Package ‘NScluster’

March 31, 2023

Version 1.3.6-1

Title Simulation and Estimation of the Neyman-Scott Type Spatial Cluster Models

Depends R (>= 3.0.0)

Imports graphics, stats, utils, methods

Description Simulation and estimation for Neyman-Scott spatial cluster point process models and their extensions, based on the methodology in Tanaka, Ogata, and Stoyan (2008) <doi:10.1002/bimj.200610339>. To estimate parameters by the simplex method, parallel computation using 'OpenMP' application programming interface is available. For more details see Tanaka, Saga and Nakano <doi:10.18637/jss.v098.i06>.

License GPL (>= 2)

MailingList Please send bug reports to ismrp@jasp.ism.ac.jp

NeedsCompilation yes

Author Ushio Tanaka [aut] (Fortran original),
Masami Saga [aut, cre],
Junji Nakano [aut]

Maintainer Masami Saga <msaga@mtb.biglobe.ne.jp>

R topics documented:

`NScluster-package` *Simulation and Estimation of the Neyman-Scott Type Spatial Cluster Models*

Description

NScluster involves the maximum Palm likelihood estimation procedure for Neyman-Scott cluster point process models and their extensions with parallel computation using OpenMP technology. The maximum Palm likelihood estimates (MPLEs for short) are those that maximize the log-Palm likelihood function. The computation of MPLEs is implemented by simplex maximization with parallel computation via OpenMP. Together with the likelihood estimation procedure, NScluster also provides a simulation procedure for cluster point process models.

Details

The documentation 'A Guide to NScluster: R Package for Maximum Palm Likelihood Estimation for Cluster Point Process Models using OpenMP' is available in the package vignette using the `vignette` function (e.g., `vignette("NScluster")`).

The package NScluster comprises of four tasks: simulation, parameter estimation (MPLE), confidence interval estimation, and non-parametric and parametric Palm intensity comparison.

- Simulation:

  The `sim.cppm` function simulates the five cluster point process models: the Thomas and Inverse-power type models, and the extended Thomas models of type A, B, and C.

- Parameter estimation (MPLE):

  The `mple.cppm` function improves the given initial parameters using the simplex method to maximize the log-Palm likelihood function.

  The expensive calculation of the estimation for calculating the parameters can be parallelized to reduce calculation time. The package is implemented to employ OpenMP, which is a simple framework for shared memory parallel computation.

- Confidence interval of parameter estimates:

  The `boot.mple` function carries out the bootstarp replicates for an object generated by `mple.cppm` and computes confidence intervals and standard errors.

- Palm intensity comparison:

  The package can depict non-parametric and parametric normalized Palm intensity function of the five cluster point process models using the `palm.cppm` function.

References

Tanaka, U., Ogata, Y. and Katsura, K. (2008) Simulation and estimation of the Neyman-Scott type spatial cluster models. *Computer Science Monographs* **34**, 1-44. The Institute of Statistical Mathematics, Tokyo. https://www.ism.ac.jp/editsec/csm/

Tanaka, U., Ogata, Y. and Stoyan, D. (2008) Parameter estimation and model selection for Neyman-Scott point processes. *Biometrical Journal* **50**, 43-57.

Tanaka, U., Saga, M. and Nakano, J. (2021) NScluster: An R Package for Maximum Palm Likelihood Estimation for Cluster Point Process Models Using OpenMP. *Journal of Statistical Software*, **98(6)**, 1-22. doi:10.18637/jss.v098.i06.

`boot.mple` *Bootstrap resampling for MPLE*

Description

Carry out bootstrap replicates of MPLE on simulated data.

Usage

```
boot.mple(mple.out, n = 100, conf.level = 0.95, se = TRUE, trace = FALSE)

## S3 method for class 'boot.mple'
summary(object, ...)
```

Arguments

`mple.out`	an object of class "mple", usually the result of a call to `mple.cppm`.
`n`	number of bootstrap replicates performed.
`conf.level`	the confidence level required.
`se`	logical. If `TRUE` standard errors are returned.
`trace`	logical: if `TRUE`, a progress bar is shown.
`object`	an object of class `"boot.mple"`.
`...`	ignored.

