

Other get.pi: `get.pi.bootstrap()`, `get.pi.ci()`, `get.pi.permute()`, `get.pi.typed()`, `get.pi.typed.bootstrap()`, `get.pi.typed.permute()`

Other spatialtau: `get.tau()`, `get.theta()`

Examples

```
data(DengueSimR02)

r.max<-seq(20,1000,20)
r.min<-seq(0,980,20)

sero.type.func<-function(a,b,tlimit=20){
  if(a[5]==b[5]&(abs(a[3]-b[3])<=tlimit)){rc=1}
  else{rc=2}
  return(rc)
}

sero.pi<-get.pi(DengueSimR02,sero.type.func,r=r.max,r.low=r.min)
```

`get.pi.bootstrap` *Bootstrap* `get.pi` *values.*

Description

Runs `get.pi` on multiple bootstraps of the data. Is formulated such that the relationships between points and themselves will not be calculated.

Usage

```
get.pi.bootstrap(
  posmat,
  fun,
  r = 1,
  r.low = rep(0, length(r)),
  boot.iter = 500,
  data.frame = TRUE
)
```

Arguments

`posmat`	a matrix with columns type, x and y
`fun`	the function to decide relationships
`r`	the series of spatial distances we are interested in

r.low	the low end of each range. 0 by default
boot.iter	the number of bootstrap iterations
data.frame	logical indicating whether to return results as a data frame (default = TRUE)

Value

pi values for all the distances we looked at

Note

In each bootstrap iteration N observations are drawn from the existing data with replacement. To avoid errors in inference resulting from the same observatin being compared with itself in the bootstrapped data set, original indices are perserved, and pairs of points in the bootstrapped dataset with the same original index are ignored.

Author(s)

Justin Lessler and Henrik Salje

See Also

Other get.pi: get.pi(), get.pi.ci(), get.pi.permute(), get.pi.typed(), get.pi.typed.bootstrap(), get.pi.typed.permute()

Examples

```
#compare normally distributed with uniform points
x<-cbind(1,runif(100,-100,100), runif(100,-100,100))
x<-rbind(x, cbind(2,rnorm(100,0,20), rnorm(100,0,20)))
colnames(x) <- c("type","x","y")

fun<-function(a,b) {
    if(a[1]!=2) return(3)
    if (b[1]==2) return(1)
    return(2)
}

r.max<-seq(10,100,10)
r.min<-seq(0,90,10)
r.mid <- (r.max+r.min)/2


pi<-get.pi(x,fun,r=r.max,r.low=r.min)
pi.boot<-get.pi.bootstrap(x,fun,r=r.max,r.low=r.min,boot.iter=100)

pi.ci<-apply(pi.boot[,-(1:2)],1,quantile,probs=c(0.25,0.75))

plot(r.mid, pi$pi , type="l")
lines(r.mid, pi.ci[1,] , lty=2)
lines(r.mid, pi.ci[2,] , lty=2)
```

get.pi.ci *Calculate bootstrapped confidence intervals for* get.pi *values.*

Description

Wrapper to get.pi.bootstrap that takes care of calculating the confidence intervals based on the bootstrapped values..

Usage

```
get.pi.ci(
  posmat,
  fun,
  r = 1,
  r.low = rep(0, length(r)),
  boot.iter = 1000,
  ci.low = 0.025,
  ci.high = 0.975,
  data.frame = TRUE
)
```

Arguments

posmat	a matrix with columns type, x and y
fun	the function to decide relationships
r	the series of spatial distances wer are interested in
r.low	the low end of each range. 0 by default
boot.iter	the number of bootstrap iterations
ci.low	the low end of the ci...0.025 by default
ci.high	the high end of the ci...0.975 by default
data.frame	logical indicating whether to return results as a data frame (default = TRUE)

Value

a matrix with a row for the high and low values and a column per distance

Author(s)

Justin Lessler

See Also

Other get.pi: get.pi(), get.pi.bootstrap(), get.pi.permute(), get.pi.typed(), get.pi.typed.bootstrap(), get.pi.typed.permute()

Examples

```
#compare normally distributed with uniform points
x<-cbind(1,runif(100,-100,100), runif(100,-100,100))
x<-rbind(x, cbind(2,rnorm(100,0,20), rnorm(100,0,20)))
colnames(x) <- c("type","x","y")

fun<-function(a,b) {
    if(a[1]!=2) return(3)
    if (b[1]==2) return(1)
    return(2)
}

r.max<-seq(10,100,10)
r.min<-seq(0,90,10)
r.mid <- (r.max+r.min)/2


pi<-get.pi(x,fun,r=r.max,r.low=r.min)
pi.ci<-get.pi.ci(x,fun,r=r.max,r.low=r.min,boot.iter=100)

plot(r.mid, pi$pi, type="l")
lines(r.mid, pi.ci[,2] , lty=2)
lines(r.mid, pi.ci[,3] , lty=2)
```

---

get.pi.permute *get the null distribution of the* get.pi *function*

Description

Does permutations to calculate the null distribution of get pi if there were no spatial dependence. Randomly reassigns coordinates to each observation permutations times

Usage

```
get.pi.permute(
  posmat,
  fun,
  r = 1,
  r.low = rep(0, length(r)),
  permutations,
  data.frame = TRUE
)
```

Arguments

posmat	a matrix with columns type, x and y
fun	the function to evaluate
r	the series of spatial distances we are interested in
r.low	the low end of each range....0 by default
permutations	the number of permute iterations
data.frame	logical indicating whether to return results as a data frame (default = TRUE)

Value

pi values for all the distances we looked at

See Also

Other get.pi: get.pi(), get.pi.bootstrap(), get.pi.ci(), get.pi.typed(), get.pi.typed.bootstrap(), get.pi.typed.permute()

Examples

```
#compare normally distributed with uniform points
x<-cbind(1,runif(100,-100,100), runif(100,-100,100))
x<-rbind(x, cbind(2,rnorm(100,0,20), rnorm(100,0,20)))
colnames(x) <- c("type","x","y")

fun<-function(a,b) {
    if(a[1]!=2) return(3)
    if (b[1]==2) return(1)
    return(2)
}

r.max<-seq(10,100,10)
r.min<-seq(0,90,10)
r.mid <- (r.max+r.min)/2

pi<-get.pi(x,fun,r=r.max,r.low=r.min)
pi.null<-get.pi.permute(x,fun,r=r.max,r.low=r.min,permutations=100)

null.ci<-apply(pi.null[,-(1:2)],1,quantile,probs=c(0.25,0.75))

plot(r.mid, pi$pi, type="l")
lines(r.mid, null.ci[1,] , lty=2)
lines(r.mid, null.ci[2,] , lty=2)
```

get.pi.typed *Optimized version of* get.pi *for typed data.*

Description

Version of the get.pi function that is optimized for statically typed data. That is data where we are interested in the probability of points within some distance of points of typeA are of typeB.

