Package 'IndTestPP'

August 1, 2016

Type Package

Title Tests of Independence Between Point Processes in Time

Version 1.0

Date 2016-08-01

Author Ana C. Cebrian <acebrian@unizar.es>

Maintainer Ana C. Cebrian <acebrian@unizar.es>

Imports parallel, stats

Description Several parametric and non-parametric tests and measures to check independence between two or more (homogeneous or nonhomogeneous) point processes in time are provided. Tools for simulating point processes in one dimension with different types of dependence are also implemented.

License GPL

NeedsCompilation no

Repository CRAN

Date/Publication 2016-08-01 23:46:57

R topics documented:

IndTestPP-package 2
BarTxTn 3
ComplPos.fun 4
CondTest.fun 5
CPSPpoints.fun 7
CPSPPOTevents.fun 8
depchi.fun 10
DepMarkedNHPP.fun 12
DepNHCPSP.fun 13
DepNHNeyScot.fun 15
DepNHPPqueue.fun 17
DistObs.fun 19
DistShift.fun 20
DistSim.fun 22

DutilleulPlot.fun 24
IndNHNeyScot.fun 25
IndNHPP.fun 27
nearestdist.fun 28
NHD.fun 29
NHF.fun 31
NHJ.fun 33
NHK.fun 37
simHPc.fun 40
simNHPc.fun 41
SpecGap.fun 42
TestIndLS.fun 43
TestIndNH.fun 45
uniongentri.fun 48

Index **50**

`IndTestPP-package` *Tests of Independence Between Point Processes in Time*

Description

It provides several parametric and non-parametric tests to check the independence between two or more homogeneous and nonhomogeneous point processes. In addition, it provides tools for simulating point processes in one dimension, usually time, with different types of dependence, in particular: common Poisson shock processes, a network of queues with exponential arrivals, multivariate Neyman-Scott process with dependent cluster centers, and a Poisson process with dependent marks.

Details

Package:	IndTestPP
Type:	Package
Version:	1.0
Date:	2016-07-25
License:	GPL (>=2)

Author(s)

Ana C. Cebrian Maintainer: Ana C. Cebrian <acebrian@unizar.es>

`BarTxTn` *Barcelona temperature data*

Description

Barcelona daily temperature series during the summer months (May, June, July, August and September) from 1951 to 2004.

Usage

```
data(BarTxTn)
```

Details

Variables

dia: Postion of the day in the year, from 121 (1st of May) to 253 (30th of September).

mes: Month of the year, from 5 to 9.

ano: Year, from 1951 to 2004.

diames: Position of the day in the month, from 1 to 30 or 31.

Tx: Daily maximum temperature.

Tn: Daily minimum temperature.

Txm31: Local maximum temperature signal. Lowess of Tx with a centered window of 31 days.

Txm15: Local maximum temperature signal. Lowess of Tx with a centered window of 15 days.

Tnm31: Local minimum temperature signal. Lowess of Tn with a centered window of 31 days.

Tnm15: Local minimum temperature signal. Lowess of Tn with a centered window of 15 days.

TTx: Long term maximum temperature signal. Lowess of Tx with a centered 40% window.

TTn: Long term minimum temperature signal. Lowess of Tn with a centered 40% window.

References

Cebrian, A.C., Abaurrea, J. and Asin, J. (2015). NHPoisson: An R Package for Fitting and Validating Nonhomogeneous Poisson Processes. *Journal of Statistical Software*, 64(6), 1-24.

Examples

```
data(BarTxTn)
```

`ComplPos.fun` *Changes the format of a vector of the occurrence times in a point process*

Description

It changes the format ofthe vector of the occurrence points in a point process. It builds a vector of length T, the length of the observation period, which takes value 0 at the non occurrence times and the position value, or 1, at the occurence times.

Usage

```
ComplPos.fun(pos, T, type='Pos')
```

Arguments

`pos`	Numeric vector. Occurrence times
`T`	Integer. Length of the observation period
`type`	Character string, 'Pos' or 'Bin'. Type of the new format

Details

This function changes the format of the occurrence points in a point process. The new format is useful when several point processes, in the same observation period, must be specfied; for example, in function `NHJ.fun` or `NHD.fun`, where the occurrence times of different point processes must be introduced as a matrix. Since the number of occurrences in each process can be different, in the new format, occurrences in each process are specified as a vector of length T, which takes value 0 at non occurrence times and the time position (if type='Pos') or 1 (if type='Bin') at the occurrence times.

Value

`Npos`	Numeric vector of lenght T containing the occurrence times in the new format.

See Also

`NHD.fun`, `NHF.fun`, `NHK.fun`, `NHJ.fun`

Examples

```
pos<-c(4,15,18,34,36,67,98)
Npos<-ComplPos.fun(pos, T=100)
```

CondTest.fun *Conditional Test of independence between two Poisson process*

Description

It calculates a test of independence between two Poisson process (PP), based on the analysis of the occurrences in the second process, given that there is an occurrence in the first one. Two different approaches to calculate the p-value are implemented.

It calls the auxiliary function `calcNmu`, not intended for the users.

Usage

```
CondTest.fun(pos1, pos2, lambda2, r, changer = TRUE, type = "All", plotRes = FALSE, ...)
```

Arguments

`pos1`	Numeric vector. Occurrence points in the first PP, N_1
`pos2`	Numeric vector. Occurrence points in the second PP, N_2
`lambda2`	Numeric vector. Intensity in each time in N_2
`r`	Numerical value. The radius of the intervals centered on the occurrence times in N_1
`changer`	Optional. Logical flag. If it is TRUE, when the defined intervals overlap, their lengths are changed to obtain not overlapped intervals. The two overlapped intervals are shortened by half of the overlapped period. In general, the resulting intervals are not centered .
`type`	Optional. Label 'Poisson', 'Normal' or 'All'. Approach to be used to calculate test p-values.
`plotRes`	Logical flag. If it is TRUE, residuals are plotted.
`...`	Further arguments to pass to the residual `plot`.

Details

The underlying idea of the tests is to analyze the behaviour of the second PP N_2, given that a point has occured in the first one, N_1. Under independence between N_1 and N_2, N_2 should be a PP with intensity lambda2.

Fo the analysis, intervals of length 2r centered on each point in N_1 are defined. To analyze the behaviour of N_2, two approaces are implemented, both based on the idea that the number of points in each interval should be a Poisson of mean μ_i equal to the integral of lambda2 in the interval.

'Poisson' option: under the null, and if the intervals are independent (that is if they do not overlap) the number of points in all them should be a Poisson of mean μ, equal to the sum of all the μ_i. The p-values is calculated as $2 * min((P(X < po) + P(X = po)/2), (P(X > po) + P(X = po)/2))$, where X is a r.v with distibution Poisson(μ) and po is the sum of the observed number of points in all the intervals. Obviously,since the p-value is calculated from a discrete distribution, it will not be uniformly distributed, and the size of the test P(reject H0 when it is true) will not be 0.05. The size depends on the value of the Poisson mean, only when it X can be approximated by a Normal, the size will be 0.05

'Normal' option: under the null, the variables $(N_i - \mu_i)/(\mu_i{**}0.5)$ must be zero mean and variance one variables but they are not identically distributed, and consequently some approximation has to be used to calculate the p-value. Since under the null, each N_i is a Poisson (μ_i), if the means mui are high enough, each variable $(N_i - \mu_i)/(\mu_i{**}0.5)$ can be approximated by a N(0,1) distribution. If μ_i are not high enough, still, the mean of the variables $(N_i - \mu_i)/(\mu_i{**}0.5)$ can be approximated by a Normal distribution using the Central limit theorem under the Lindeberg condition for r.v which are independent but not identically distributed. To apply this approximation a big sample size (number of points in N_1) is needed. Empirically, it has been seen that the smaller the values of mui, a bigger sample size is required to have a valid normal approximation. As a reference, with mui values between 0 and 1, a sample size of 50 is satisfactory

Hence, if the processes are independent, the variables $(N_i - \mu_i)/(\mu_i * *0.5)$ must have zero mean and variance one, and Nn the sample mean of $(N_i - \mu_i)/(\mu_i * *0.5)$ can be approximated by a $N(0, 1/n * *0.5)$ distribution, so that the p-value is calculated as $2 * P(Nn > abs(Nn0)) = 2 * P(Z > abs(Nn0) * n * *0.5)$.

Value

`Ni`	Number of occurrences in each interval
`mui`	Theoretical mean of the number of occurrences in each interval under the independence assumption
`Res`	Residual vector obtained from each interval
`pvP`	P-value obtained from the 'Poisson' approach
`PvN`	P-value obtained from the 'Normal' approach
`linf`	Lower limit of each interval
`lsup`	Upper limit of each interval

References

Billingsley, P. (1995). Probability and Measure. 3rd Ed. John Wiley and sons.

See Also

`TestIndNH.fun`, `DutilleulPlot.fun`

Examples

```
#Two dependent Poisson processes from  a NHCPSP
set.seed(123)
lambdao1<-runif(1000)/10
set.seed(124)
lambdao2<-runif(1000)/10
set.seed(125)
lambda12<-runif(1000)/20
aux<-DepNHCPSP.fun(lambdai1=lambdao1, lambdai2=lambdao2, lambdai12=lambda12,
fixed.seed=123)
```

```
zz<-CondTest.fun(pos1=aux$posNH1,pos2=aux$posNH2,lambda2=aux$lambda2, r=3)
zz$pvP
zz$pvN
```

`CPSPpoints.fun` *Calculates the occurrence times of the three indicator processes of a bivariate Common Poisson shock process*

Description

This function calculates the occurrence times of the points in the three indicator processes of a bivariate Common Poisson shock process (CPSP), using as input information, the two marginal processes N_1 and N_2.

Usage

```
CPSPpoints.fun(X, Y, date = NULL)
```

Arguments

`X`	Binary vector. The first CPSP marginal process; occurrence points are marked with 1 and the other with 0.
`Y`	Binary vector. The second CPSP marginal process; occurrence points are marked with 1 and the other with 0.
`date`	Optional. A vector or matrix indicating the date of each observation.

Details

A bivariate CPSP N is usually specified by its two marginal, and possibly dependent, processes N_1 and N_2, which are the observed processes. However, N can be decomposed into three independent indicator processes: $N_{(1)}$, $N_{(2)}$ and $N_{(12)}$, which are the processes of the points occurring only in the first marginal process, only in the second and in both of them (simultaneous points). The union of $N_{(1)}$ and $N_{(12)}$, and $N_{(2)}$ and $N_{(12)}$ gives respectively the two marginal processes.

Value

A list with components

`PxX`	Vector of the occurrence points in $N_{(1)}$.
`PxY`	Vector of the occurrence points in $N_{(2)}$.
`PxXY`	Vector of the occurrence points in $N_{(12)}$.
`X`	Input argument.
`Y`	Input argument.
`date`	Input argument.

References

Abaurrea, J. Asin, J. and Cebrian, A.C. (2015). A Bootstrap Test of Independence Between Three Temporal Nonhomogeneous Poisson Processes and its Application to Heat Wave Modeling. *Environmental and Ecological Statistics*, 22(1), 127-144.

See Also

CPSPPOTevents.fun

Examples

```
set.seed(123)
X<-as.numeric(runif(100)<0.10)
set.seed(124)
Y<-as.numeric(runif(100)<0.15)

aux<-CPSPpoints.fun(X=X,Y=Y)
```

CPSPPOTevents.fun — *Calculates the occurrence times of the three indicator processes of the bivariate Common Poisson shock process resulting from applying a POT approach*

Description

This function calculates the occurrence times and other characteristics (length, maximum and mean intensity) of the extreme events of the three indicator processes of a bivariate Common Poisson shock process (CPSP) obtained from a POT approach.

Usage

```
CPSPPOTevents.fun(X, Y, thresX, thresY, date = NULL)
```

Arguments

X	Numeric vector. Series (x_i) whose threshold exceedances define the first CPSP marginal process.
Y	Numeric vector. Series (y_i) whose threshold exceedances define the second CPSP marginal process.
thresX	Numeric value. Threshold used to define the extreme events in (x_i).
thresY	Numeric value. Threshold used to define the extreme events in (y_i).
date	Optional. A vector or matrix indicating the date of each observation.