Value

`boot.mple` returns an object of class `"boot.mple"` containing the following components:

`boot.mples`	a matrix of `n` rows each of which is a bootstrap replicate of the result of calling `mple.cppm`.
`confint`	confidence intervals for MPLEs.
`mple`	MPLE of `mple.out` passed as `'pars'` argument to `sim.cppm`.

Examples

```
### Thomas Model
# simulation
pars <- c(mu = 50.0, nu = 30.0, sigma = 0.03)
t.sim <- sim.cppm("Thomas", pars, seed = 117)

## Not run:  # estimation (need long CPU time)
init.pars <- c(mu = 40.0, nu = 40.0, sigma = 0.05)
```

```
t.mple <- mple.cppm("Thomas", t.sim$offspring$xy, init.pars)
t.boot <- boot.mple(t.mple)
summary(t.boot)

## End(Not run)
```

`mple.cppm` *MPLE of Neyman-Scott Cluster Point Process Models and Their Extensions*

Description

MPLE of the five cluster point process models.

Usage

```
mple.cppm(model = "Thomas", xy.points, pars = NULL, eps = 0.001, uplimit = 0.3,
          skip = 1)

## S3 method for class 'mple'
coef(object, ...)
## S3 method for class 'mple'
summary(object, ...)
```

Arguments

`model`	a character string indicating each cluster point process model: `"Thomas"`, `"IP"`, `"TypeA"`, `"TypeB"`, and `"TypeC"`.
`xy.points`	a matrix containing the coordinates `(x,y)` of points in $W = [0,1] \times [0,1]$.
`pars`	a named vector containing a given initial guess of each parameter. If `NULL`, any suitable parameters are used. See Details' in `sim.cppm` for the parameters of each model.
`eps`	the sufficiently small number to implement the optimization procedure for the log-Palm likelihood function. The procedure is iterated at most 1000 times until the `process2$stderr` becomes smaller than the `eps`.
`uplimit`	upper limit in place of ∞ of the integral in the probability distribution function relative to the random distance between two descendant points within the same cluster. The `uplimit` is valid for `"IP"` and `"TypeA"`.
`skip`	the variable enables one to obtain speedily the initial MPLEs, but rough approximation. The `skip` calculates the Palm intensity function of the log-Palm likelihood function for every `skip`-th r_{ij} in the ordered distances of the pairs i and j. The `skip` is valid for `"IP"` and `"TypeA"`.
`object`	an object of class `"mple"`.
`...`	ignored.

Details

- `"Thomas"` (Thomas model)
  The Palm intensity function is given as follows:
  For all $r \geq 0$,

$$\lambda_{\boldsymbol{o}}(r) = \mu\nu + \frac{\nu}{4\pi\sigma^2} \exp\left(-\frac{r^2}{4\sigma^2}\right).$$

  The log-Palm likelihood function is given by

$$\log L(\mu, \nu, \sigma) = \sum_{\{i,j;i<j,r_{ij}\leq 1/2\}} \log \nu \left\{\mu + \frac{1}{4\pi\sigma^2} \exp\left(-\frac{{r_{ij}}^2}{4\sigma^2}\right)\right\}$$

$$-N(W)\nu \left\{\frac{\pi\mu}{4} + 1 - \exp\left(-\frac{1}{16\sigma^2}\right)\right\}.$$