Usage

```
get.pi.typed(
  posmat,
  typeA = -1,
  typeB = -1,
  r = 1,
  r.low = rep(0, length(r)),
  data.frame = TRUE
)
```

Arguments

posmat	a matrix with columns type, x and y
typeA	the "from" type that we are interested in, -1 is wildcard
typeB	the "to" type that we are interested i, -1 is wildcard
r	the series of spatial distances wer are interested in
r.low	the low end of each range....0 by default
data.frame	logical indicating whether to return results as a data frame (default = TRUE)

Value

pi values for all the distances we looked at

Author(s)

Justin Lessler and Henrik Salje

See Also

Other get.pi: get.pi(), get.pi.bootstrap(), get.pi.ci(), get.pi.permute(), get.pi.typed.bootstrap(), get.pi.typed.permute()

Examples

```
data(DengueSimR02)

r.max<-seq(20,1000,20)
r.min<-seq(0,980,20)

#Lets see if there's a difference in spatial dependence by time case occurs
type<-2-(DengueSimR02[,"time"]<120)
tmp<-cbind(DengueSimR02,type=type)

typed.pi<-get.pi.typed(tmp,typeA=1,typeB=2,r=r.max,r.low=r.min)
```

---

get.pi.typed.bootstrap

runs bootstrapping on `get.pi.typed`

Description

Bootstraps typed pi values. Makes sure distances between a sample and another draw of itself are left out

Usage

```
get.pi.typed.bootstrap(
  posmat,
  typeA = -1,
  typeB = -1,
  r = 1,
  r.low = rep(0, length(r)),
  boot.iter,
  data.frame = TRUE
)
```

Arguments

`posmat`	a matrix with columns type, x and y
`typeA`	the "from" type that we are interested in, -1 is wildcard
`typeB`	the "to" type that we are interested i, -1 is wildcard
`r`	the series of spatial distances we are interested in
`r.low`	the low end of each range....0 by default
`boot.iter`	the number of bootstrap iterations
`data.frame`	logical indicating whether to return results as a data frame (default = TRUE)

Value

pi values for all the distances we looked at

See Also

Other get.pi: get.pi(), get.pi.bootstrap(), get.pi.ci(), get.pi.permute(), get.pi.typed(), get.pi.typed.permute()

Examples

```
data(DengueSimR02)

r.max<-seq(20,1000,20)
r.min<-seq(0,980,20)

#Lets see if there's a difference in spatial dependence by time case occurs
type<-2-(DengueSimR02[,"time"]<120)
tmp<-cbind(DengueSimR02,type=type)

typed.pi.bs<-get.pi.typed.bootstrap(tmp,typeA=1,typeB=2,r=r.max,r.low=r.min,boot.iter=100)
```

---

get.pi.typed.permute *get the null distribution of the get.pi.typed function*

Description

Does permutations to calculate the null distribution of get pi if there were no spatial dependence. Randomly reassigns coordinates to each observation permutations times

Usage

```
get.pi.typed.permute(
  posmat,
  typeA = -1,
  typeB = -1,
  r = 1,
  r.low = rep(0, length(r)),
  permutations,
  data.frame = TRUE
)
```

Arguments

posmat	a matrix with columns type, x and y
typeA	the "from" type that we are interested in, -1 is wildcard
typeB	the "to" type that we are interested i, -1 is wildcard
r	the series of spatial distances we are interested in
r.low	the low end of each range....0 by default
permutations	the number of permute iterations
data.frame	logical indicating whether to return results as a data frame (default = TRUE)

Value

pi values for all the distances we looked at

Author(s)

Justin Lessler and Henrik Salje

See Also

Other get.pi: get.pi(), get.pi.bootstrap(), get.pi.ci(), get.pi.permute(), get.pi.typed(), get.pi.typed.bootstrap()

Examples

```
data(DengueSimR02)

r.max<-seq(20,1000,20)
r.min<-seq(0,980,20)

#Lets see if there's a difference in spatial dependence by time case occurs
type<-2-(DengueSimR02[,"time"]<75)
tmp<-cbind(DengueSimR02,type=type)

typed.pi<-get.pi.typed(tmp,typeA=1,typeB=2,r=r.max,r.low=r.min)
typed.pi.type.null<-get.pi.typed.permute(tmp,typeA=1,typeB=2,r=r.max,r.low=r.min,permutations=100)
```

---

get.tau *generalized version of* get.tau

Description

returns the relative probability (or odds) that points at some distance from an index point share some relationship with that point versus the probability (or odds) any point shares that relationship with that point.

Usage

```
get.tau(
  posmat,
  fun,
  r = 1,
  r.low = rep(0, length(r)),
  comparison.type = "representative",
  data.frame = TRUE
)
```

Arguments

posmat	a matrix with columns x, y and any other named columns columns needed by fun
fun	a function that takes in two rows of posmat and returns: 1. for pairs included in the numerator (and the denominator for independent data) 2. for pairs that should only be included in the denominator 3. for pairs that should be ignored all together Note that names from posmat are not preserved in calls to fun, so the columns of the matrix should be referenced numerically so this is not available to fun
r	the series of spatial distances (or there maximums) we are interested in
r.low	the low end of each range, 0 by default
comparison.type	what type of points are included in the comparison set. • "representative" if comparison set is representative of the underlying population • "independent" if comparison set is cases/events coming from an indepedent process
data.frame	logical indicating whether to return results as a data frame (default = TRUE)

Value

The tau value for each distance we look at. If comparison.type is "representative", this is:

tau = get.pi(posmat, fun, r, r.low)/get.pi(posmat,fun,infinity,0)

If comparison.type is "independent", this is:

tau = get.theta(posmat, fun, r, r.low)/get.theta(posmat,fun,infinity,0)

Author(s)

Justin Lessler and Henrik Salje

See Also

Other get.tau: get.tau.bootstrap(), get.tau.ci(), get.tau.permute(), get.tau.typed(), get.tau.typed.bootstrap(), get.tau.typed.permute()