Details

A CPSP N can be decomposed into three independent indicator processes: $N_{(1)}$, $N_{(2)}$ and $N_{(12)}$, the processes of the points occurring only in the first marginal process, only in the second and in both of them (simultaneous points). In the CPSP resulting from applying a POT approach, the events in $N_{(1)}$ are a run of consecutive days where (x_i) exceeds its extreme threshold but (y_i) does not exceed its extreme threshold. An extreme event in $N_{(2)}$ is defined analogously. A simultaneous event, or event in $N_{(12)}$, is a run where both series exceed their thresholds.

For the events defined in each indicator process, three magnitudes (length, maximum intensity and mean intensity) are calculated together with the initial point and the point of maximum excess of each event. In the $N_{(12)}$, the maximum and the mean intensity of both (x_i) and (y_i) are also calculated. The occurrence point of each event is located at the time of the maximum sum of the excesses over the threholds (where an excess is 0 if the observation does not exceed its corresponding threshold). According to this definition, the occurrence point in $N_{(1)}$ is the point with maximum intensity in (x_i) within the run.

The vectors `inddatX`, `inddatY` and `inddatXY`, elements of the output list, mark the observations that should be used in the estimation of each indicator process. The observations in an extreme event which are not the occurrence point are marked with 0 and treated as non observed in the estimation process. The rest are marked with 1 and must be included in the likelihood function.

Value

A list with components

`ImX`	Vector of mean excesses (over the threshold) of the extreme events in $N_{(1)}$.
`IxX`	Vector of maximum excesses (over the threshold) of the extreme events in $N_{(1)}$.
`LX`	Vector of lengths of the extreme events in $N_{(1)}$.
`PxX`	Vector of points of maximum excess of the extreme events in $N_{(1)}$.
`PiX`	Vector of initial points of the extreme events in $N_{(1)}$.
`inddatX`	Index equal to 1 in the observations which should be used in the estimation process of $N_{(1)}$, and to 0 in the others.
`ImY`	Vector of mean excesses (over the threshold) of the extreme events in $N_{(2)}$.
`IxY`	Vector of maximum excesses (over the threshold) of the extreme events in $N_{(2)}$.
`LY`	Vector of lengths of the extreme events in $N_{(2)}$.
`PxY`	Vector of points of maximum excess of the extreme events in $N_{(2)}$.
`PiY`	Vector of initial points of the extreme events in $N_{(2)}$.
`inddatY`	Index equal to 1 in the observations which should be used in the estimation process of $N_{(2)}$ and to 0 in the others.
`ImXYx`	Vector of mean excesses of the series (x_i) in $N_{(12)}$.
`IxXYx`	Vector of maximum excesses the series (x_i) in $N_{(12)}$.
`ImXYy`	Vector of mean excesses of the series (y_i) in $N_{(12)}$.
`IxXYy`	Vector of maximum excesses the series (y_i) in $N_{(12)}$.
`LXY`	Vector of lengths of the extreme events in $N_{(12)}$.
`PxXY`	Vector of points of maximum excess of the extreme events in $N_{(12)}$.

`PiXY`	Vector of initial points of the extreme events in $N_{(12)}$.
`inddatXY`	Index equal to 1 in the observations which should be used in the estimation process of $N_{(12)}$ and to 0 in the others.
`X`	Input argument.
`Y`	Input argument.
`thresX`	Input argument.
`thresY`	Input argument.
`date`	Input argument.

References

Abaurrea, J. Asin, J. and Cebrian, A.C. (2015). A Bootstrap Test of Independence Between Three Temporal Nonhomogeneous Poisson Processes and its Application to Heat Wave Modeling. *Environmental and Ecological Statistics*, 22(1), 127-144.

See Also

`CPSPpoints.fun`

Examples

```
data(BarTxTn)
dateB<-cbind(BarTxTn$ano,BarTxTn$mes,BarTxTn$diames)
BarBivEv<-CPSPPOTevents.fun(X=BarTxTn$Tx,Y=BarTxTn$Tn,thresX=318,
thresY=220, date=dateB)
```

`depchi.fun` *Estimates extremal dependence measures between two variables*

Description

This function estimates the extremal dependence coefficients χ and $\bar{\chi}$ by Coles et al. (1999). It also plots the functions $\chi(u)$ and $\bar{\chi}(u)$ against a grid of values in [0,1] to analyse the extremal dependence of two variables.

Usage

```
depchi.fun(X, Y, thresval = c(0:99)/100, tit = "", indgraph = TRUE,
bothest = TRUE, xlegend = "topleft")
```

Arguments

`X`	Numeric vector. Values of the first variable.
`Y`	Numeric vector. Values of the second variable.
`thresval`	Numeric vector. Grid values where the functions $\chi(u)$ and $\bar{\chi}(u)$ are evaluated.
`tit`	Character string. A title for the plots.
`indgraph`	Logical flag. If it is TRUE, plots are carried out in individual windows. If it is FALSE, windows with 2×1 plots are used.
`bothest`	Logical flag. If it is TRUE, two estimated coefficientes (for X given Y and for Y given X) are displayed in the same plot. Otherwise, only the coefficient for Y given X is plotted.
`xlegend`	Optional. Label "topleft" or"bottomright". Position where the legend on the graph will be located.

Details

The extremal dependence between two variables X and Y is the tendency for one variable to be large, given that the other one is large. The extremal dependence coefficients χ and $\bar{\chi}$ are defined as $\chi = \lim_{u \to 1} \chi(u)$ and $\bar{\chi}(u) = 2log(P(U > u)/logP(U > u, V > u) - 1$, where $\chi(u) = P(U > u|V > u)$ and (U,V) are the transformed uniform marginals of the variables X and Y.

χ is on the scale [0, 1], with the set (0, 1] corresponding to asymptotic dependence, and the measure $\bar{\chi}$ falls within the range [-1, 1], with the set [-1, 1) corresponding to asymptotic independence. Thus, the complete pair (χ, $\bar{\chi}$) is required as a summary of extremal dependence: ($\chi > 0$, $\bar{\chi} = 1$) signifies asymptotic dependence, in which case the value of χ determines a measure of strength of dependence within the class; alternatively, ($\chi = 0$, $\bar{\chi} < 1$) signifies asymptotic independence, in which case the value of $\bar{\chi}$ determines the strength of dependence within this class. Full details can be found in Coles et al. (1999).

In the χ plot, the expected behaviour under independence of X and Y is also plotted.

Value

A list with elements

`chiX`	Estimated χ function for Y given X evaluated at the threshold grid.
`chiY`	Estimated χ function for X given Y evaluated at the threshold grid.
`chiBX`	Estimated $\bar{\chi}$ function for Y given X evaluated at the threshold grid.
`chiBY`	Estimated $\bar{\chi}$ function for X given Y evaluated at the threshold grid.
`PX`	Estimation of the probabilities $P(U < thresval)$
`PY`	Estimation of the probabilities $P(V < thresval)$
`PXY`	Estimation of the probabilities $P[(U < thresval)\&(V < thresval)]$
`thresval`	Input argument

References

Coles, S., Heffernan, J. and Tawn, J. (1999) Dependence measures for extreme value analysis. Extremes, 2, 339-365.

See Also

TestIndNH.fun, DutilleulPlot.fun, CondTest.fun

Examples

```
data(BarTxTn)

aux<-depchi.fun(X=BarTxTn$Tx,Y=BarTxTn$Tn, thresval = c(0:99)/100,
tit = "Barcelona", xlegend = "topleft")
```

DepMarkedNHPP.fun   *Generates trajectories of dependent point processes using a marked Poison Process*

Description

This function generates d dependent (homogeneous or nonhomogeneous) point processes using a marked PP, where the marks are generated by a Markov chain process defined by a given transition matrix.

Usage

```
DepMarkedNHPP.fun(lambdaTot, MarkovM, inival = 1, fixed.seed=NULL)
```

Arguments

lambdaTot	Numeric vector. Intensity values of the underlying PP used to generate the dependent processes.
MarkovM	Matrix. Trasition probabilities of the d-state Markov chain used to generate the marks of the PP.
inival	Optional. Initial mark value used to generate the series of marks.
fixed.seed	Optional. An integer or NULL. Value used to set the seed in random generation processes; if it is NULL, a random seed is used.

Details

Points of the marked PP are generated in continuous time, using the following procedure: First, a trajectory of the underlying PP, which represents the global process of the occurrences in all the processes, is generated. Then, the mark series is generated using a d-state Markov chain. The mark series takes values in 1,2,...,d and determines in which of the d processes the point occurs

A transition matrix $P = (p_{ij})$ with equal rows leads to d independent point processes, and the more similar the rows of P, the less dependent the resulting processes. Some dependence measures between the generated processes, such as the spectral gap, are suggested in Abaurrea et al. (2014).

It is noteworthy, that the processes defined by the marks are not Poisson, since the generated marks are dependent observations, see Isham (1980).

Value

A list with elements

`posNH`	Numeric vector of the occurrences times of the underlying PP generated.
`mark`	Vector of the generated marks, which indicate the process where the point occurs.
`lambdaTot`	Input argument.
`MarkovM`	Input argument.

References

Abaurrea, J. Asin, J. and Cebrian, A.C. (2014). A Bootstrap Test of Independence Between Three Temporal Nonhomogeneous Poisson Processes and its Application to Heat Wave Modeling. *Environmental and Ecological Statistics*, 22(1), 127-144.

Isham, V. (1980). Dependent thinning of point processes. *J. Appl. Probab.*, 17(4), 987-95.

See Also

`DepNHPPqueue.fun`, `DepNHNeyScot.fun`, `DepNHCPSP.fun`, `IndNHPP.fun`, `SpecGap.fun`

Examples

```
# Generation of three dependent  point processes using a marked PP
set.seed(123)
lambdaTot<-runif(1000)/10

aux<-DepMarkedNHPP.fun(lambdaTot=lambdaTot,
MarkovM=cbind(c(0.3,0.1,0.6), c(0.1, 0.6, 0.3), c(0.6, 0.3,0.1)),fixed.seed=123)
print(cbind(aux$posNH, aux$mark))
```

`DepNHCPSP.fun` *Generates a trajectory of two dependent Poisson processes from a Common Poisson shock process*

Description

This function generates dependent (homogeneous or nonhomogeneous) Poisson processes obtained as the two marginal processes of a bivariate Common Poisson shock process.

Usage

```
DepNHCPSP.fun(lambdai1, lambdai2, lambdai12,fixed.seed=NULL)
```

Arguments

`lambdai1`	Numeric vector. Intensity values of $N_{(1)}$.
`lambdai2`	Numeric vector. Intensity values of $N_{(2)}$.
`lambdai12`	Numeric vector. Intensity values of $N_{(12)}$.
`fixed.seed`	Optional. An integer or NULL. Value used to set the seed in random generation processes; if it is NULL, a random seed is used.

Details

A bivariate CPSP N is usually specified by its two marginal, and possibly dependent, processes N_1 and N_2, which are the observed processes. However, N can be decomposed into three independent indicator processes: $N_{(1)}$, $N_{(2)}$ and $N_{(12)}$, which are the processes of the points occurring only in the first marginal process, only in the second and in both of them (simultaneous points). The union of $N_{(1)}$ and $N_{(12)}$, and $N_{(2)}$ and $N_{(12)}$ gives respectively the two marginal processes.

In a CPSP, the indicator processes are three independet PPs and the CPSP distribution is completely specified by them. The intensity vector of N_1, is `lambdai1+lambdai12` and the intensity vector of N_2 is `lambdaoi2+lambdai12`. Conditionally independent marginal processes are obtained if and only if `lambdai12`=0.

The decomposition into indicator processes can be readily applied for data generation, and it reduces to the generation of three independet PPs. Points in each process are generated in continuous time.