- `"TypeB"` (Type B model)
  The Palm intensity function is given as follows:
  For all $r \geq 0$,

$$\lambda_{\boldsymbol{o}}(r) = \lambda + \frac{\nu}{4\pi} \left\{\frac{a}{{\sigma_1}^2} \exp\left(-\frac{r^2}{4{\sigma_1}^2}\right) + \frac{(1-a)}{{\sigma_2}^2} \exp\left(-\frac{r^2}{4{\sigma_2}^2}\right)\right\},$$

  where $\lambda = \nu(\mu_1 + \mu_2)$ and $a = \mu_1/(\mu_1 + \mu_2)$ are the total intensity and the ratio of the intensity of the parent points of the smaller cluster to the total one, respectively.
  The log-Palm likelihood function is given by
  $\log L(\lambda, \alpha, \beta, \sigma_1, \sigma_2)$

$$= \sum_{\{i,j;i<j,r_{ij}\leq 1/2\}} \log \left[\lambda + \frac{1}{4\pi} \left\{\frac{\alpha}{{\sigma_1}^2} \exp\left(-\frac{{r_{ij}}^2}{4{\sigma_1}^2}\right) + \frac{\beta}{{\sigma_2}^2} \exp\left(-\frac{{r_{ij}}^2}{4{\sigma_2}^2}\right)\right\}\right]$$

$$-N(W) \left[\frac{\pi\lambda}{4} + \alpha \left\{1 - \exp\left(-\frac{1}{16{\sigma_1}^2}\right)\right\} + \beta \left\{1 - \exp\left(-\frac{1}{16{\sigma_2}^2}\right)\right\}\right],$$

  where $\alpha = a\nu$ and $\beta = (1-a)\nu$.

- `"TypeC"` (Type C model)
  The Palm intensity function is given as follows:
  For all $r \geq 0$,

$$\lambda_{\boldsymbol{o}}(r) = \lambda + \frac{1}{4\pi} \left\{\frac{a\nu_1}{{\sigma_1}^2} \exp\left(-\frac{r^2}{4{\sigma_1}^2}\right) + \frac{(1-a)\nu_2}{{\sigma_2}^2} \exp\left(-\frac{r^2}{4{\sigma_2}^2}\right)\right\},$$

  where $\lambda = \mu_1\nu_1 + \mu_2\nu_2$ and $a = \mu_1\nu_1/\lambda$ are the total intensity and the ratio of the intensity of the smaller cluster to the total one, respectively.
  The log-Palm likelihood function is given by

$$\log L(\lambda, \alpha, \beta, \sigma_1, \sigma_2)$$

$$= \sum_{\{i,j;i<j,r_{ij}\le 1/2\}} \log\left[\lambda + \frac{1}{4\pi}\left\{\frac{\alpha}{{\sigma_1}^2}\exp\left(-\frac{{r_{ij}}^2}{4{\sigma_1}^2}\right) + \frac{\beta}{{\sigma_2}^2}\exp\left(-\frac{{r_{ij}}^2}{4{\sigma_2}^2}\right)\right\}\right]$$

$$-N(W)\left[\frac{\pi\lambda}{4} + \alpha\left\{1-\exp\left(-\frac{1}{16{\sigma_1}^2}\right)\right\} + \beta\left\{1-\exp\left(-\frac{1}{16{\sigma_2}^2}\right)\right\}\right],$$

where $\alpha = a\nu_1$ and $\beta = (1-a)\nu_2$.

For the inverse-power model and the Type A models, we need to take the alternative form without explicit representation of the Palm intensity function. See the second reference below for details.

Value

`mple.cppm` returns an object of class `"mple"` containing the following main components:

`mple` MPLE (maximum Palm likelihood estimate).

`log.mpl` the log maximum Palm likelihood.

`aic` AIC.

`process1` a list with following components.

- **cflg** 1 (="update") or -1 (="testfn"), where "update" indicates that -log L value has attained the minimum so far, otherwise not.
- **logl** the minimized -log L in the process to minimize the negative log-Palm likelihood function.
- **mples** corresponding MPLEs.

`process2` a list with following components.

- **logl.simplex** the minimized -log L by the simplex method.
- **stderr** standard error.
- **pa.normal** the normalized variables corresponding to the MPLEs relative to the initial estimates.

There are other methods `plot.mple` and `print.mple` for this class.