Other spatialtau: get.pi(), get.theta()

Examples

```
data(DengueSimR01)
data(DengueSimR02)
data(DengueSimRepresentative)

r.max<-seq(20,1000,20)
r.min<-seq(0,980,20)
r.mid<-(r.max+r.min)/2

sero.type.func<-function(a,b,tlimit=20){
     if(a[5]==b[5]&(abs(a[3]-b[3])<=tlimit)){rc=1}
     else{rc=2}
     return(rc)
}

geno.type.func<-function(a,b,tlimit=20){
     if(a[4]==b[4]&(abs(a[3]-b[3])<=tlimit)){rc=1}
     else{rc=2}
     return(rc)
}

sero.type.rep.func<-function(a,b,tlimit=20){
     if(a[5]==1&b[5]==1&(abs(a[3]-b[3])<=tlimit)){rc=1}
     else{if(a[5]==1&b[5]==-999){rc=2}else{rc=3}}
     return(rc)
}

sero.tau.R01 <- get.tau(DengueSimR01, sero.type.func, r=r.max, r.low=r.min,
                        comparison.type="independent")
geno.tau.R01 <- get.tau(DengueSimR01, geno.type.func, r=r.max, r.low=r.min,
                        comparison.type="independent")

sero.tau.R02 <- get.tau(DengueSimR02, sero.type.func, r=r.max, r.low=r.min,
                        comparison.type="independent")
geno.tau.R02 <- get.tau(DengueSimR02, geno.type.func, r=r.max, r.low=r.min,
                        comparison.type="independent")

sero.tau.representative <- get.tau(DengueSimRepresentative, sero.type.rep.func,
                                   r=r.max, r.low=r.min, comparison.type="representative")

## R0 of 1
plot(r.mid,sero.tau.R01$tau,ylim=c(0.3,max(geno.tau.R01$tau)),log="y",
     cex.axis=1.25,col=rgb(t(col2rgb("blue")/255),alpha=0.6),
     xlab="Distance (m)",ylab="Tau",cex.main=0.9,lwd=2,type="l",las=1,cex.axis=0.75)
abline(h=1,lty=2)
```

```
abline(v=100,lty=1,lwd=2)
lines(r.mid,geno.tau.R01$tau,pch=20,col=rgb(t(col2rgb("dark green")/255),alpha=0.6),lwd=1)
lines(r.mid,sero.tau.representative$tau,pch=20,col=rgb(t(col2rgb("dark blue")/255),alpha=0.6),lty=2)
legend("topright",
       legend=c("Genotype",
                "Serotype",
                "Serotype (representative population)",
                "Maximum transmission distance"),
       lwd=1,col=c("dark green","blue","blue","black"),
       lty=c(1,1,2,1),bty="n")

## R0 of 2
plot(r.mid,sero.tau.R02$tau,ylim=c(0.3,max(geno.tau.R02)),log="y",
     cex.axis=1.25,col=rgb(t(col2rgb("blue")/255),alpha=0.6),
     xlab="Distance (m)",ylab="Tau",cex.main=0.9,lwd=2,type="l",las=1,cex.axis=0.75)
abline(h=1,lty=2)
abline(v=100,lty=1,lwd=2)
lines(r.mid,geno.tau.R02$tau,pch=20,col=rgb(t(col2rgb("dark green")/255),alpha=0.6),lwd=1)
legend("topright",
       legend=c("Genotype",
                "Serotype",
                "Maximum transmission distance"),
       lwd=1,col=c("dark green","blue","black"),lty=1,bty="n")
```

`get.tau.bootstrap` *Bootstrap* `get.tau` *values.*

Description

Runs `get.tau` on multiple bootstraps of the data. Is formulated such that the relationship between points and themselves will not be calculated

Usage

```
get.tau.bootstrap(
  posmat,
  fun,
  r = 1,
  r.low = rep(0, length(r)),
  boot.iter,
  comparison.type = "representative",
  data.frame = TRUE
)
```

Arguments

posmat	a matrix appropriate for input to get.tau
fun	a function appropriate as input to get.pi
r	the series of spatial distances wer are interested in
r.low	the low end of each range....0 by default
boot.iter	the number of bootstrap iterations
comparison.type	the comparison type to pass as input to get.pi
data.frame	logical indicating whether to return results as a data frame (default = TRUE)

Value

a matrix containing all bootstrapped values of tau for each distance interval

Author(s)

Justin Lessler and Henrik Salje

See Also

Other get.tau: get.tau(), get.tau.ci(), get.tau.permute(), get.tau.typed(), get.tau.typed.bootstrap(), get.tau.typed.permute()

Examples

```
#compare normally distributed with uniform points
x<-cbind(1,runif(100,-100,100), runif(100,-100,100))
x<-rbind(x, cbind(2,rnorm(100,0,20), rnorm(100,0,20)))
colnames(x) <- c("type","x","y")

fun<-function(a,b) {
    if(a[1]!=2) return(3)
    if (b[1]==2) return(1)
    return(2)
}

r.max<-seq(10,100,10)
r.min<-seq(0,90,10)
r.mid <- (r.max+r.min)/2

tau<-get.tau(x,fun,r=r.max,r.low=r.min)
tau.boot<-get.tau.bootstrap(x,fun,r=r.max,r.low=r.min,boot.iter=50)

tau.ci<-apply(tau.boot[,-(1:2)],1,quantile,probs=c(0.25,0.75))

plot(r.mid, tau$tau ,ylim=c(min(tau.ci),max(tau.ci)), type="l", log="y")
lines(c(0,100),c(1,1), lty=3, col="grey")
lines(r.mid, tau.ci[1,] , lty=2)
```

```
lines(r.mid, tau.ci[2,] , lty=2)
```

`get.tau.ci` *Bootstrap confidence interval for the* `get.tau` *values*

Description

Wrapper to `get.tau.bootstrap` that takes care of calulating the confidence intervals based on the bootstrapped values

Usage

```
get.tau.ci(
  posmat,
  fun,
  r = 1,
  r.low = rep(0, length(r)),
  boot.iter = 1000,
  comparison.type = "representative",
  ci.low = 0.025,
  ci.high = 0.975,
  data.frame = TRUE
)
```