Value

A list with elements

`posNH1`	A numeric vector which contains the points in N_1
`posNH2`	A numeric vector which contains the points in N_2
`posNHi1`	A numeric vector which contains the points in $N_{(1)}$
`posNHi2`	A numeric vector which contains the points in $N_{(2)}$
`posNHi12`	A numeric vector which contains the points in $N_{(12)}$
`lambda1`	Numeric vector which contains the intensity vector of N_1
`lambda2`	Numeric vector which contains the intensity vector of N_2

References

Abaurrea, J. Asin, J. and Cebrian, A.C. (2015). Modeling and projecting the occurrence of bivariate extreme heat events using a nonhomogeneous common Poisson shock process. Stochastic and Environmental Research and risk assessment, 29(1), 309-322.

See Also

`DepNHNeyScot.fun`, `DepNHPPqueue.fun`, `DepMarkedNHPP.fun`

Examples

```
set.seed(123)
lambdai1<-runif(200,0,0.1)
set.seed(124)
lambdai2<-runif(200,0,0.07)
set.seed(125)
lambdai12<-runif(200,0,0.15)

aux<-DepNHCPSP.fun(lambdai1=lambdai1, lambdai2=lambdai2, lambdai12=lambdai12,
fixed.seed=123)

aux$posNH1
aux$posNH2
```

`DepNHNeyScot.fun` *Generates trajectories of dependent Neyman-Scott cluster processes*

Description

This function generates dependent (homogeneous or nonhomogeneous) point processes based on Neyman-Scott processes with the same trajectory of cluster centers.

It calls the auxiliary function `GenSons.fun` (not intended for the users), see Details section.

Usage

```
DepNHNeyScot.fun(lambdaParent, d, lambdaNumP = 1, dist = "normal",
sigmaC = 1, minC = -1, maxC = 1,fixed.seed=NULL)
```

Arguments

`lambdaParent`	Numeric vector. Intensity values of the underlying PP used to generate the centers of the clusters of the Neyman-Scott process.
`d`	Integer. Number of dependent processes to be generated.
`lambdaNumP`	Optional. Numeric vector. Mean values of the number of sons of each dependent process. If its length is equal to 1, the same value is used to generate all the dependent processes.
`dist`	Optional. Label "normal" or "uniform". Distribution used to generate the point locations of each cluster.
`sigmaC`	Optional. Numeric vector. Only used if dist="normal". Standard deviation of the normal distribution. If its lengthis equal to 1, the same value is used in the d processes.
`minC`	Optional. Numeric vector. Only used if dist="uniform". Lower limits of the Uniform distribution. If its length is equal to 1, the same value is used in the d processes.

`maxC`	Optional. Numeric vector. Only used if dist="uniform". Upper limits of the Uniform distribution. If its length is equal to 1, the same value is used in the d processes.
`fixed.seed`	Optional. An integer or NULL. Value used to set the seed in random generation processes; if it is NULL, a random seed is used.

Details

A Neyman-Scott process is a Poisson cluster process where the points in each cluster are randomly distributed around the cluster center.

Dependent homogeneous or NH point processes, in continuous time, are obtained by generating Neyman-Scott processes with the same trajectory of cluster centers. First, the Poisson process of the cluster centers is generated. Then, the number of points in each cluster is generated using a Poisson distribution with means which can be different in each process. The point locations around each center can be generated using two distributions N(0, sigmaC) or Uniform(minC, maxC).

Remark that high values of sigmaC or the range maxC-minC lead to a high variability around the center and to a low dependence between the processes.

Value

A list with elements

`posNH`	A list of d vectors, containing the occurrence points of the d dependent processes. The name of the elements of the list are PP1, PP2...PPd

References

Abaurrea, J. Asin, J. and Cebrian, A.C. (2014). A Bootstrap Test of Independence Between Three Temporal Nonhomogeneous Poisson Processes and its Application to Heat Wave Modeling. *Environmental and Ecological Statistics*.

See Also

`IndNHNeyScot.fun`, `DepNHPPqueue.fun`, `DepMarkedNHPP.fun`, `DepNHCPSP.fun`

Examples

```
# Generation of three dependent Neyman-Scott processes with  the same mean number
#of sons per cluster and locations generated by the same normal distribution

set.seed(123)
lambdaParent<-runif(100,0,0.1)

DepNHNeyScot.fun(lambdaParent=lambdaParent, d=3, lambdaNumP = 2,
 dist = "normal", sigmaC = 3,fixed.seed=123)
```

`DepNHPPqueue.fun` *Generates trajectories of dependent Poisson processes based on queue simulation*

Description

This function generates d dependent (homogeneous or nonhomogeneous) Poisson processes. using $d-1$ queues in tandem.

Usage

```
DepNHPPqueue.fun(lambda, d, rate = 1,type="NH", T=NULL, nEv ,fixed.seed=NULL)
```

Arguments

`lambda`	Numeric value or vector. Intensity of the first PP. If its length is 1, homogeneous PPs are generated.
`d`	Integer. Number of dependent processes to be generated.
`rate`	Optional. Numeric value or vector. Parameters (inverse of the mean) of the exponential distributions used to generate the time services, see `rexp`. If its length is 1, the same parameter is used to generate the time services in the $d-1$ steps.
`type`	Optional. Labels "H" or "NH". Type of the processes,homogeneous or nonhomogeneous. If type="H", the length of lambda must be 1.
`T`	Optional. Positive integer. Length of the period where the point are going to be generated. Only used if type="H" .
`nEv`	Optional. Positive integer. Number of points to be generated in the PPs. Only used if type="H".
`fixed.seed`	Optional. An integer or NULL. Value used to set the seed in random generation processes; if it is NULL, a random seed is used.

Details

The generation of dependent homogeneous PPs is based on Burke s theorem, Burke (1956), which states that in a stationary system formed by a tandem of M M 1 queues with input intensity λ and exponential time services, the input and the output processes of the first queue and the output of the second are three dependent PPs with the same intensity λ.

Points are generated in continuous time, using the following procedure: first the points of a PP and their time services are generated, and their corresponding output times obtained. The output times are the occurrence times of the second process. Using them as input times of the following service and generating new time services, the second output times, which are the occurrence times of the third PP, are obtained. The last step is repeated up to obtain d dependent PPs.

Analogously, the generation of dependent NHPPs is based on a result by Keilson and Servi (1994), which states that the output process from a M(t) G 1 queue is also a PP with a NH intensity equal to the convolution $\lambda_{out}(t) = \lambda_{inp}(t) * f(t)$, where $\lambda_{inp}(t)$ is the input intensity and $f(t)$ the density function of the service time, in this case an exponential distribution.

In the homogeneous processes, the argument λ can be an integer or a vector with equal values. In the first case, the argument `nEv` must be specified; in the second, the length of the vector determines the length of period where the points are generated (as in the nonhomogeneous case).

Value

A list with elements

`posNHs`	A list of d vectors, containing the occurrence points in each PP. The name of the elements of the list are PP1, PP2,..., PPd.
`lambdaM`	A d-column matrix containing the intensity vectors of the d dependent processes.

References

Abaurrea, J. Asin, J. and Cebrian, A.C. (2014). A Bootstrap Test of Independence Between Three Temporal Nonhomogeneous Poisson Processes and its Application to Heat Wave Modeling. *Environmental and Ecological Statistics*, 22(1), 127-144.

Burke, P. J. (1956). The Output of a Queuing System. *Operations Research*. 4(6), 699-704.

Keilson, J. Servi, L.D. (1994). Networks of nonhomogeneous M G ∞. *J. Appl. Probab.*, 31, 157-68.

See Also

`IndNHPP.fun`, `DepNHNeyScot.fun`, `DepMarkedNHPP.fun`, `DepNHCPSP.fun`

Examples

```
#Generation of 3 dependent HPPs, with  mean service time equal to 10
aux<-DepNHPPqueue.fun(lambda=0.05, d=3, rate=0.1, type="H", nEv=25,
fixed.seed=123)
aux$posNHs

#Generation of 3 dependent NHPPs, with  mean service time equal to 10
#lambda<-runif(200,0,0.1)

#aux<-DepNHPPqueue.fun(lambda=lambda, d=3, rate=0.1, type="NH")
#aux$posNHs
```

`DistObs.fun`	*Calculates the set of close points and the mean distance for each point in the first process of a set or two or three processes*

Description

Given a set or two or three processes, this function calculates the set of close points and the mean distance for each point in the first process.

It calls the functions `calcdist.fun`, not intended for the users, and `uniongentri.fun`.

Usage

```
DistObs.fun(posx, posy, posz=NULL,  info = FALSE,
PA = FALSE, procName=c('X','Y','Z'),...)
```

Arguments

`posx`	Numeric vector. Position of the occurrence points in the first process.
`posy`	Numeric vector. Position of the occurrence points in the second process.
`posz`	Optional. Numeric vector. Position of the occurrence points in the third process.
`info`	Optional logical flag. If it is TRUE, information about the generated points is showed on the screen and dotcharts and bivariate charts of the occurrence points of the processes are displayed.
`PA`	Optional logical flag. If it is TRUE, the close point relation is broadened by including the previous and the following points to the overlapping intervals.
`procName`	Vector of character strings. Labels for the first, second and third processes.
`...`	Further arguments to pass to `plot` if argument *info="TRUE"*

Details

Given a set of two or three point proccesses, for each point t_{x_i} in the first process of the set, this function calcultes its set of close points and the mean distance to its close points. The definition of set of close points can be found in Abaurrea et al. (2015)) and the distances are defined as $|t_{y_j} - t_{x_i}|$ if there are two processes and as $|t_{y_j} - t_{x_i}| + |t_{z_k} - t_{x_i}|$ if there are three.

Value

`DistTri`	The vector of the means of the distances of points $t_{x_i}, t_{y_i}, t_{z_i}$ for each x-coordinate

References

Abaurrea, J. Asin, J. and Cebrian, A.C. (2014). A Bootstrap Test of Independence Between Three Temporal Nonhomogeneous Poisson Processes and its Application to Heat Wave Modeling. *Environmental and Ecological Statistics*, 22(1), 127-144.

See Also

TestIndNH.fun, DistSim.fun, uniongentri.fun

Examples

```
data(BarTxTn)
dateB<-cbind(BarTxTn$ano,BarTxTn$mes,BarTxTn$diames)
BarBivEv<-CPSPPOTevents.fun(X=BarTxTn$Tx,Y=BarTxTn$Tn,thresX=318,
thresY=220, date=dateB)
DistObs.fun(BarBivEv$PxX, BarBivEv$PxY, BarBivEv$PxXY,info = TRUE)
```

DistShift.fun *Shifts, conditionally on the first process, the remaining processes in a set of up to three processes and calculates the set of close points and the mean distance for each point in the first process*

Description

Given a set of up to three processes, this function fixes the first one and shiftes the other ones a given distance. Then, it calculates the set of close points and the mean distance for each point t_{x_i} in the first process. It can be used with homogeneous and non homogeneous processes.

Usage

```
DistShift.fun(posx,posy,posz=NULL, T,  shii1, shii2=NULL, PA = FALSE,info=FALSE,...)
```

Arguments

posx	Numeric vector. Position of the occurrence points in the first process.
posy	Numeric vector. Position of the occurrence points in the second process.
posz	Optional. Numeric vector. Position of the occurrence points in the third process.
T	Numeric value. Length of the observation period of the processes.
shii1	Numeric value. Distance used to shift the points in the second process. It must be a positive value lower than the length of each lambda vector.
shii2	Optional. Numeric value. Distance used to shift the points in the third process. If there is only two processes it must be NULL. It must be a positive value lower than the length of each lambda vector.
PA	Optional. Logical flag. If it is TRUE, the close point relation is broadened by including the previous and the following points to the overlapping intervals.
info	Optional. Logical flag. If it is TRUE, information about the generated points is shown on the screen and dotcharts and bivariate charts of the occurrence points of the three processes are displayed.
...	Further arguments to pass to plot and to dotchart if the argument info=T

Details

Given a set of three (or two) point processes, this function aims to transform the set of processes by a translation, conditionally on the first process. The idea of this translation is to keep the distribution of the processes but break any dependence between them, without the need of parametric models to describe the univariate marginal patterns. The only information required is the marginal intensities of the processes.

The key idea, see Lotwick and Silverman (1982), is to wrap the processes onto a torus by identifying opposite sides, The first process is fixed, while the others are translated over the torus. In nonhomogenous processes, translation may change their distribution, and to compensate, the intensity must also be translated.