References

Tanaka, U., Ogata, Y. and Katsura, K. (2008) Simulation and estimation of the Neyman-Scott type spatial cluster models. *Computer Science Monographs* **34**, 1-44. The Institute of Statistical Mathematics, Tokyo. https://www.ism.ac.jp/editsec/csm/.

Tanaka, U., Ogata, Y. and Stoyan, D. (2008) Parameter estimation and model selection for Neyman-Scott point processes. *Biometrical Journal* **50**, 43-57.

Examples

```
## Not run:
# The computation of MPLEs takes a long CPU time in the minimization procedure,
# especially for the Inverse-power type and the Type A models.

### Thomas Model
 # simulation
 pars <- c(mu = 50.0, nu = 30.0, sigma = 0.03)
 t.sim <- sim.cppm("Thomas", pars, seed = 117)
 ## estimation
 init.pars <- c(mu = 40.0, nu = 40.0, sigma = 0.05)
 t.mple <- mple.cppm("Thomas", t.sim$offspring$xy, init.pars)
 coef(t.mple)

### Inverse-Power Type Model
 # simulation
 pars <- c(mu = 50.0, nu = 30.0, p = 1.5, c = 0.005)
 ip.sim <- sim.cppm("IP", pars, seed = 353)
 ## estimation
 init.pars <- c(mu = 55.0, nu = 35.0, p = 1.0, c = 0.01)
 ip.mple <- mple.cppm("IP", ip.sim$offspring$xy, init.pars, skip = 100)
 coef(ip.mple)

### Type A Model
 # simulation
 pars <- c(mu = 50.0, nu = 30.0, a = 0.3, sigma1 = 0.005, sigma2 = 0.1)
 a.sim <- sim.cppm("TypeA", pars, seed = 575)
 ## estimation
 init.pars <- c(mu = 60.0, nu = 40.0, a = 0.5, sigma1 = 0.01, sigma2 = 0.1)
 a.mple <- mple.cppm("TypeA", a.sim$offspring$xy, init.pars, skip = 100)
 coef(a.mple)

### Type B Model
 # simulation
 pars <- c(mu1 = 10.0, mu2 = 40.0, nu = 30.0, sigma1 = 0.01, sigma2 = 0.03)
 b.sim <- sim.cppm("TypeB", pars, seed = 257)
 ## estimation
 init.pars <- c(mu1 = 20.0, mu2 = 30.0, nu = 30.0, sigma1 = 0.02, sigma2 = 0.02)
 b.mple <- mple.cppm("TypeB", b.sim$offspring$xy, init.pars)
 coef(b.mple)

### Type C Model
 # simulation
 pars <- c(mu1 = 5.0, mu2 = 9.0, nu1 = 30.0, nu2 = 150.0,
           sigma1 = 0.01, sigma2 = 0.05)
 c.sim <- sim.cppm("TypeC", pars, seed = 555)
 ## estimation
 init.pars <- c(mu1 = 10.0, mu2 = 10.0, nu1 = 30.0, nu2 = 120.0,
                sigma1 = 0.03, sigma2 = 0.03)
 c.mple <- mple.cppm("TypeC", c.sim$offspring$xy, init.pars)
 coef(c.mple)
```

```
## End(Not run)
```

palm.cppm *Non-parametric and Parametric Estimation for Palm Intensity*

Description

Compute the non-parametric and the parametric Palm intensity function of the Neyman-Scott cluster point process models and their extensions.

Usage

```
palm.cppm(mple, pars = NULL, delta = 0.001, uplimit = 0.3)

## S3 method for class 'Palm'
print(x, ...)
```

Arguments

mple	an object of class "mple".
pars	a named vector of the true parameters, if any.
delta	a width for the non-parametric Palm intensity function.
uplimit	upper limit in place of ∞ of the integral in the probability distribution function relative to the random distance between two descendant points within the same cluster. The uplimit is valid for "IP" and "TypeA".
x	an object of class "Palm".
...	ignored.