Arguments

`posmat`	a matrix appropriate for input to `get.tau`
`fun`	a function appropriate as input to `get.pi`
`r`	the series of spatial distances we are interested in
`r.low`	the low end of each range....0 by default
`boot.iter`	the number of bootstrap iterations
`comparison.type`	the comparison type to pass to get.tau
`ci.low`	the low end of the ci...0.025 by default
`ci.high`	the high end of the ci...0.975 by default
`data.frame`	logical indicating whether to return results as a data frame (default = TRUE)

Value

a data frame with the point estimate of tau and its low and high confidence interval at each distance

Author(s)

Justin Lessler and Henrik Salje

See Also

Other get.tau: get.tau(), get.tau.bootstrap(), get.tau.permute(), get.tau.typed(), get.tau.typed.bootstrap(), get.tau.typed.permute()

Examples

```
#compare normally distributed with uniform points
x<-cbind(1,runif(100,-100,100), runif(100,-100,100))
x<-rbind(x, cbind(2,rnorm(100,0,20), rnorm(100,0,20)))
colnames(x) <- c("type","x","y")

fun<-function(a,b) {
     if(a[1]!=2) return(3)
     if (b[1]==2) return(1)
     return(2)
}

r.max<-seq(10,100,10)
r.min<-seq(0,90,10)
r.mid <- (r.max+r.min)/2

tau <- get.tau.ci(x,fun,r=r.max,r.low=r.min,boot.iter=50)

plot(r.mid, tau$pt.est, ylim=c(1/max(tau[,3:5]), max(tau[,3:5])), type="l", log="y",
     xlab="Distance", ylab="Tau")
lines(r.mid, tau$ci.low , lty=2)
lines(r.mid, tau$ci.high, lty=2)
lines(c(0,100),c(1,1), lty=3, col="grey")
```

---

get.tau.permute *get the null distribution of the* get.tau *function*

Description

Does permutations to calculate the null distribution of get pi if there were no spatial dependence. Randomly reassigns coordinates to each observation permutations times

Usage

```
get.tau.permute(
  posmat,
  fun,
  r = 1,
  r.low = rep(0, length(r)),
  permutations,
  comparison.type = "representative",
```

```
  data.frame = TRUE
)
```

Arguments

posmat	a matrix appropriate for input to get.tau
fun	a function appropriate for input to get.tau
r	the series of spatial distances we are interested in
r.low	the low end of each range....0 by default
permutations	the number of permute iterations
comparison.type	the comparison type to pass as input to get.pi
data.frame	logical indicating whether to return results as a data frame (default = TRUE)

Value

tau values for all the distances we looked at

Author(s)

Justin Lessler and Henrik Salje

See Also

Other get.tau: get.tau(), get.tau.bootstrap(), get.tau.ci(), get.tau.typed(), get.tau.typed.bootstrap(), get.tau.typed.permute()

Examples

```
#compare normally distributed with uniform points
x<-cbind(1,runif(100,-100,100), runif(100,-100,100))
x<-rbind(x, cbind(2,rnorm(100,0,20), rnorm(100,0,20)))
colnames(x) <- c("type","x","y")

fun<-function(a,b) {
    if(a[1]!=2) return(3)
    if (b[1]==2) return(1)
    return(2)
}

r.max<-seq(10,100,10)
r.min<-seq(0,90,10)
r.mid <- (r.max+r.min)/2

tau<-get.tau(x,fun,r=r.max,r.low=r.min,comparison.type = "independent")
tau.null<-get.tau.permute(x,fun,r=r.max,r.low=r.min,permutations=50,comparison.type = "independent")

null.ci<-apply(tau.null[,-(1:2)],1,quantile,probs=c(0.25,0.75))
```

```
plot(r.mid, tau$tau, ylim=c(1/max(tau$tau),max(tau$tau)), type="l", log="y")
lines(c(0,100),c(1,1), lty=3, col="grey")
lines(r.mid, null.ci[1,] , lty=2)
lines(r.mid, null.ci[2,] , lty=2)
```

`get.tau.typed` *Optimized version of* `get.tau` *for typed data*

Description

Version of th e `get.tau` function that is optimized for statically typed data. That is data where we want the relationship between points of type A and points of type B

Usage

```
get.tau.typed(
  posmat,
  typeA = -1,
  typeB = -1,
  r = 1,
  r.low = rep(0, length(r)),
  comparison.type = "representative",
  data.frame = TRUE
)
```

Arguments

`posmat`	a matrix with columns type, x and y
`typeA`	the "from" type that we are interested in, -1 is wildcard
`typeB`	the "to" type that we are interested i, -1 is wildcard
`r`	the series of spatial distances we are interested in
`r.low`	the low end of each range....0 by default
`comparison.type`	what type of points are included in the comparison set. • "representative" if comparison set is representative of the underlying population • "independent" if comparison set is cases/events coming from an indepedent process
`data.frame`	logical indicating whether to return results as a data frame (default = TRUE)

Value

data frame of tau values for all the distances

Author(s)

Justin Lessler and Henrik Salje

See Also

Other get.tau: `get.tau()`, `get.tau.bootstrap()`, `get.tau.ci()`, `get.tau.permute()`, `get.tau.typed.bootstrap()`, `get.tau.typed.permute()`

Examples

```
data(DengueSimR02)

r.max<-seq(20,1000,20)
r.min<-seq(0,980,20)
r.mid<-(r.max+r.min)/2

#Lets see if there's a difference in spatial dependence by time case occurs
type<-2-(DengueSimR02[,"time"]<120)
tmp<-cbind(DengueSimR02,type=type)

typed.tau<-get.tau.typed(tmp,typeA=1,typeB=2,r=r.max,r.low=r.min,comparison.type = "independent")

plot(r.mid,typed.tau$tau,log="y",cex.axis=1.25,
     xlab="Distance (m)",ylab="Tau",cex.main=0.9,lwd=2,type="l")
abline(h=1,lty=2)
```

---

get.tau.typed.bootstrap

runs bootstrapping for `get.tau.typed`

Description

runs bootstrapping for `get.tau.typed`

Usage

```
get.tau.typed.bootstrap(
  posmat,
  typeA = -1,
  typeB = -1,
  r = 1,
  r.low = rep(0, length(r)),
  boot.iter,
  comparison.type = "representative",
  data.frame = TRUE
)
```

Arguments

posmat	a matrix with columns type, x and y
typeA	the "from" type that we are interested in, -1 is wildcard
typeB	the "to" type that we are interested i, -1 is wildcard
r	the series of spatial distances we are interested in
r.low	the low end of each range....0 by default
boot.iter	the number of bootstrap iterations
comparison.type	what type of points are included in the comparison set. • "representative" if comparison set is representative of the underlying population • "independent" if comparison set is cases/events coming from an independent process
data.frame	logical indicating whether to return results as a data frame (default = TRUE)