Then, the function calculates the set of close points and the mean distance for each point t_{x_i} in the first process, in the new shifted set of processes.

This function is mainly used to develop a Lotwick-Silverman type test to check the independecne between the processes, see `TestIndLS.fun`.

For homogenous processes, the intensity vectors in `lambda` must be constant (that is all the values in a column must be equal).

Value

`DistTri`	Vector of the mean distance to the points in the set of close points, for each t_{x_i} in the first process.

References

Cebrian, A.C. Abaurrea, J. Asin, J. (2014). NHPoisson: An R Package for Fitting and Validating Nonhomogeneous Poisson Processes. Journal of Statistical Software.

Lotwick, H.W. and Silverman, B.W. (1982). Methods for analysing Spatial processes of several types of points. *J.R. Statist. Soc. B*, 44(3), pp. 406-13

See Also

`TestIndLS.fun`, `DistSim.fun`

Examples

```
set.seed(123)
lambdax<-runif(200, 0.01,0.17)
set.seed(124)
lambday<-runif(200, 0.015,0.15)
set.seed(125)
lambdaz<-runif(200, 0.005,0.1)
posx<-simNHPc.fun(lambda=lambdax, fixed.seed=123)$posNH
posy<-simNHPc.fun(lambda=lambday, fixed.seed=123)$posNH
posz<-simNHPc.fun(lambda=lambdaz, fixed.seed=123)$posNH

DistShift.fun(posx=posx,posy=posy,posz=posz,T=200,
shii1=59, shii2=125 )
```

`DistSim.fun`	*Generates a set of up to three independent processes and calculates the set of close points and the mean distance for each point in the first process*

Description

Given a point process, this function generates a set of one or two more processes and calculates the set of close points and the mean distance for each point t_{x_i} in the first process. Two types of processes, Poisson processes (PP) and Neyman-Scott cluster processes (NSP), are implemented up to now. Homogeneous and nonhomogeneous processes can be generated in both cases.

If a seed must be fixed in the generation process, function `DistSimfix.fun` has to be used.

Usage

```
DistSim.fun(posx, NumProcess=2, type = "Poisson", lambdaMarg = NULL,
lambdaParent = NULL, lambdaNumP=NULL, dist = "normal", sigmaC = 1,
minC = -1, maxC = 1, PA = FALSE,info=FALSE,...)

DistSimfix.fun(posx, NumProcess=2, type = "Poisson", lambdaMarg = NULL,
lambdaParent = NULL,lambdaNumP=NULL, dist = "normal", sigmaC = 1,
minC = -1, maxC = 1, PA = FALSE,info=FALSE,fixed.seed=1,...)
```

Arguments

`posx`	Numeric vector. Position of the occurrence points in the first process.
`NumProcess`	Optional. Integer equal to 2 or 3, the number of processes involved.
`type`	Optional. Label "Poisson" or "PoissonCluster". Type of point processes to be generated. Up to now, only two types are available: Poisson processes ("Poisson") and Neyman-Scott cluster processes ("PoissonCluster").
`lambdaMarg`	Two-column matrix. Only used when *type="Poisson"*. Each column is the intensity $\lambda(t)$ used to generate the PPs.
`lambdaParent`	Numeric vector. Only used when *type="PoissonCluster"*. Intensity values of the underlying PP used to generate the centers of the clusters of the NSP.
`lambdaNumP`	Numeric vector (length $\leq$ 2). Only used when *type="PoissonCluster"*. Mean values of the number of sons of each process. If its length is 1 and `NumProcess=2`, the same value is used for both processes.
`dist`	Optional. Label "normal" or "uniform". Only used when *type="PoissonCluster"*. Distribution used to generate the point locations in each cluster.
`sigmaC`	Optional. Numeric vector. Only used when *type="PoissonCluster"* and *dist="normal"*. Standard deviation of the normal distribution. If its length is 1 and `NumProcess=2`, the same value is used for both processes.
`minC`	Optional. Numeric vector. Only used when *type="PoissonCluster"* and *dist="uniform"*. Lower limits of the Uniform distribution. If its length is 1 and `NumProcess=2`, the same value is used for both processes.

`maxC`	Optional. Numeric vector. Only used when *type="PoissonCluster"* and *dist="uniform"*. Upper limits of the Uniform distribution. If its length is 1 and `NumProcess=2`, the same value is used for both processes.
`PA`	Optional. Logical flag. If it is TRUE, the close point relation is broadened by including the previous and the following points to the overlapping intervals.
`info`	Optional. Logical flag. If it is TRUE, information about the generated points is shown on the screen and dotcharts and bivariate charts of the occurrence points of the three processes are displayed.
`fixed.seed`	Optional. Only available in `DistSimfix.fun`. Integer value used to set the seed in random generation procedures.
`...`	Further arguments to pass to `plot` and to `dotchart` if the argument `info=T`

Details

Up to now, generation of two types of processes (Poisson, "Poisson", and Neyman-Scott cluster processes,"PoissonCluster") is available. Generation of NHPPs is done using the inversion approach, see Cebrian et al. (2014) and `simNHPc.fun`. For generation of NSPs, see `IndNHNeyScot.fun`

The only difference between `DistSim.fun` and `DistSimfix.fun` is that the first one uses a random seed while in the second one a seed is set by the argument `fixed.seed`.

This function is mainly used in `TestIndNH.fun`.

The lenght of the period where the processes are generated is determined by the length of the argument `lambdaParent` or the number of rows of `lambdaMarg`.Homogenous processes are generated if the intensity vectors in `lambdaParent` or in `lambdaMarg` are constant (that is if all the values in the vector are equal).

Value

`DistTri`	Vector of the mean distance to the points in the set of close points, for each t_{x_i} in the first process.

References

Abaurrea, J. Asin, J. and Cebrian, A.C. (2014). A Bootstrap Test of Independence Between Three Temporal Nonhomogeneous Poisson Processes and its Application to Heat Wave Modeling. Environmental and Ecological Statistics.

Cebrian, A.C. Abaurrea, J. Asin, J. (2014). NHPoisson: An R Package for Fitting and Validating Nonhomogeneous Poisson Processes. Journal of Statistical Software. To appear.

See Also

`TestIndNH.fun`, `DistObs.fun`, `IndNHNeyScot.fun`, `simNHPc.fun`

Examples

```
#Observed process: PP; simulated processes: two independent PPs
```

```
set.seed(123)
lambdax<-runif(200, 0.01,0.15)
set.seed(124)
lambday<-runif(200, 0.005,0.1)
set.seed(125)
lambdaz<-runif(200, 0.005,0.2)
posaux<-simNHPc.fun(lambda=lambdax, fixed.seed=123)$posNH

DistSimfix.fun(posx=posaux, type = "Poisson",
lambdaMarg = cbind(lambday,lambdaz), fixed.seed=123)
#DistSim.fun(posx=posaux, type = "Poisson",
# lambdaMarg = cbind(lambday,lambdaz))
```

DutilleulPlot.fun — *Performs a Diggle's randomization testing procedure to check independence between two point processes*

Description

This function applies the Diggle's randomization testing procedure extended by Dutilleul(2011) and performs a plot which checks graphically the independence of two point proceses. It is implemented for homogenous and non homogenous processes.

Usage

```
DutilleulPlot.fun(posx, posy, lambday, nsim = 1000, lenve = c(0.025, 0.975), ...)
```

Arguments

`posx`	Numeric vector. Position of the occurrence points of the first point process.
`posy`	Numeric vector. Position of the occurrence points of the second point process.
`lambday`	Numeric vector. Intensity values of the second point process
`nsim`	Optional. Positive integer. Number of simulations performed to obtain the confidence bands.
`lenve`	Optional. Numeric vector. Lower and the upper percentiles which determine the limits of the confidence band.
`...`	Further arguments to be passed to `plot`.

Details

This is a procedure to check graphically the independence of two point proceses. It is based on the comparison of the cumulative relative frequency of the nearest neighbour distances between the points in the two observed processes, with their counterpart in two independent processes with the same intensities. The procedure consists on plotting the cumulative relative frequency of the observed processes and a confidence band calculated from *nsim* simulated independent processes.

Value

A list with the elements used in the plot

`quantobs`	Vector of observed percentiles of the nearest neighbour distances.
`enve1`	Vector of lower limits of the confidence band.
`enve2`	Vector of upper limits of the confidence band.

References

Dutilleul, P. (2011), *Spatio-temporal heterogeneity: Concepts and analyses*, Cambridge University Press.

See Also

`TestIndNH.fun`, `CondTest.fun`,`nearestdist.fun`

Examples

```
#Two independent NHPPs
set.seed(123)
lambdax<-runif(200, 0.01,0.1)
set.seed(124)
lambday<-runif(200, 0.015,0.15)
posx<-simNHPc.fun(lambdax,fixed.seed=123)$posNH
posy<-simNHPc.fun(lambday, fixed.seed=123)$posNH

aux<-DutilleulPlot.fun(posx, posy, lambday,  nsim = 100)


#Two dependent NSPs
#set.seed(123)
#lambdaParent<-runif(200)/10
#DepPro<-DepNHNeyScot.fun(lambdaParent=lambdaParent, d=2, lambdaNumP = 3,
# dist = "normal", sigmaC = 3,fixed.seed=123)
#posx<-DepPro$PP1
#posy<-DepPro$PP2
#aux<-DutilleulPlot.fun(posx, posy, lambday, nsim = 100)
```

`IndNHNeyScot.fun` *Generates trajectories of independent Neyman-Scott cluster processes*

Description

This function generates independent (homogeneous or nonhomogeneous) Neyman-Scott cluster processes with independent trajectories of cluster centers with the same intensity.

It calls the auxiliary function `GenSons.fun` (not intended for the users), see Details section.

Usage

```
IndNHNeyScot.fun(lambdaParent, d, lambdaNumP = 1,  dist = "normal",
sigmaC = 1, minC = -1, maxC = 1,fixed.seed=NULL)
```

Arguments

`lambdaParent`	Numeric vector. Intensity values of the underlying PP used to generate the centers of the clusters of the Neyman-Scott process.
`d`	Integer. Number of independent processes to be generated.
`lambdaNumP`	Optional. Numeric vector. Mean values of the number of sons of each dependent process. If its length is equal to 1, the same value is used to generate all the dependent processes.
`dist`	Optional. Label "normal" or "uniform". Distribution used to generate the point locations of each cluster.
`sigmaC`	Optional. Numeric vector. Standard deviation of the normal distribution. Only used if dist="normal". If its length is equal to 1, the same value is used in the d processes.
`minC`	Optional. Numeric vector. Lower limits of the Uniform distribution. Only used if dist="uniform". If its length is equal to 1, the same value is used in the d processes.
`maxC`	Optional. Numeric vector. Upper limits of the Uniform distribution. Only used if dist="uniform". If its length is equal to 1, the same value is used in the d processes.
`fixed.seed`	Optional. An integer or NULL. Value used to set the seed in random generation processes; if it is NULL, a random seed is used.

Details

A Neyman-Scott process is a Poisson cluster process where the points in each cluster are randomly distributed around the cluster center. For generating each process, an independent trajectory of the Poisson process of the cluster centers is generated first. Then, the number of points in each cluster is generated using a Poisson distribution with mean value μ_{P_i} (i=1,...d). Finally, th e point locations around each center can be generated using two distributions N(0, sigmaC) or Uniform(minC, maxC).

The lenght of the period where the processes are generated is determined by the length of the argument `lambdaParent`.

Homogenous processes are generated if the intensity vector `lambdaParent` is constant (that is if all the values are equal).

Value

A list with elements

`posNH`	A list of d vectors, each one containing the time occurrences of one of the dependent processes. The name of the elements of the list are PP1, PP2...PPd

References

Abaurrea, J. Asin, J. and Cebrian, A.C. (2014). A Bootstrap Test of Independence Between Three Temporal Nonhomogeneous Poisson Processes and its Application to Heat Wave Modeling. *Environmental and Ecological Statistics*.