Value

An object of class "Palm" containing the following components:

r	the distance $r = j\Delta$, where $j = 1, 2, \ldots, [R/\Delta]$, $[\cdot]$ is the Gauss' symbol and $R = 1/2$ in the program.
np.palm	the corresponding values of the non-parametric Palm intensity function, which is normalized by the total intensity estimate (the mean number of points in W) of a given point pattern data.
norm.palm	the corresponding values of the normalized Palm intensity function, i.e., $\lambda_o(r)/\hat{\lambda}$, where $\lambda_o(r)$ is the Palm intensity and λ is an intensity of a cluster point process model. See 'Details' in mple.cppm.

There is another method plot.Palm for this class.

References

Tanaka, U., Ogata, Y. and Katsura, K. (2008) Simulation and estimation of the Neyman-Scott type spatial cluster models. *Computer Science Monographs* **34**, 1-44. The Institute of Statistical Mathematics, Tokyo. https://www.ism.ac.jp/editsec/csm/.

See Also

See sim.cppm and mple.cppm to simulate the Neyman-Scott cluster point process models and their extensions and to compute the MPLEs, respectively.

Examples

```
## Not run:
# The computation of MPLEs takes a long CPU time in the minimization procedure,
# especially for the Inverse-power type and the Type A models.

### Thomas Model
 #simulation
 pars <- c(mu = 50.0, nu = 30.0, sigma = 0.03)
 t.sim <- sim.cppm("Thomas", pars, seed = 117)
 ## estimation => Palm intensity
 init.pars <- c(mu = 40.0, nu = 40.0, sigma = 0.05)
 t.mple <- mple.cppm("Thomas", t.sim$offspring$xy, init.pars)
 t.palm <- palm.cppm(t.mple, pars)
 plot(t.palm)

### Inverse-Power Type Model
 # simulation
 pars <- c(mu = 50.0, nu = 30.0, p = 1.5, c = 0.005)
 ip.sim <- sim.cppm("IP", pars, seed = 353)
 ## estimation => Palm intensity
 init.pars <- c(mu = 55.0, nu = 35.0, p = 1.0, c = 0.01)
 ip.mple <- mple.cppm("IP", ip.sim$offspring$xy, init.pars, skip = 100)
 ip.palm <- palm.cppm(ip.mple, pars)
 plot(ip.palm)

### Type A Model
# simulation
 pars <- c(mu = 50.0, nu = 30.0, a = 0.3, sigma1 = 0.005, sigma2 = 0.1)
 a.sim <- sim.cppm("TypeA", pars, seed=575)
 ## estimation => Palm intensity
 init.pars <- c(mu=60.0, nu=40.0, a=0.5, sigma1=0.01, sigma2=0.1)
 a.mple <- mple.cppm("TypeA", a.sim$offspring$xy, init.pars, skip=100)
 a.palm <- palm.cppm(a.mple, pars)
 plot(a.palm)

### Type B Model
 # simulation
 pars <- c(mu1 = 10.0, mu2 = 40.0, nu = 30.0, sigma1 = 0.01, sigma2 = 0.03)
 b.sim <- sim.cppm("TypeB", pars, seed = 257)
 ## estimation => Palm intensity
 init.pars <- c(mu1 = 20.0, mu2 = 30.0, nu = 30.0, sigma1 = 0.02, sigma2 = 0.02)
 b.mple <- mple.cppm("TypeB", b.sim$offspring$xy, init.pars)
 b.palm <- palm.cppm(b.mple, pars)
 plot(b.palm)


### Type C Model
```

```
# simulation
pars <- c(mu1 = 5.0, mu2 = 9.0, nu1 = 30.0, nu2 = 150.0,
          sigma1 = 0.01, sigma2 = 0.05)
c.sim <- sim.cppm("TypeC", pars, seed = 555)
## estimation => Palm intensity
init.pars <- c(mu1 = 10.0, mu2 = 10.0, nu1 = 30.0, nu2 = 120.0,
               sigma1 = 0.03, sigma2 = 0.03)
c.mple <- mple.cppm("TypeC", c.sim$offspring$xy, init.pars)
c.palm <- palm.cppm(c.mple, pars)
plot(c.palm)

## End(Not run)
```

`plot.mple` *Show the Process for Optimizing Parameter Set*

Description

Plot method for object of class "mple" shows process for optimizing the normalized parameters depending on a given initial guess of each parameter.