Value

tau values for all the distances we looked at

Author(s)

Justin Lessler and Henrik Salje

See Also

Other get.tau: get.tau(), get.tau.bootstrap(), get.tau.ci(), get.tau.permute(), get.tau.typed(), get.tau.typed.permute()

Examples

```
data(DengueSimulationR02)

r.max<-seq(20,1000,20)
r.min<-seq(0,980,20)
r.mid<-(r.max+r.min)/2

# Lets see if there's a difference in spatial dependence between those that occurred
# late versus early in the outbreak

type <- 2 - (DengueSimR02[,"time"] < 120)
tmp <- cbind(DengueSimR02, type=type)

typed.tau <- get.tau.typed(tmp, typeA=1, typeB=2, r=r.max, r.low=r.min,
                           comparison.type = "independent")

typed.tau.type.bs <- get.tau.typed.bootstrap(tmp, typeA=1, typeB=2, r=r.max, r.low=r.min,
                                        boot.iter=100, comparison.type = "independent")
```

```
ci <- apply(typed.tau.type.bs[,-(1:2)], 1, quantile, probs=c(0.025,0.975))

plot(r.mid, typed.tau$tau, log="y",
     ylim=c(0.1,4), cex.axis=1.25,
     xlab="Distance (m)", ylab="Tau",
     cex.main=0.9, lwd=2, type="n")
abline(h=1,lty=1)
lines(r.mid,typed.tau$tau,pch=20,col=1,lwd=3)
lines(r.mid, ci[1,] , lty=2)
lines(r.mid, ci[2,] , lty=2)
```

`get.tau.typed.permute` *get the null distribution for the* `get.tau.typed` *function*

Description

get the null distribution for the `get.tau.typed` function

Usage

```
get.tau.typed.permute(
  posmat,
  typeA = -1,
  typeB = -1,
  r = 1,
  r.low = rep(0, length(r)),
  permutations,
  comparison.type = "representative",
  data.frame = TRUE
)
```

Arguments

`posmat`	a matrix with columns type, x and y
`typeA`	the "from" type that we are interested in, -1 is wildcard
`typeB`	the "to" type that we are interested i, -1 is wildcard
`r`	the series of spatial distances we are interested in
`r.low`	the low end of each range....0 by default
`permutations`	the number of permute iterations
`comparison.type`	what type of points are included in the comparison set. • "representative" if comparison set is representative of the underlying population

- "independent" if comparison set is cases/events coming from an indepedent process

`data.frame` logical indicating whether to return results as a data frame (default = TRUE)

Value

a matrix with permutation tau values for each distance specified

Author(s)

Justin Lessler and Henrik Salje

See Also

Other get.tau: `get.tau()`, `get.tau.bootstrap()`, `get.tau.ci()`, `get.tau.permute()`, `get.tau.typed()`, `get.tau.typed.bootstrap()`

Examples

```
data(DengueSimulationR02)

r.max<-seq(20,1000,20)
r.min<-seq(0,980,20)
r.mid<-(r.max+r.min)/2

#Lets see if there's a difference in spatial dependence by time case occurs
type <- 2 - (DengueSimR02[,"time"] < 120)
tmp <- cbind(DengueSimR02, type=type)

typed.tau <- get.tau.typed(tmp, typeA=1, typeB=2, r=r.max, r.low=r.min,
                           comparison.type = "independent")

typed.tau.type.null<-get.tau.typed.permute(tmp, typeA=1, typeB=2, r=r.max, r.low=r.min,
                                     permutations=100, comparison.type = "independent")

null.ci <- apply(typed.tau.type.null[,-(1:2)], 1, quantile, probs=c(0.025,0.975))

plot(r.mid, typed.tau$tau, ylim=c(0.3,4), log="y", cex.axis=1.25,
     xlab="Distance (m)", ylab="Tau", cex.main=0.9, lwd=2, type="n")
abline(h=1,lty=1)
lines(r.mid,typed.tau$tau,pch=20,col=1,lwd=3)
lines(r.mid, null.ci[1,] , lty=2)
lines(r.mid, null.ci[2,] , lty=2)
```

`get.theta` *Generalized version of* `get.theta`

Description

Generalized version of the `get.theta` function that takes in an arbitrary function and returns the odds that a point within a particular range of a point of interest shares the relationship specified by the passed in function with that point.

Usage

```
get.theta(posmat, fun, r = 1, r.low = rep(0, length(r)), data.frame = TRUE)
```

Arguments

`posmat`	a matrix with columns x, y and any other named columns columns needed by fun
`fun`	a function that takes in two rows of posmat and returns: 1. for pairs that are (potentially) related 2. for pairs that are unrelated 3. for pairs that should be ignored all together Note that names from `posmat` are not preserved in calls to `fun`, so the columns of the matrix should be referenced numerically so this is not available to the fun
`r`	the series of spatial distances (or there maximums) we are interested in
`r.low`	the low end of each range, 0 by default
`data.frame`	logical indicating whether to return results as a data frame (default = TRUE)

Value

theta value for each distance range that we look at. Where:

$$\theta(d_1, d_2) = \frac{\sum \mathbf{1} d_{ij} \in [d_1, d_2)) \mathbf{1}(f(i,j) = 1)}{\sum \sum \mathbf{1} d_{ij} \in [d_1, d_2)) \mathbf{1}(f(i,j) = 2)}$$

Author(s)

Justin Lessler and Henrik Salje

See Also

Other get.theta: `get.theta.bootstrap()`, `get.theta.ci()`, `get.theta.permute()`, `get.theta.typed()`, `get.theta.typed.bootstrap()`, `get.theta.typed.permute()`

Other spatialtau: `get.pi()`, `get.tau()`

Examples

```
data(DengueSimR02)

r.max<-seq(20,1000,20)
r.min<-seq(0,980,20)

sero.type.func<-function(a,b,tlimit=20){
  if(a[5]==b[5]&(abs(a[3]-b[3])<=tlimit)){rc=1}
  else{rc=2}
  return(rc)
}

sero.theta<-get.theta(DengueSimR02,sero.type.func,r=r.max,r.low=r.min)
```

---

get.theta.bootstrap *Bootstrap* get.theta *values.*

Description

Runs get.theta on multiple bootstraps of the data. Is formulated such that the relationships between points and themselves will not be calculated.