See Also

`DepNHNeyScot.fun`, `IndNHPP.fun`

Examples

```
set.seed(123)
lambda<-runif(1000)/10

IndNHNeyScot.fun(lambdaParent=lambda, d=3, lambdaNumP = c(2,3,2),  dist = "normal",
sigmaC = 2, fixed.seed=123)
```

`IndNHPP.fun` *Generates trajectories of independent Poisson processes*

Description

This function generates independent Poisson processes (PPs), which can be homogeneous or non-homogeneous depending on the value of the intensity vectors.

Usage

```
IndNHPP.fun(lambdas, fixed.seed=NULL)
```

Arguments

`lambdas`	Matrix where each column contains the intensity vector to generate a Poisson process
`fixed.seed`	An integer or NULL. If it is an integer, that is the value used to set the seed in random generation processes. It it is NULL, a random seed is used.

Details

The number of generated processes is determined by the number of columns of the argument `lambdas`. The lenght of the period where the processes are generated is determined by the number of rows of `lambdas`.

Homogenous processes are generated if the corresponding intensity vector is constant (that is if all the rows of the corresponding column are equal).

For the generation algorithm of each PP, see `simNHPc.fun`.

Value

posNHsA list of d vectors, each one containing the time occurrences of the independent NHPP. The name of the elements of the list are PP1, PP2...PPd

See Also

`IndNHNeyScot.fun`, `simNHPc.fun`

Examples

```
set.seed(123)
lambdas<-cbind(runif(500)/10, rep(0.05,500))

IndNHPP.fun(lambdas=lambdas, fixed.seed=123)
```

`nearestdist.fun` *Calculates the distance to the nearest point in the second process for each point in the first process*

Description

Given the occurrence points in two point processes, this function calculates for each point in the first process, the distance to the nearest occurrence point in the second process.

Usage

```
nearestdist.fun(posx, posy)
```

Arguments

`posx`	Numeric vector. Position of the occurrence points in the first point process.
`posy`	Numeric vector. Position of the occurrence points in the second point process.

Details

The distance between two points x_i and y_i in time, is the absolute value of their difference: $|x_i - y_i|$.

To obtain the vector of nearest points, this function applies to each point in `posx`, the function `pdist.fun`, which calculates the distance to its nearest point in `posy`.

Value

Vector of distances to the nearest point in the second process for each point in the first process.

See Also

`DutilleulPlot.fun`

Examples

```
posx<-c(3,8,23,54,57,82)
posy<-c(2,8,14,16,29,32,45,55,65)
nearestdist.fun(posx, posy)
```

`NHD.fun` *It estimates the cross D-function for two sets of point processes*

Description

This function estimates the cross nearest neighbour distance distribution function, D, between two sets, C and D, of (homogenous or nonhomogeneous) point processes. The D-function is evaluated in a grid of values r, and it can be optionally plotted.

It calls the auxiliary functions NHDaux.fun and other functions, not intended for users.

Usage

```
NHD.fun(lambdaC, lambdaD, T=NULL,Ptype='inhom', posC, typeC=1, posD, typeD=1,
r = NULL, dplot = TRUE, tit = "D(r)")
```

Arguments

`lambdaC`	A matrix. Each column is the intensity vector of one of the point processes in C. If there is only one process in C, it can be a vector or even a numeric value if the process is homogeneous.
`lambdaD`	A matrix. Each column is the intensity vector of one of the point process in D. If there is only one process in D, it can be a vector oe even a numeric value if the process is homogeneous.
`T`	Numeric value. Length of the observation period. It only must be specified if all the processes are homogeneous the number of rows in `lambdaC` and `lambdaD` is 1.
`Ptype`	Optional. Label: 'hom' or 'inhom'. The first one indicates that all the point processes in sets C and D are homogeneous. In that case, columns of arguments `lambdaC` and `lambdaD` can be a number or a constant vector.
`posC`	Numeric vector. Time position of the points in all the point processes in C.
`typeC`	Numeric vector with the same length as `posC`. Code of the point process in C where points in `posC` have occurred. See Details.
`posD`	Numeric vector. Time position of the points in all the point processes in D.
`typeD`	Numeric vector with the same length as `posD`. Code of the point process in D where points in `posD` have occurred.
`r`	Numeric vector. Values where D-function must be evaluated.
`dplot`	Optional. A logical flag. If it is TRUE, a plot of the D-function is shown.
`tit`	Optional. The title to be used in the plot of the D-function.

Details

This function estimates the cross nearest neighbour distance disribution function D, between two sets C and D of point processes, using the estimator suggested by Cronie and van Lieshout (2015), adapted to time point processes.

The D-function is the distribution function of the distances from a point in a process in C to the nearest point in a process D. In homogeneous proceesses, it estimates the probability that at least one point in a process in set D occurs at a distance lower than r of a given point in a process in set C. If the processes are nonhomogenous, the inhomogenous version of the function, adjusted for time varying intensities, is used. It is calculated using the Hanisch estimator, see Van Lieshout (2006)

Small values of cross D-function suggest few points in processes in D in the r-neighbourhood of points of processes in C. Large values indicate that points in processes in D are attracted by those of processes in C.

For inference about independence of the processes, K and J-functions should be used.

The occurrence points in all the processes in C must be part of the input. Since the number of points will be possibly different in each process, a matrix cannot be used. Instead two vectors with the same length are used: the first one contains the occrrence points in all the processes while the second one indicates the point process in C where the point in the same row in `posC` has occurred. The codes used in `typeC` are the column number where the intensity of that process is in matrix `lambdaC`. The same for set D.

See `NHJ.fun` for details on default values of `r` and `L`.

Value

A list with elements

`r`	Vector of values where D-function is evaluated.
`NHDr`	Estimations of D-function at values r.
`T`	Length of the observation period.

References

Cronie, O. and van Lieshout, M.N.M. (2015). Summary statistics for inhomogeneous marked point processes. Ann Inst Stat Math. DOI 10.1007/s10463-015-0515-z

Stoyan et al (2001). On the estimation of distance distribution functions for points processes and random sets. Image Anal Stereol, 20, 65-69

Van Lieshout, M.N.M. (2006) A J-function for marked point patterns. AISM, 58, 235-259. DOI 10.1007/s10463-005-0015-7

See Also

`NHK.fun`, `NHJ.fun`, `NHF.fun`

Examples

```
#Sets  C and D with one  independent NHPP
set.seed(123)
lambda1<-runif(500, 0.05, 0.1)
set.seed(124)
lambda2<-runif(500, 0.01, 0.2)
pos1<-simNHPc.fun(lambda=lambda1, fixed.seed=123)$posNH
pos2<-simNHPc.fun(lambda=lambda2, fixed.seed=123)$posNH

aux<-NHD.fun(lambdaC=lambda1, lambdaD=lambda2, posC=pos1, typeC=1, posD=pos2, typeD=1)
aux$NHDr


#Sets  C and D with two independent NHPPs
#pos3<-simNHPc.fun(lambda=lambda1, fixed.seed=321)$posNH
#pos4<-simNHPc.fun(lambda=lambda2, fixed.seed=321)$posNH

#NHD.fun(lambdaC=cbind(lambda1,lambda2), lambdaD=cbind(lambda1,lambda2), posC=c(pos1,pos2),
# typeC=c(rep(1, length(pos1)), rep(2, length(pos2))), posD=c(pos3, pos4),
# typeD=c(rep(1, length(pos3)), rep(2, length(pos4))))
```

`NHF.fun` *It estimates the cross F-function for two sets of point processes*

Description

This function estimates the cross F-function, between two sets, C and D, of (homogenous or non-homogeneous) point processes. The F-function is evaluated in a grid of values r, and it can be optionally plotted.

It calls the auxiliary functions NHFaux.fun and other functions not intended for users.

Usage

```
NHF.fun(lambdaD, T=NULL, Ptype='inhom', posD, typeD=1, r=NULL,L=NULL, dplot=TRUE,
tit='F(r)')
```

Arguments

`lambdaD`	A matrix. Each column is the intensity vector of one of the point process in D. If there is only one process in D, it can be a vector oe even a numeric value if the process is homogeneous.
`T`	Numeric value. Length of the observation period. It only must be specified if all the processes are homogeneous the number of rows in `lambdaC` and `lambdaD` is 1.

`Ptype`	Optional. Label: 'hom' or 'inhom'. The first one indicates that all the point processes in sets C and D are homogeneous. In that case, columns of arguments `lambdaC` and `lambdaD` can be a number or a constant vector.
`posD`	Numeric vector. Time position of the points in all the point processes in D.
`typeD`	Numeric vector with the same length as `posD`. Code of the point process in D where the point in the same row in `posD` has occurred. The code must be the column number where the intensity of that process is in matrix `lambdaD`.
`r`	Numeric vector. Values where F-function must be evaluated.
`L`	Optional. Numeric vector. Net of values during the observation period used to calculate the F-function. If the value is NULL, a default vector, formed by all the integers in the observation period, is used
`dplot`	Optional. Logical flag. If it is true, the plot of the F-function is performed.
`tit`	Optional. The title to be used in the plot of the F-function.

Details

This function estimates the cross F-function for the processes in set D, using the estimator suggested by Cronie and van Lieshout (2015), adapted to time point processes.

The cross F-function, also known as empty space function, is the distribution function of the distances from an arbitray point in the space to the nearest point in a process in D. In homogeneous processes, it estimates the probability that at least one point in processes in D occurs at a distance lower than r of an arbitray point in the space. If the processes are nonhomogenous, the inhomogenous version of the function, adjusted for time varying intensities, is used.

See `NHJ.fun` for details on default values of `r` and `L`.

The occurrence points in all the processes in D must be part of the input. Since the number of points will be possibly different in each process, a matrix cannot be used. Instead two vectors with the same length are used: the first one contains the occrrence points in all the processes while the second one indicates the point process in D where the point in the same row in `posD` has occurred. The codes used in `typeD` are the column number where the intensity of that process is in matrix `lambdaD`.

Value

A list with elements

`r`	Values r where the F-function is evaluated.
`NHFr`	Estimated values of the F-function evaluated at values r.
`T`	Length of the observation period of the process.
`L`	Net of values in the observation period used to calculate the F-function.

References

Cronie, O. and van Lieshout, M.N.M. (2015). Summary statistics for inhomogeneous marked point processes. Ann Inst Stat Math. DOI 10.1007/s10463-015-0515-z

Stoyan et al (2001). On the estimation of distance distribution functions for points processes and random sets. Image Anal Stereol, 20, 65-69

Van Lieshout, M.N.M. (2006) A J-function for marked point patterns. AISM, 58, 235-259. DOI 10.1007/s10463-005-0015-7

See Also

NHK.fun, NHJ.fun, NHD.fun

Examples

```
set.seed(123)
lambda1<-runif(500, 0.05, 0.1)
pos1<-simNHPc.fun(lambda=lambda1, fixed.seed=123)$posNH

aux<-NHF.fun(lambdaD=lambda1, posD=pos1, typeD=1)
aux$NHFr

#Set D with two processes
#lambda2<-runif(1000, 0.01, 0.2)
#pos2<-simNHPc.fun(lambda=lambda2, fixed.seed=123)$posNH
#NHF.fun(lambdaD=cbind(lambda1,lambda2), posD=c(pos1,pos2),
# typeD=c(rep(1, length(pos1)), rep(2, length(pos2))) )
```

`NHJ.fun` *It estimates the cross J-function for two sets of point processes*

Description

This function estimates the cross J-function between two sets, C and D, of (homogenous or non-homogeneous) point processes. The J-function is evaluated in a grid of values r, and it can be optionally plotted. An independence test based on J-function is also available.

It calls the auxiliary functions NHJaux.fun and Jenv, not intended for users.

Usage

```
NHJ.fun(lambdaC, lambdaD, T=NULL,Ptype='inhom', posC, typeC=1, posD,  typeD=1, r = NULL,
L = NULL,  test=FALSE, nTrans=100, rTest=20, conf=0.95, dplot = NULL,
tit = rep("J-function", 3), mfrow = c(1, 1), cores=1, fixed.seed=NULL)
```

Arguments

`lambdaC`	A matrix. Each column is the intensity vector of one of the point processes in C. If there is only one process in C, it can be a vector or even a numeric value if the process is homogeneous.
`lambdaD`	A matrix. Each column is the intensity vector of one of the point process in D. If there is only one process in D, it can be a vector oe even a numeric value if the process is homogeneous.