Usage

```
## S3 method for class 'mple'
plot(x, ...)
```

Arguments

x	an object of class "mple" returned by mple.cppm.
...	further graphical parameters from par.

`plot.Palm` *Plot Non-Parametric and Parametric Normalized Palm Intensity*

Description

Plot method for objects of class "Palm".

Usage

```
## S3 method for class 'Palm'
plot(x, ..., log = "xy")
```

Arguments

x	an object of class "Palm", usually the result of a call to palm.cppm.
...	optional. At most 4 additional objects of class "Palm".
log	a character string indicating if logarithmic axes are to be used.

print.mple — *Print Process for Maximizing Log-Palm Likelihood Function*

Description

Print the process for minimizing the negative log-Palm likelihood function and/or the process for optimizing the normalized parameters depending on a given initial guess of each parameter by the simplex method.

Usage

```
## S3 method for class 'mple'
print(x, print.level = 0, ...)
```

Arguments

`x`	an object of class `"mple"` returned by `mple.cppm`.
`print.level`	We have the following processes: **0** output initial values and MPLE. **1** output the process for minimizing the negative log-Palm likelihood function, in addition. (`x$process1`) **2** output the process for optimizing the normalized parameters depending on a given initial guess of each parameter by the simplex method, in addition. (`x$process2`) **3** output both processes.
`...`	ignored.

sim.cppm — *Simulation for Neyman-Scott Cluster Point Process Models and Their Extensions*

Description

Simulation for the Thomas and Inverse-power type models, and the extended Thomas models of type A, B, and C.

Usage

```
sim.cppm(model = "Thomas", pars, seed = NULL)

## S3 method for class 'sim.cpp'
print(x, ...)
## S3 method for class 'sim.cpp'
plot(x, parents.distinct = FALSE, ...)
```

Arguments

`model`	a character string indicating each cluster point process model: `"Thomas"`, `"IP"`, `"TypeA"`, `"TypeB"`, and `"TypeC"`.
`pars`	a named vector giving the values of each parameter. See 'Details'.
`seed`	arbitrary positive integer to generate a sequence of uniform random numbers. The default seed is based on the current time.
`x`	an object of class `"sim.cpp"`.
`parents.distinct`	logical. If `TRUE`, simulated points are distinguished by two groups specified by parameters. (Only valid if `model = "TypeB"` or `"TypeC"`.)
`...`	further graphical parameters from `par` for `plot` or ignored for `print`.

Details

We consider the five cluster point process models: the Thomas and Inverse-power type models, and the extended Thomas models of type A, B, and C.

- `"Thomas"` (Thomas model)

  The parameters of the model are as follows:

  - `mu`: the intensity of parent points.
  - `nu`: the expectation of a random number of descendant points of each parent point.
  - `sigma`: the parameter set of the dispersal kernel.

  Let a random variable U be independently and uniformly distributed in [0,1].

  Consider

$$U = \int_0^r q_\sigma(t)dt = 1 - \exp\left(-\frac{r^2}{2\sigma^2}\right),$$

  where r is the random variable of the distance between each parent point and the descendant points associated with the given parent. The distance is distributed independently and identically according to the dispersal kernel.