Usage

```
get.theta.bootstrap(
  posmat,
  fun,
  r = 1,
  r.low = rep(0, length(r)),
  boot.iter = 500,
  data.frame = TRUE
)
```

Arguments

posmat	a matrix with columns type, x and y
fun	the function to decide relationships
r	the series of spatial distances we are interested in
r.low	the low end of each range. 0 by default
boot.iter	the number of bootstrap iterations
data.frame	logical indicating whether to return results as a data frame (default = TRUE)

Value

theta values for all the distances we looked at

Note

In each bootstrap iteration N observations are drawn from the existing data with replacement. To avoid errors in inference resulting from the same observatin being compared with itself in the bootstrapped data set, original indices are perserved, and pairs of points in the bootstrapped dataset with the same original index are ignored.

Author(s)

Justin Lessler and Henrik Salje

See Also

Other get.theta: `get.theta()`, `get.theta.ci()`, `get.theta.permute()`, `get.theta.typed()`, `get.theta.typed.bootstrap()`, `get.theta.typed.permute()`

Examples

```
#compare normally distributed with uniform points
x<-cbind(1,runif(100,-100,100), runif(100,-100,100))
x<-rbind(x, cbind(2,rnorm(100,0,20), rnorm(100,0,20)))
colnames(x) <- c("type","x","y")

fun<-function(a,b) {
    if(a[1]!=2) return(3)
    if (b[1]==2) return(1)
    return(2)
}

r.max<-seq(10,100,10)
r.min<-seq(0,90,10)
r.mid <- (r.max+r.min)/2


theta<-get.theta(x,fun,r=r.max,r.low=r.min)
theta.boot<-get.theta.bootstrap(x,fun,r=r.max,r.low=r.min,boot.iter=100)

theta.ci<-apply(theta.boot[,-(1:2)],1,quantile,probs=c(0.25,0.75))

plot(r.mid, theta$theta , type="l")
lines(r.mid, theta.ci[1,] , lty=2)
lines(r.mid, theta.ci[2,] , lty=2)
```

`get.theta.ci` *Calculate bootstrapped confidence intervals for* `get.theta` *values.*

Description

Wrapper to `get.theta.bootstrap` that takes care of calculating the confience intervals based on the bootstrapped values.

Usage

```
get.theta.ci(
  posmat,
  fun,
  r = 1,
  r.low = rep(0, length(r)),
  boot.iter = 1000,
  ci.low = 0.025,
  ci.high = 0.975,
  data.frame = TRUE
)
```

Arguments

`posmat`	a matrix with columns type, x and y
`fun`	the function to decide relationships
`r`	the series of spatial distances we are interested in
`r.low`	the low end of each range. 0 by default
`boot.iter`	the number of bootstrap iterations
`ci.low`	the low end of the ci...0.025 by default
`ci.high`	the high end of the ci...0.975 by default
`data.frame`	logical indicating whether to return results as a data frame (default = TRUE)

Value

a matrix with a row for the high and low values and a column per distance

Author(s)

Justin Lessler

See Also

Other get.theta: `get.theta()`, `get.theta.bootstrap()`, `get.theta.permute()`, `get.theta.typed()`, `get.theta.typed.bootstrap()`, `get.theta.typed.permute()`

Examples

```
#compare normally distributed with uniform points
x<-cbind(1,runif(100,-100,100), runif(100,-100,100))
x<-rbind(x, cbind(2,rnorm(100,0,20), rnorm(100,0,20)))
colnames(x) <- c("type","x","y")

fun<-function(a,b) {
    if(a[1]!=2) return(3)
    if (b[1]==2) return(1)
    return(2)
}

r.max<-seq(10,100,10)
r.min<-seq(0,90,10)
r.mid <- (r.max+r.min)/2

theta<-get.theta(x,fun,r=r.max,r.low=r.min)
theta.ci<-get.theta.ci(x,fun,r=r.max,r.low=r.min,boot.iter=100)

plot(r.mid, theta$theta, type="l")
lines(r.mid, theta.ci[,2] , lty=2)
lines(r.mid, theta.ci[,3] , lty=2)
```

---

get.theta.permute *get the null distribution of the* get.theta *function*

Description

Does permutations to calculate the null distribution of get theta if there were no spatial dependence. Randomly reassigns coordinates to each observation permutations times

Usage

```
get.theta.permute(
  posmat,
  fun,
  r = 1,
  r.low = rep(0, length(r)),
  permutations,
  data.frame = TRUE
)
```

Arguments

posmat	a matrix with columns type, x and y

fun	the function to evaluate
r	the series of spatial distances we are interested in
r.low	the low end of each range....0 by default
permutations	the number of permute iterations
data.frame	logical indicating whether to return results as a data frame (default = TRUE)

Value

theta values for all the distances we looked at

See Also

Other get.theta: get.theta(), get.theta.bootstrap(), get.theta.ci(), get.theta.typed(), get.theta.typed.bootstrap(), get.theta.typed.permute()

Examples

```
#compare normally distributed with uniform points
x<-cbind(1,runif(100,-100,100), runif(100,-100,100))
x<-rbind(x, cbind(2,rnorm(100,0,20), rnorm(100,0,20)))
colnames(x) <- c("type","x","y")

fun<-function(a,b) {
    if(a[1]!=2) return(3)
    if (b[1]==2) return(1)
    return(2)
}

r.max<-seq(10,100,10)
r.min<-seq(0,90,10)
r.mid <- (r.max+r.min)/2

theta<-get.theta(x,fun,r=r.max,r.low=r.min)
theta.null<-get.theta.permute(x,fun,r=r.max,r.low=r.min,permutations=100)

null.ci<-apply(theta.null[,-(1:2)],1,quantile,probs=c(0.25,0.75))

plot(r.mid, theta$theta , type="l")
lines(r.mid, null.ci[1,] , lty=2)
lines(r.mid, null.ci[2,] , lty=2)
```

`get.theta.typed` *Optimized version of* `get.theta` *for typed data.*

Description

Version of the `get.theta` function that is optimized for statically typed data. That is data where we are interested in the odds that points within some distance of points of typeA are of typeB.