`T`	Numeric value. Length of the observation period. It only must be specified if all the processes are homogeneous, that is if the number of rows in `lambdaC` and `lambdaD` is 1.
`Ptype`	Optional. Label: 'hom' or 'inhom'. The first one indicates that all the point processes in sets C and D are homogeneous. In that case, columns of arguments `lambdaC` and `lambdaD` can be a number or a constant vector.
`posC`	Numeric vector. Time position of the points in all the point processes in C.
`typeC`	Numeric vector with the same length as `posC`. Code of the point process in C where the points in `posC` have occurred. See Details
`posD`	Numeric vector. Time position of the points in all the point processes in D.
`typeD`	Numeric vector with the same length as `posD`. Code of the point process in D where the points in `posD` have occurred.
`r`	Optional. Numeric vector. Values where J-function must be evaluated. If it is NULL, a default vector is used, see Details
`L`	Optional. Numeric vector. Net of values in the observation period used to calculate the J-function. If it is NULL, a default vector is used, see Details.
`test`	Optional. Logical flag. If it is TRUE, a test to check the independence and a 95% envelope for the J-function are calculated.
`nTrans`	Optional. Numeric value. Only used if `test=TRUE`. Number of translations to be performed in the test and envelope calculation.
`rTest`	Optional. Numeric value. Maximum value of r used to calculate the independence test statistc, see Details.
`conf`	Optional. Numeric value in (0,1). Confidence level of the envelope for the K-function.
`dplot`	Optional. Label 'JDF' or 'J'. If it is equal to'JDF', plots of J, D and F-functions are displayed. If it is 'J', only J-function is plotted.
`tit`	Optional. A vector with one or three titles to be used in the plots of J, D and F-functions.
`mfrow`	Optional. Argument to be passed to `par` for the plot of the J-function.
`cores`	Optional. Number of cores of the computer to be used in the calculations.
`fixed.seed`	An integer or NULL. If it is an integer, that is the value used to set the seed in random generation processes. It it is NULL, a random seed is used.

Details

This function estimates the cross J-function between two sets, C and D, of homogenoeus or nonhomogeneous time point processes, using a version of the spatial estimator suggested by Cronie and van Lieshout (2015) adapted to time processes. In the case of nonhomogeneous processes, the estimator is only defined for second order intensity-reweighted stationary point processes. J-function measures the interpoint dependence between points in any of the processes in D, and points in any of the processes in C, adjusted for time varying intensity in the case of nonhomogenous processes.

The cross J-function is defined as $J_{CD}(r) = (1 - D_{CD}(r))/(1 - F_D(r))$, if $F_D(r) < 1$ and it is not calculated otherwise. It compares D(r), the distribution function of the distances from a point in any of the processes in set C to the nearest point in any of the processes in set D, to F(r), the distribution function of the distances from a fixed point in the space to the nearest point in any of the processes in set D.

If the processes in C are independent of the processes in D (conditionally on the marginal structure of the processes) J-funtion is equal to 1, since D(r)=F(r). Hence, deviations of J(r) estimations from 1, suggest dependence betweent the two sets of processes.

If argument `r` is NULL, it is calculated as

r1<-max(20, floor(T/20))

r<-seq(1,r1,by=2)

if (length(r)>200) r<-seq(1,r1,length.out=200)

If argument `L` is NULL, it is calculated as

L<-seq(1, T, by=2)

if (length(L)>200) L<-seq(1,T,by = round((T - 1)/199))

Testing independence

In order to test the independence hypothesis using this function, a Lotwick-Silverman type test is available. This test provides a nonparametric way to test independence when the study area is rectangular. The key idea is to wrap the processes onto a torus by identifying opposite sides, keeping the processes in set C fixed and translating the processes in set D randomly over the torus. In nonhomogenous processes, random translation may change their distribution, and to compensate, the intensity must also be translated. The considered translations keep the distribution of the processes in D but break any dependence between them, without the need of parametric models to describe the univariate marginal patterns. Critical values are estimated by bootstrap methods. Using this approach, not only the p-value of a test to check independence but also an envelope for $J(r)$ values is calculated. The test is based on the statistic $max_r(abs(J(r) - 1))$.

One disadvantage of the proposed test is that it may be quite sensitive to the values of r. In addition, in time processes, dependence often appears between close observations, and with high r values it is more difficult that the J-function is able to discriminate between dependent and independent processes. By this reason, the argument `rTest` allows us to fix a maximum value of r and only J(r) estimations for $r < rTest$ will be used to calculate the test statistic. The value `rTest` is drawn in the plot of the J-function as a vertical grey line, in order to help us to identify an adequate value.

The occurrence points in all the processes in C must be part of the input. Since the number of points will be possibly different in each process, a matrix cannot be used. Instead two vectors with the same length are used: the first one contains the occrrence points in all the processes while the second one indicates the point process in C where the point in the same row in `posC` has occurred. The codes used in `typeC` are the column number where the intensity of that process is in matrix `lambdaC`. The same for set D.

Value

A list with elements

`r`	Vector of values r where J-function is evaluated.
`NHJr`	Estimated values of function J(r).
`NHDr`	Estimated values of function D(r).

NHFr	Estimated values of function J(r).
JenvL	Lower limits of the envelope for J(r).
JenvU	Upper limits of the envelope for J(r).
JStatOb	Observed value of the statistic used to test the independence assumption.
JStatTr	Sample of the values of the test statistic obtained by random translation.
pv	P-value of the independence test.
T	Length of the observation period of the process.
L	Net of values L used to calculate F-funtion.

References

Cronie, O. and van Lieshout, M.N.M. (2015). Summary statistics for inhomogeneous marked point processes. Ann Inst Stat Math. DOI 10.1007/s10463-015-0515-z

Lotwick, H.W. and Silverman, B.W. (1982). Methods for analysing Spatial processes of several types of points. *J.R. Statist. Soc. B*, 44(3), pp. 406-13

Stoyan et al (2001). On the estimation of distance distribution functions for points processes and random sets. Image Anal Stereol, 20, 65-69

Van Lieshout, M.N.M. (2006) A J-function for marked point patterns. AISM, 58, 235-259. DOI 10.1007/s10463-005-0015-7

See Also

NHK.fun, NHD.fun, NHF.fun

Examples

```
set.seed(122)
lambda1<-runif(100, 0.05, 0.1)
set.seed(121)
lambda2<-runif(100, 0.01, 0.2)
pos1<-simNHPc.fun(lambda=lambda1,fixed.seed=123)$posNH
pos2<-simNHPc.fun(lambda=lambda2,fixed.seed=123)$posNH

aux<-NHJ.fun(lambdaC=lambda1, lambdaD=lambda2, posC=pos1,nTrans=50,
 posD=pos2, rTest=7, dplot='J', cores=1,test=TRUE)
aux$pv


#Sets with two processes
#pos3<-simNHPc.fun(lambda=lambda1,fixed.seed=321)$posNH
#pos4<-simNHPc.fun(lambda=lambda2,fixed.seed=321)$posNH
#aux<-NHJ.fun(lambdaC=cbind(lambda1,lambda2), lambdaD=cbind(lambda1,lambda2),
# posC=c(pos1,pos2), typeC=c(rep(1, length(pos1)), rep(2, length(pos2))),
# posD=c(pos3, pos4), typeD=c(rep(1, length(pos3)), rep(2, length(pos4))),
# dplot='J', test=TRUE)
#aux$pv
```

NHK.fun *It estimates the cross K-function for two sets of point processes*

Description

This function estimates the cross K-function between two sets, C and D, of (homogenous or non-homogeneous) time point processes. It is evaluated in a grid of distances r, and it can be optionally plotted. An independence test based on K-function is also available.

It calls the auxiliary functions NHKaux.fun, NHKaux2.fun, NHKaux3.fun and Kenv, not intended for users.

Usage

```
NHK.fun(lambdaC, lambdaD, T=NULL, posC, typeC=1, posD, typeD=1, r=NULL,
 test=TRUE, nTrans=100, conf=0.95, rTest=20, dplot=FALSE,
tit="K-function", cores=1, fixed.seed=NULL)
```

Arguments

`lambdaC`	A matrix. Each column is the intensity vector of one of the point processes in C. If there is only one process in C, it can be a vector or even a numeric value if the process is homogeneous.
`lambdaD`	A matrix. Each column is the intensity vector of one of the point process in D. If there is only one process in D, it can be a vector oe even a numeric value if the process is homogeneous.
`T`	Numeric value. Length of the observation period. It only must be specified if all the processes are homogeneous, that is if the number of rows in `lambdaC` and `lambdaD` is 1.
`posC`	Numeric vector. Time position of the points in all the point processes in C.
`typeC`	Numeric vector with the same length as `posC`. Code of the point process in C where the points in `posC` have occurred. See Details.
`posD`	Numeric vector. Time position of the points in all the point processes in D.
`typeD`	Numeric vector with the same length as `posD`. Code of the point process in D where the points in `posD` have occurred.
`r`	Optional. Numeric vector. Values where K-function must be evaluated. If it is NULL, a default vector is used, see Details
`test`	Optional. Logical flag. If it is TRUE, a test to check the independence and a 95% envelope for the K-function are calculated.
`nTrans`	Optional. Numeric value. Only used if `test=TRUE`. Number of translations to be performed in the test and envelope calculation.
`conf`	Optional. Numeric value in (0,1). Confidence level of the envelope for the K-function.
`rTest`	Optional. Numeric value. Maximum value of r used to calculate the independence test statistc, see Details.

`dplot`	Optional. A logical flag. If it is true a plot of the K-function is shown.
`tit`	Optional. Title to be used in the plot of the K-function.
`cores`	Optional. Number of cores of the computer to be used in the calculations.
`fixed.seed`	An integer or NULL. If it is an integer, that is the value used to set the seed in random generation processes. It it is NULL, a random seed is used.

Details

This function estimates the cross K-function between two sets, C and D, of (homogenous or nonhomogeneous) time point processes, using a version of the spatial estimator suggested by Moller and Waagepetersen (2007), p662, adapted to cross time processes; the edge correction suggested in that paper is also used. In the case of nonhomogeneous processes, the estimator is only defined for second order intensity-reweighted stationary point processes. Homogenous processes are generated if the intensity vector `lambda` is constant (that is if all the values are equal) or its length is 1.

K-function measures the degree of dependence between two point processes (or two sets of point processes) and counts the expected number of points in any of the processes in D, within a given distance of a point in any of the processes in C, adjusted for time varying intensity in the case of nonhomogenous processes.

The cross K-function of independent Poisson processes (conditionally on the marginal structure of the processes) is the length of the considered intervals, $K_{CD}(r) = 2r + 1$. Then, values $K_{CD}(r) > 2r + 1$ indicate attraction between the processes, while values lower than $2r + 1$ indicate repulsion.

If argument `r` is NULL, it is calculated as

r1<-max(20, floor(T/20))

r<-seq(1,r1,by=2)

if (length(r)>200) r<-seq(1,r1,length.out=200)

Testing independence

In order to test the independence hypothesis using this function, a Lotwick-Silverman type test is available. This test provides a nonparametric way to test independence when the study area is rectangular. The key idea is to wrap the processes onto a torus by identifying opposite sides, keeping the processes in set C fixed and translating the processes in set D randomly over the torus. In nonhomogenous processes, random translation may change their distribution, and to compensate, the intensity must also be translated. The considered translations keep the distribution of the processes in D but break any dependence between them, without the need of parametric models to describe the univariate marginal patterns. Critical values are estimated by bootstrap methods. Using this approach, not only the p-value of a test to check independence but also an envelope for the $K(r)$ values is calculated. The test is based on the statistic $max_r(K(r)/(2r + 1))$.

One disadvantage of the proposed test is that it may be quite sensitive to the values of r. In addition, in time processes, dependence often appears between close observations, and with high r values it is more difficult that the K-function is able to discriminate between dependent and independent processes. By this reason, the argument `rTest` allows us to fix a maximum value of r and only K(r) estimations for $r < rTest$ will be used to calculate the test statistic. The value `rTest` is drawn in the plot of the K-function as a vertical grey line, in order to help us to identify an adequate value.

The occurrence points in all the processes in C must be part of the input. Since the number of points will be possibly different in each process, a matrix cannot be used. Instead two vectors with the same length are used: the first one contains the occrrence points in all the processes while the second one indicates the point process in C where the point in the same row in `posC` has occurred. The codes used in `typeC` are the column number where the intensity of that process is in matrix `lambdaC`. The same for set D.

Value

A list with elements

`r`	Vector of values r where K-function is evaluated.
`NHKr`	Estimated values of function K(r).
`KenvL`	Lower limits of the envelope for K(r).
`KenvU`	Upper limits of the envelope for K(r).
`KStatOb`	Observed value of the statistic used to test the independence assumption.
`KStatTr`	Sample of the values of the test statistic obtained by random translation.
`pv`	P-value of the independence test.
`T`	Length of the observation period of the process.

References

Cronie, O. and van Lieshout, M.N.M. (2015). Summary statistics for inhomogeneous marked point processes. Ann Inst Stat Math. DOI 10.1007/s10463-015-0515-z

Lotwick, H.W. and Silverman, B.W. (1982). Methods for analysing Spatial processes of several types of points. *J.R. Statist. Soc. B*, 44(3), pp. 406-13

Moller, J. and Waagepetersen, R.P. (2007). Modern statistics for spatial point processes. Scandinavian Journal of Statistics, 34(4), 643 684

See Also

`NHD.fun`, `NHJ.fun`, `NHF.fun`

Examples

```
set.seed(122)
lambda1<-runif(100, 0.05, 0.1)
set.seed(121)
lambda2<-runif(100, 0.01, 0.2)
pos1<-simNHPc.fun(lambda=lambda1, fixed.seed=123)$posNH
pos2<-simNHPc.fun(lambda=lambda2, fixed.seed=123)$posNH

aux<-NHK.fun(lambdaC=lambda1, lambdaD=lambda2, posC=pos1, posD=pos2, rTest=7,
 dplot=TRUE,tit="K(r).", nTrans=75, cores=1)
aux$pv




#Sets with two processes
#pos3<-simNHPc.fun(lambda=lambda1, fixed.seed=321)$posNH
```

```
#pos4<-simNHPc.fun(lambda=lambda2, fixed.seed=321)$posNH
#aux<-NHK.fun(lambdaC=cbind(lambda1,lambda2), lambdaD=cbind(lambda1,lambda2), posC=c(pos1,pos2),
# typeC=c(rep(1, length(pos1)), rep(2, length(pos2))), posD=c(pos3, pos4),
# typeD=c(rep(1, length(pos3)), rep(2, length(pos4))), dplot=TRUE)
#aux$pv
```

`simHPc.fun` *Generate a given number of occurrence points in a homogenous Poisson process in continuous time*

Description

This function generates a given number of occurrence points in a homogenous Poisson process (HPP) in continuous time.

Usage

```
simHPc.fun(lambda, nEv, fixed.seed=NULL)
```

Arguments

`lambda`	Numeric value. Intensity λ used to generate the PP.
`nEv`	Optional. Positive integer. Number of points to be generated in the PPs.
`fixed.seed`	An integer or NULL. If it is an integer, that is the value used to set the seed in random generation processes. It it is NULL, a random seed is used.

Details

The points in a HPP are generated using independent exponentials with mean λ.

Points in a HPP can also be generated using `simNHPc.fun`. The main difference is that here, the number of points to be generated is given, while in the other function, points are generated in a period of a given length T.

Value

A list with elements

`posN`	Numeric vector. Occurrence points of the PP.
`T`	Length of the period where the given number of points are generated.
`fixed.seed`	Input argument.

References

Ross, S.M. (2006). *Simulation.* Academic Press.

See Also

`simNHPc.fun`, `IndNHPP.fun`

Examples

```
aux<-simHPc.fun(lambda=0.01, nEv=50,fixed.seed=123)
aux$posH
```

`simNHPc.fun` *Generate the occurrence points of a Poisson process in continuous time*

Description

This function generates the occurrence points of a homogenous or nonhomogeneous Poisson process (NHPP) with a given intensity $\lambda(t)$, in a continuous period of time (0, T).

It calls the auxiliary function `buscar` (not intended for the users), see Details section.

Usage

```
simNHPc.fun(lambda, fixed.seed=NULL)
```

Arguments

`lambda`	Numeric vector of length T, the obsevation period length. Intensity $\lambda(t)$ used to generate the PP.
`fixed.seed`	An integer or NULL. If it is an integer, that is the value used to set the seed in random generation processes. It it is NULL, a random seed is used.

Details

The generation of the NHPP points consists in two steps. First, the points of a homogeneous PP of intensity 1 are generated using independent exponentials. Then, the homogeneous occurrence times are transformed into the points of a non homogeneous process with intensity $\lambda(t)$. This transformation is performed by the auxiliary function `buscar` (not intended for the user).

The lenght of the period where the processes are generated is determined by the length of the argument `lambda`.

Homogenous processes are generated if the intensity vector `lambda` is constant (that is if all the values are equal).

Value

A list with elements

`posNH`	Numeric vector. Occurrence points of the PP.
`lambda`	Input argument.
`fixed.seed`	Input argument.

References

Cebrian, A.C., Abaurrea, J. and Asin, J. (2015). NHPoisson: An R Package for Fitting and Validating Nonhomogeneous Poisson Processes. *Journal of Statistical Software*, 64(6), 1-24.

Ross, S.M. (2006). *Simulation.* Academic Press.

See Also

`simHPc.fun`, `IndNHPP.fun`

Examples

```

#Generation  of a HPP
aux<-simNHPc.fun(lambda=rep(0.1,200),fixed.seed=123)
aux$posNH


#Generation of a NHPP
set.seed(123)
lambdat<-runif(500, 0.01,0.1)
aux<-simNHPc.fun(lambda=lambdat,fixed.seed=123)
aux$posNH
```

`SpecGap.fun` *It calculates the stationary distribution of a matrix and its spectral gap*

Description

This function calculates the stationary distribution of the transition matrix of a Markov chain process and its spectral gap.

Usage

```
SpecGap.fun(P)
```

Arguments

`P`	Matrix. It must be a markovian matrix

Details

The spectral gap of a matrix P assesses the convergence speed of P to a matrix P_I with all the rows equal to $(\pi_1, \pi_2, ...\pi_k)$, the stationary distribution of P. It takes values in [0,1].

The spectral gap of a transition matrix can be used as a dependence measure between the multitype processes defined by a marked Poisson procces with marks generated by a Markov chain with that transition matrix, see Abaurrea et al (2015) for details.

Value

A list with elements

`SG`	Spectral gap value of the matrix.
`pi`	Vector of the stationary distribution of the matrix.

References

Abaurrea, J. Asin, J. and Cebrian, A.C. (2014). A Bootstrap Test of Independence Between Three Temporal Nonhomogeneous Poisson Processes and its Application to Heat Wave Modeling. *Environmental and Ecological Statistics*, 22(1), 127-144.

See Also

`DepMarkedNHPP.fun`

Examples

```
P<-cbind(c(0.7, 0.1, 0.2), c(0.2, 0.7, 0.1), c(0.1, 0.2, 0.7))
SpecGap.fun(P)
```

`TestIndLS.fun` *Calculate a Lotwick-Silverman test to check the independence between up to three homogeneous point processes in time*

Description

This function calculates a test of type Lotwick-Silverman (LS) to check the independence between up to three homogeneous point processes in time. The statistic is based on the close point relation, which adapts the crossed nearest neighbour distance ideas of spatial point processes to the case of point processes in time. The p-value is obtained using a bootstrap based on a type LS approach.

Usage

```
TestIndLS.fun(posx, posy, posz=NULL, T,  alpha = 0.05, nTrans = 100,
PA = FALSE, cores=1,fixed.seed=NULL)
```

Arguments

`posx`	Numeric vector. Position of the occurrence points in the first process.
`posy`	Numeric vector. Position of the occurrence points in the second process.
`posz`	Numeric vector. Position of the occurrence points in the third process. Only used if there are 3 processes in the set.
`T`	Numeric value. Length of the observation period of the processes.

`alpha`	Optional. Significance level used to obtain a decision (reject-no reject) based on the test p-value.
`nTrans`	Optional. Positive integer. Number of translations performed to calculate the test.
`PA`	Optional. Logical flag. If it is TRUE, the close point relation is broadened by including the previous and the following points to the overlapping intervals.
`cores`	Optional. Number of cores of the computer to be used in the calculations.
`fixed.seed`	Optional. An integer or NULL. If it is an integer, that is the value used to set the seed in random generation processes. It it is NULL, a random seed is used.

Details

The underlying idea of the test is to compare, for each point in the first process, the behavior of its set of close points in the observed process vector (N_x, N_y, N_z), and in a new process vector with the same characteristics and mutually independent components. In the new vectors, the process N_x is fixed and second and third processes are obtained by a translation of the original ones. If the observed behavior is significantly different, independence is rejected. The test statistic is the same as the one used in `TestIndNH.fun`, but the p-value is obtained using a different bootstrap technique, based on a LS approach, see Lotwick and Silverman (1982).

This test provides a nonparametric way to test independence. The key idea is to wrap the processes onto a torus by identifying opposite sides, keeping the first process fixed and translating the other processes randomly over the torus. The translation keeps the distribution of the processes (since they are homogenous) but breaks any dependence between them, without the need of parametric models to describe the univariate marginal patterns, not even the intensity of the processes.

Value

A list with elements

`KSpv`	P-value of the independence test.
`reject`	Binary vale indicating if the test is rejected (value 1) or not (value 0) at the alpha significance level.
`KSest`	Statistic of the independence test.

References

Abaurrea, J. Asin, J. and Cebrian, A.C. (2015). A Bootstrap Test of Independence Between Three Temporal Nonhomogeneous Poisson Processes and its Application to Heat Wave Modeling. *Environmental and Ecological Statistics*.

Lotwick, H.W. and Silverman, B.W. (1982). Methods for analysing Spatial processes of several types of points. *J.R. Statist. Soc. B*, 44(3), pp. 406-13

See Also

`TestIndNH.fun`, `CondTest.fun`, `DutilleulPlot.fun`, `DistShift.fun`

Examples

```
#Test applied to three independent NHPP
posx<-simHPc.fun(0.1, 20, fixed.seed=123)$posH
posz<-simHPc.fun(0.15,22, fixed.seed=124)$posH
posy<-simHPc.fun(0.1, 18, fixed.seed=125)$posH
T<-max(posx,posy,posz)+10

aux<-TestIndLS.fun(posx, posy, posz,T=T,
cores=1,fixed.seed=321)
aux$KSpv
```

`TestIndNH.fun` *Calculate a bootstrap test to check the independence between up to three nonhomogeneous Poisson or point processes in time*

Description

This function calculates the test by Abaurrea et al. (2014). The statistic is based on the close point relation, which adapts the crossed nearest neighbour distance ideas of spatial point processes to the case of (homogenoeus or nonhomogeneous) point processes in time. The test can be applied to two or three homogeneous or non homogeneous Poisson processes or to any point process which can be simulated. Currently, it is implemented for Poisson processes and for Neyman-Scott cluster processes. The p-value is obtained using a parametric bootstrap approach.

Usage

```
TestIndNH.fun(posx, posy, posz=NULL, NumProcess=2, alpha = 0.05,
nsim = 100, PA = FALSE, cores = 1, type = "Poisson",
lambdaMarg = NULL,   lambdaParent = NULL, lambdaNumP = NULL,
dist = "normal", sigmaC = 1, minC = -1, maxC = 1,fixed.seed=NULL)
```

Arguments

`posx`	Numeric vector. Position of the occurrence points in the first process.
`posy`	Numeric vector. Position of the occurrence points in the second process.
`posz`	Numeric vector. Position of the occurrence points in the third process. Only used if `NumProcess=3`.
`NumProcess`	Optional. Integer equal to 1 or 2. Number of processes added to the first one.
`alpha`	Optional. Significance level used to obtain a decision (reject-no reject) based on the test p-value.
`nsim`	Optional. Positive integer. Number of simulations performed to calculate the test.
`PA`	Optional. Logical flag. If it is TRUE, the close point relation is broadened by including the previous and the following points to the overlapping intervals.