Usage

```
get.theta.typed(
  posmat,
  typeA = -1,
  typeB = -1,
  r = 1,
  r.low = rep(0, length(r)),
  data.frame = TRUE
)
```

Arguments

`posmat`	a matrix with columns type, x and y
`typeA`	the "from" type that we are interested in, -1 is wildcard
`typeB`	the "to" type that we are interested i, -1 is wildcard
`r`	the series of spatial distances wer are interested in
`r.low`	the low end of each range....0 by default
`data.frame`	logical indicating whether to return results as a data frame (default = TRUE)

Value

theta values for all the distances we looked at

Author(s)

Justin Lessler and Henrik Salje

See Also

Other get.theta: `get.theta()`, `get.theta.bootstrap()`, `get.theta.ci()`, `get.theta.permute()`, `get.theta.typed.bootstrap()`, `get.theta.typed.permute()`

Examples

```
data(DengueSimR02)

r.max<-seq(20,1000,20)
r.min<-seq(0,980,20)

#Lets see if there's a difference in spatial dependence by time case occurs
type<-2-(DengueSimR02[,"time"]<120)
tmp<-cbind(DengueSimR02,type=type)

typed.theta.R01<-get.theta.typed(tmp,typeA=2,typeB=2,r=r.max,r.low=r.min)
```

---

get.theta.typed.bootstrap

runs bootstrapping on `get.theta.typed`

Description

Bootstraps typed pi values. Makes sure distances between a sample and another draw of itself are left out

Usage

```
get.theta.typed.bootstrap(
  posmat,
  typeA = -1,
  typeB = -1,
  r = 1,
  r.low = rep(0, length(r)),
  boot.iter,
  data.frame = TRUE
)
```

Arguments

`posmat`	a matrix with columns type, x and y
`typeA`	the "from" type that we are interested in, -1 is wildcard
`typeB`	the "to" type that we are interested i, -1 is wildcard
`r`	the series of spatial distances we are interested in
`r.low`	the low end of each range....0 by default
`boot.iter`	the number of bootstrap iterations
`data.frame`	logical indicating whether to return results as a data frame (default = TRUE)

Value

theta values for all the distances we looked at

See Also

Other get.theta: `get.theta()`, `get.theta.bootstrap()`, `get.theta.ci()`, `get.theta.permute()`, `get.theta.typed()`, `get.theta.typed.permute()`

Examples

```
data(DengueSimR01)

r.max<-seq(20,1000,20)
r.min<-seq(0,980,20)

#Lets see if there's a difference in spatial dependence by time case occurs
type<-2-(DengueSimR01[,"time"]<75)
tmp<-cbind(DengueSimR01,type=type)

typed.theta.bs<-get.theta.typed.bootstrap(tmp,typeA=1,typeB=2,r=r.max,r.low=r.min,boot.iter=100)
```

---

get.theta.typed.permute

get the null distribution of the get.theta.typed function

Description

Does permutations to calculate the null distribution of get theta if there were no spatial dependence. Randomly reassigns coordinates to each observation permutations times

Usage

```
get.theta.typed.permute(
  posmat,
  typeA = -1,
  typeB = -1,
  r = 1,
  r.low = rep(0, length(r)),
  permutations,
  data.frame = TRUE
)
```

Arguments

posmat	a matrix with columns type, x and y
typeA	the "from" type that we are interested in, -1 is wildcard
typeB	the "to" type that we are interested i, -1 is wildcard
r	the series of spatial distances we are interested in
r.low	the low end of each range....0 by default
permutations	the number of permute iterations
data.frame	logical indicating whether to return results as a data frame (default = TRUE)

Value

theta values for all the distances we looked at

Author(s)

Justin Lessler and Henrik Salje

See Also

Other get.theta: get.theta(), get.theta.bootstrap(), get.theta.ci(), get.theta.permute(), get.theta.typed(), get.theta.typed.bootstrap()

Examples

```
data(DengueSimR01)

r.max<-seq(20,1000,20)
r.min<-seq(0,980,20)

#Lets see if there's a difference in spatial dependence by time case occurs
type<-2-(DengueSimR01[,"time"]<75)
tmp<-cbind(DengueSimR01,type=type)

typed.theta.R01<-get.theta.typed(tmp,typeA=1,typeB=2,r=r.max,r.low=r.min)
typed.theta.type.null<-get.theta.typed.permute(tmp, typeA=1, typeB=2,
                                               r=r.max, r.low=r.min, permutations=100)
```

`get.transdist.theta` *Get weights of transmission distance theta*

Description

This function estimates the weights of each theta (number of transmission events separating cases at two time points). A randomized transmission tree is drawn and the number of transmission events separating cases at two time points is calculated based on probabilies found in the Wallinga-Teunis matrix.

Usage

```
get.transdist.theta(
  wal.teun.mat,
  cases,
  gen.t.mean,
  max.sep,
  ret.theta.mat = FALSE
)
```

Arguments

`wal.teun.mat`	a Wallinga-Teunis matrix produced by the `est.wt.matrix` function
`cases`	a vector of case times for each case
`gen.t.mean`	the mean generation time of the infecting pathogen
`max.sep`	maximum number of transmission events allowed between two cases
`ret.theta.mat`	logical value which returns the matrix of estimated theta values (default = FALSE)

Value

a three-dimensional array containing normalized theta weights. Columns and rows represent unique case times. The third dimension is the number of transmission events between two cases.

Author(s)

John Giles, Justin Lessler, and Henrik Salje

References

Salje H, Cummings DAT and Lessler J (2016). "Estimating infectious disease transmission distances using the overall distribution of cases." Epidemics, 17, pp. 10–18. ISSN 1755-4365, doi: 10.1016/j.epidem.2016.10.001.

See Also

Other transdist: `est.transdist()`, `est.transdist.bootstrap.ci()`, `est.transdist.temporal()`, `est.transdist.temporal.bootstrap.ci()`, `est.transdist.theta.weights()`

Examples

```
case.times <- c(1,2,2,3,3)
gen <- c(0, 2/3, 1/3, 0, 0)
t.density <- gen/sum(gen)

gen.time <- 2 # mean generation time

wt <- est.wt.matrix(case.times=case.times, gen.t.dist=t.density)

ngen <- round((max(case.times) - min(case.times)) / gen.time) + 1 # Number of generations

a <- get.transdist.theta(wal.teun.mat=wt,
                         cases=case.times,
                         gen.t.mean=gen.time,
                         max.sep=ngen*2)
```

---

sim.epidemic *Simulation of an epidemic in space and time*

Description

A function which simulates the spatial spread of infections through time given the reproductive number (R), a function describing the spatial transmission kernel (trans.kern.func), and the mean and standard deviation of the generation time distribution (gen.t.mean and gen.t.sd) for the infecting pathogen. The function returns the location (x, y) and time (t) for each case of infection in the simulation.

Usage

```
sim.epidemic(
  R,
  gen.t.mean,
  gen.t.sd,
  trans.kern.func,
  tot.generations = 10,
  min.cases = 0,
  max.try = 1000
)
```

Arguments

R	a scalar or a vector of length tot.generations providing the reproductive number for the epidemic. If scalar, the R value is constant. If a vector, the R value varies according to each generation in the vector.
gen.t.mean	mean of generation time

gen.t.sd	standard deviation of the generation time (assumed to be normally distributed)
trans.kern.func	a function for the transmission kernel that takes n as an arguement. Function and associated parameters must be given in a list object.
tot.generations	the total number of generations in the epidemic, where the index case (x,y,t = [0,0,0]) is considered generation zero (default = 10)
min.cases	the minimum number of cases in the epidemic (default = 0)
max.try	maximum number of tries to acheive the minimum number of cases (default = 1000)

Value

a numerical matrix with three columns giving the coordinates x and y, and time t of simulated cases

Author(s)

John Giles, Justin Lessler, and Henrik Salje

Examples

```
set.seed(1)

dist.func <- alist(n=1, a=1/100, rexp(n, a)) # Exponential transmission kernel with mean = sd = 100

# Simulate epidemic with constant R value
a <- sim.epidemic(R=1.5,
             gen.t.mean=7,
             gen.t.sd=2,
             tot.generations=15,
             min.cases=100,
             trans.kern.func=dist.func)

sim.plot(a)

# Simulate an epidemic with variable R value
r1 <- 2
r2 <- 0.25
tg <- 25
R <- seq(r1, r2, (r2 -r1)/(tg - 1))

b <- sim.epidemic(R=R,
             gen.t.mean=7,
             gen.t.sd=2,
             tot.generations=tg,
             min.cases=100,
             trans.kern.func=dist.func)

sim.plot(b)
```

`sim.plot` *Plot output of simulated epidemic*

Description

A simple visualization function which plots the location of the index case and the spatial distribution of subsequent cases, and the epidemic curve showing the case count over time.

Usage

```
sim.plot(sim)
```

Arguments

`sim`	a three-column matrix object produced by the `sim.epidemic` function

Value

A two-panel plotted object

Author(s)

John Giles, Justin Lessler, and Henrik Salje

Index

∗ **datasets**
DengueSimR01, 2
DengueSimR02, 3
DengueSimRepresentative, 4
∗ **est.wt**
est.transdist, 4
est.wt.matrix, 15
est.wt.matrix.weights, 16
∗ **get.pi**
get.pi, 20
get.pi.bootstrap, 21
get.pi.ci, 23
get.pi.permute, 24
get.pi.typed, 26
get.pi.typed.bootstrap, 27
get.pi.typed.permute, 28
∗ **get.tau**
get.tau, 29
get.tau.bootstrap, 32
get.tau.ci, 34
get.tau.permute, 35
get.tau.typed, 37
get.tau.typed.bootstrap, 38
get.tau.typed.permute, 40
∗ **get.theta**
get.theta, 42
get.theta.bootstrap, 43
get.theta.ci, 45
get.theta.permute, 46
get.theta.typed, 48
get.theta.typed.bootstrap, 49
get.theta.typed.permute, 50
∗ **spatialtau**
get.pi, 20
get.tau, 29
get.theta, 42
∗ **transdist**
est.transdist, 4
est.transdist.bootstrap.ci, 6
est.transdist.temporal, 8
est.transdist.temporal.bootstrap.ci, 11
est.transdist.theta.weights, 13
get.transdist.theta, 52

DengueSimR01, 2
DengueSimR02, 3
DengueSimRepresentative, 4

est.transdist, 4, *8*, *10*, *12*, *14*, *16, 17*, *52*
est.transdist.bootstrap.ci, *5*, 6, *10*, *12*, *14*, *52*
est.transdist.temporal, *5*, *8*, 8, *12*, *14*, *52*
est.transdist.temporal.bootstrap.ci, *5*, *8*, *10*, 11, *14*, *52*
est.transdist.theta.weights, *5*, *8*, *10*, *12*, 13, *52*
est.wt.matrix, *5*, 15, *17*
est.wt.matrix.weights, *5*, *16*, 16

get.cross.K, 17
get.cross.PCF, 19
get.pi, 20, *22, 23*, *25, 26*, *28, 29*, *31*, *42*
get.pi.bootstrap, *21*, 21, *23*, *25, 26*, *28, 29*
get.pi.ci, *21, 22*, 23, *25, 26*, *28, 29*
get.pi.permute, *21–23*, 24, *26*, *28, 29*
get.pi.typed, *21–23*, *25*, 26, *28, 29*
get.pi.typed.bootstrap, *21–23*, *25, 26*, 27, *29*
get.pi.typed.permute, *21–23*, *25, 26*, *28*, 28
get.tau, *21*, 29, *33*, *35, 36*, *38, 39*, *41, 42*
get.tau.bootstrap, *31*, 32, *35, 36*, *38, 39*, *41*
get.tau.ci, *31*, *33*, 34, *36*, *38, 39*, *41*
get.tau.permute, *31*, *33*, *35*, 35, *38, 39*, *41*
get.tau.typed, *31*, *33*, *35, 36*, 37, *39*, *41*
get.tau.typed.bootstrap, *31*, *33*, *35, 36*, *38*, 38, *41*

get.tau.typed.permute, *31*, *33*, *35, 36*, *38, 39*, 40
get.theta, *21*, *31*, 42, *44, 45*, *47, 48*, *50, 51*
get.theta.bootstrap, *42*, 43, *45*, *47, 48*, *50, 51*
get.theta.ci, *42*, *44*, 45, *47, 48*, *50, 51*
get.theta.permute, *42*, *44, 45*, 46, *48*, *50, 51*
get.theta.typed, *42*, *44, 45*, *47*, 48, *50, 51*
get.theta.typed.bootstrap, *42*, *44, 45*, *47, 48*, 49, *51*
get.theta.typed.permute, *42*, *44, 45*, *47, 48*, *50*, 50
get.transdist.theta, *5*, *8*, *10*, *12*, *14*, 52

Kcross, *17, 18*

pcf, *19*

sim.epidemic, 53
sim.plot, 55