`cores`	Optional. Number of cores of the computer to be used in the calculations.
`type`	Optional. Label "Poisson" or "PoissonCluster". Type of point processes to be generated in the parametric bootstrap. Up to now, only two types are available: Poisson processes ("Poisson") and Neyman-Scott cluster processes ("Poisson-Cluster").
`lambdaMarg`	Matrix of dimension $T \times (NumProcess - 1)$. Only used if *type='Poisson'*. Each column is the intensity vector to generate the Poisson processes.
`lambdaParent`	Numeric vector. Only used if *type='PoissonCluster'*. Intensity vector of the PP used to generate the centers of the clusters of the Neyman-Scott process.
`lambdaNumP`	Numeric vector with 1 or 2 values. Only used if *type='PoissonCluster'*. Mean values of the number of sons of the processes to be generated. If its length is equal to 1, the same value is used in both processes.
`dist`	Optional. Label "normal" or "uniform". Only used if *type='PoissonCluster'*. Distribution used to generate the point locations of each cluster.
`sigmaC`	Optional. Numeric vector with 1 or 2 values. Only used if *type='PoissonCluster'* and *dist='normal'*. Standard deviation of the normal distribution. If its length is equal to 1, the same value is used in both processes.
`minC`	Optional. Numeric vector with 1 or 2 values. Only used if *type='PoissonCluster'* and *dist='uniform'*. Lower limits of the Uniform distribution. If its lengthis equal to 1, the same value is used in both processes.
`maxC`	Optional. Numeric vector with 1 or 2 values. Only used if *type='PoissonCluster'* and *dist='uniform'*. Upper limits of the Uniform distribution. If its lengthis equal to 1, the same value is used in both processes.
`fixed.seed`	Optional. An integer or NULL. If it is an integer, that is the value used to set the seed in random generation processes. It it is NULL, a random seed is used.

Details

The underlying idea of the test is to compare, for each point in the first process, the behavior of its set of close points in the observed vector process (N_x, N_y, N_z), and in simulated vector processes with the same characteristics and mutually independent components. In the simulated vectors, the process N_x is fixed and second and third processes with intensities λ_y and λ_z (the intensities of N_y and N_z) are generated. If the observed behavior is significantly different, independence is rejected. The distribution of the statistic is obtained using a Monte Carlo approach if the intensities $\lambda_y(t)$ and $\lambda_z(t)$ are known, or a parametric bootstrap if they have been estimated. See Abaurrea et al. (2014) for more details

It is noteworthy that is being assumed that the processes are Poisson or Neyman-Scott cluster processes. Then, if necessary, validation of that assumption should be previously carried out.

The lenght of the observation period is determined by the length of the intensity vector λ, that is *lambdaParent* (if *type='PoissonCluster'*) or the first element of the dimension of *lambdaMarg* (if *type='PoissonC'*.

Homogenous processes are generated if the intensity vector `lambda` is constant (that is if all the values are equal).

The test `TestIndLS.fun`, which uses a non parametric bootstrap approach, is available for any homogeneous point process.

Value

A list with elements

`KSpv`	P-value of the independence test.
`reject`	Binary vale indicating if the test is rejected (value 1) or not (value 0) at the alpha significance level.
`KSest`	Statistic of the independence test.

References

Abaurrea, J. Asin, J. and Cebrian, A.C. (2014). A Bootstrap Test of Independence Between Three Temporal Nonhomogeneous Poisson Processes and its Application to Heat Wave Modeling. *Environmental and Ecological Statistics*.

See Also

`TestIndLS.fun`,`CondTest.fun`, `DutilleulPlot.fun`, `DistSim.fun`, `DistObs.fun`, `uniongentri.fun`

Examples

```
#Test applied to 3  independent NHPP
set.seed(123)
lambdax<-runif(150, 0.01,0.1)
set.seed(124)
lambday<-runif(150, 0.02,0.1)
set.seed(125)
lambdaz<-runif(150, 0.015,0.1)
posx<-simNHPc.fun(lambdax, fixed.seed=123)$posNH
posy<-simNHPc.fun(lambday, fixed.seed=124)$posNH
posz<-simNHPc.fun(lambdaz, fixed.seed=125)$posNH

aux<-TestIndNH.fun(posx, posy, posz, nsim=50, type='Poisson',
NumProcess=3,lambdaMarg=cbind(lambday,lambdaz), fixed.seed=321)
aux$KSpv




#Test applied to 3 dependent NS cluster processes with 2 cores
#set.seed(123)
#lambdaParent<-runif(500,0,0.1)
#DepPro<-DepNHNeyScot.fun(lambdaParent=lambdaParent, d=3, lambdaNumP = 3,
#  dist = "normal", sigmaC = 1, fixed.seed=123,cores=2)
#posx<-DepPro$PP1
#posy<-DepPro$PP2
#posz<-DepPro$PP3
#aux<-TestIndNH.fun(posx, posy, posz, cores=1, type='PoissonCluster',
# NumProcess=3,lambdaParent = lambdaParent, lambdaNumP = 3,
# dist = "normal", sigmaC = 1, fixed.seed=123, nsim=200)
#aux$KSpv
```

uniongentri.fun *Calculates the set of close points of each point in a process*

Description

This function calculates the set of close points of each occurence point in the first process of a set of uo to three processes.

Usage

```
uniongentri.fun(posx, posy, posz=NULL,  info = FALSE, PA = FALSE,
procName=c('X','Y','Z'),...)
```

Arguments

`posx`	Numeric vector. Position of the occurrence points in the first process.
`posy`	Numeric vector. Position of the occurrence points in the second process.
`posz`	Optional. Numeric vector. Position of the occurrence points in the third process. Only used when three processes are involved.
`info`	Optional. Logical flag. If it is TRUE, information about the generated points is shown on the screen and dotcharts and bivariate charts of the occurrence points in the processes are displayed.
`procName`	Vector of character strings. Names for the processes.
`PA`	Optional. Logical flag. If it is TRUE, the close point relation is broadened by including the previous and the following points to the overlapping intervals.
`...`	Further arguments to be passed to `plot` if `info=T`.

Details

A point in a process is close to a point in another process, if their time intervals overlap; the time interval of a point is the time interval between itself and the previous point. If there are three processes, the set of close points of t_{x_k}, $S_{x_i;xyz}$ is defined as the set of the pairs of points (t_{y_j}, t_{z_k}) such that t_{x_i} is close to t_{y_j} and t_{y_j} is close to t_{z_k}. If there are two processes, $S_{x_i;xy}$ is the set of points t_{y_j} such that t_{x_i} is close to t_{y_j}.

The algortihm to calculate the sets of close points (in the case of three processes) is the following, see Abaurrea et al. (2015) for details: First, given two processes, the pairs of close points in those processes are calculated. If the last point occurs in the first process, there is a censored time interval in the second process (the point overlaps a time interval whose occurrence point has not been observed) and that pair is not considered). This step is performed for all the combinations of pairs of processes. The basic close point relation is commutative, and only three different pairs (XY, YZ, XZ) must be considered. This is not the case of the broadened definition, where two more points (the previous and the following ones) are added to the set; in that case, the six pairs (XY, YX, YZ, ZY, XZ, ZX) must be calculated.

Once all the pairs of close points are obtained, the set of close points for each point t_{x_i} is obtained by concatenating the adequate pairs of points from all the possible orders of the three processes: XYZ, XZY and YXZ for the basic definition, and the six possible permutations for the broadened definition. The final set of close points of t_{x_i} is the union of the different pairs from all the possible permutations.

Value

A list with elements

`X`	First elements of the 3-tuples of points $(t_{x_i}, t_{y_i}, t_{z_i})$ in the sets of close points
`iX`	Position i (=1,2,3....) of the point t_{x_i} in the first process
`Y`	Second elements of the 3-tuples of points $(t_{x_i}, t_{y_i}, t_{z_i})$ in the sets of close points
`iY`	Position i (=1,2,3....) of the point t_{y_i} in the second process
`Z`	Third elements of the 3-tuples of points $(t_{x_i}, t_{y_i}, t_{z_i})$ in the sets of close points. It is NULL if `posz=NULL`.
`iZ`	Position i (=1,2,3....) of the point t_{z_i} in the third process. It is NULL if posz=NULL.

References

Abaurrea, J. Asin, J. and Cebrian, A.C. (2014). A Bootstrap Test of Independence Between Three Temporal Nonhomogeneous Poisson Processes and its Application to Heat Wave Modeling. *Environmental and Ecological Statistics*.

See Also

`TestIndNH.fun`, `DistSim.fun`, `DistObs.fun`

Examples

```
set.seed(123)
posx<-sort(runif(20,0,1000))
posy<-sort(runif(25,0,1000))
posz<-sort(runif(40,0,1000))
aux<-uniongentri.fun(posx, posy, posz, info=TRUE)
```

Index

*Topic **Tests of Independence Between Point Processes in Time**
IndTestPP-package, 2

BarTxTn, 3
buscar *(simNHPc.fun)*, 41

calcdist.fun, *19*
calcdist.fun *(DistObs.fun)*, 19
calcNmu *(CondTest.fun)*, 5
ComplPos.fun, 4
CondTest.fun, 5, *12*, *25*, *44*, *47*
CPSPpoints.fun, 7, *10*
CPSPPOTevents.fun, *8*, 8

depchi.fun, 10
DepMarkedNHPP.fun, 12, *14*, *16*, *18*, *43*
DepNHCPSP.fun, *13*, 13, *16*, *18*
DepNHNeyScot.fun, *13*, *14*, 15, *18*, *27*
DepNHPPqueue.fun, *13*, *14*, *16*, 17
DistObs.fun, 19, *23*, *47*, *49*
DistShift.fun, 20, *44*
DistSim.fun, *20*, *21*, 22, *47*, *49*
DistSimfix.fun *(DistSim.fun)*, 22
dotchart, *20*, *23*
DutilleulPlot.fun, *6*, *12*, 24, *28*, *44*, *47*

fn2 *(TestIndNH.fun)*, 45
fn2B *(TestIndLS.fun)*, 43
fn2fix *(TestIndNH.fun)*, 45
fn3 *(DutilleulPlot.fun)*, 24

genbiPos.fun *(uniongentri.fun)*, 48
GenSons.fun *(DepNHNeyScot.fun)*, 15
gentriPos.fun *(uniongentri.fun)*, 48

HDFaux *(NHD.fun)*, 29

IndNHNeyScot.fun, *16*, *23*, 25, *28*
IndNHPP.fun, *13*, *18*, *27*, 27, *40*, *42*
IndTestPP *(IndTestPP-package)*, 2
IndTestPP-package, 2

Jenv *(NHJ.fun)*, 33

Kenv *(NHK.fun)*, 37

miKS.fun *(TestIndNH.fun)*, 45
mirank.fun *(TestIndNH.fun)*, 45

nearestD *(NHD.fun)*, 29
nearestdist.fun, *25*, 28
NHD.fun, *4*, 29, *33*, *36*, *39*
NHDaux *(NHD.fun)*, 29
NHDFaux *(NHJ.fun)*, 33
NHF.fun, *4*, *30*, 31, *36*, *39*
NHFaux *(NHF.fun)*, 31
NHJ.fun, *4*, *30*, *32*, *33*, 33, *39*
NHJaux *(NHJ.fun)*, 33
NHK.fun, *4*, *30*, *33*, *36*, 37
NHKaux *(NHK.fun)*, 37
NHKaux2 *(NHK.fun)*, 37
NHKaux3 *(NHK.fun)*, 37
nMenr *(NHD.fun)*, 29

par, *34*
pdist.fun *(nearestdist.fun)*, 28
plot, *5*, *19*, *20*, *23*, *24*, *48*
prodN2 *(NHD.fun)*, 29

rexp, *17*

simHPc.fun, 40, *42*
simNHPc.fun, *23*, *27*, *28*, *40*, 41
SpecGap.fun, *13*, 42

TestIndLS.fun, *21*, 43, *46*, *47*
TestIndNH.fun, *6*, *12*, *20*, *23*, *25*, *44*, 45, *49*

uniongentri.fun, *19*, *20*, *47*, 48