  We have

$$r = \sigma\sqrt{-2\log(1-U)}.$$

  Let $(x_i^p, y_i^p), i = 1, 2, \ldots, I$, be a coordinate of each parent point where the integer I is generated from the Poisson random variable $Poisson(\mu)$ with mean μ from now on. Then, for each i, the number of offspring J_i is generated by the random variable $Poisson(\nu)$ with mean ν. Then, using series of different uniform random numbers $\{U\}$ for different i and j, each of the offspring coordinates $(x_j^i, y_j^i), j = 1, 2, \ldots, J_i$ is given by

$$x_j^i = x_i^p + r\cos(2\pi U),$$

$$y_j^i = y_i^p + r\sin(2\pi U),$$

  owing to the isotropy condition of the distribution.

Given a positive number ν and let a sequence of a random variable $\{U_k\}$ be independently and uniformly distributed in [0,1], the Poisson random number M is the smallest integer such that

$$\sum_{k=1}^{M+1} -\log U_k > \nu,$$

where $\log$ represents natural logarithm.

- `"IP"` (Inverse-power type model)
  The parameters of the model are as follows:
  - `mu`: the intensity of parent points.
  - `nu`: the expectation of a random number of descendant points of each parent point.
  - `p`, `c`: the set of parameters of the dispersal kernel, where `p` > 1 and `c` > 0.

  Let U be as above.
  For all $r \geq 0$,

$$Q_{p,c}(r) := \int_0^r q_{p,c}(t)dt$$
$$= c^{p-1}(p-1)\frac{(r+c)^{1-p} - c^{1-p}}{1-p}$$
$$= 1 - c^{p-1}(r+c)^{1-p}.$$

  Here, we put $Q_{p,c}(r) = U$. From this, we have

$$r = c\{(1-U)^{1/(1-p)} - 1\}.$$

  The parent points and their descendant points are generated the same as the Thomas model.

- `"TypeA"` (Type A model)
  The parameters of the model are as follows:
  - `mu`: the intensity of parent points.
  - `nu`: the expectation of a random number of descendant points of each parent point.
  - `a`, `sigma1`, `sigma2`: the set of parameters of the dispersal kernel, where where `a` is a mixture ratio parameter with 0 < `a` < 1.

  Let each random variable $U_k, k = 1, 2$, be independently and uniformly distributed in [0,1].
  Then r satisfies as follows:

$$r = \sigma_1\sqrt{-2\log(1-U_1)}, \quad U_2 \leq a,$$
$$r = \sigma_2\sqrt{-2\log(1-U_1)}, \quad \text{otherwise}.$$

  The parent points and their descendant points are generated the same as the Thomas model.

- `"TypeB"` (Type B model)
  The TypeB is a superposed Thomas model. The parameters of the model are as follows:
  - `mu1`, `mu2`: the corresponding intensity of parent points of each Thomas model.

- nu: the expectation of a random number of descendant points of each parent point.
- sigma1, sigma2: the corresponding set of parameters of the dispersal kernel of each Thomas model.

Consider the two types of the Thomas model with parameters (μ_1, ν, σ_1) and (μ_2, ν, σ_2). Parents' configuration and numbers of the descendant cluster sizes are generated by the two types of uniformly distributed parents (x_i^k, y_i^k) with $i = 1, 2, \ldots, Poisson(\mu_k)$ for $k = 1, 2$, respectively.

Then, using series of different uniform random numbers $\{U\}$ for different i and j, each of the descendant coordinates $(x_j^{k,i}, y_j^{k,i})$ of the parents (x_i^k, y_i^k), $k = 1, 2$, $j = 1, 2, \ldots, Poisson(\nu)$, is given by

$$x_j^{k,i} = x_i^k + r_k \cos(2\pi U),$$
$$y_j^{k,i} = y_i^k + r_k \sin(2\pi U),$$

where

$$r_k = \sigma_k \sqrt{-2\log(1 - U_k)}, \quad k = 1, 2,$$

with different random numbers $\{U_k, U\}$ for different k, i, and j.

- "TypeC" (Type C model)

  The TypeC is a superposed Thomas model. The parameters of the model are as follows:

  - mu1, mu2: the corresponding intensity of parent points of each Thomas model.
  - nu1, nu2: the corresponding expectation of a random number of descendant points of each Thomas model.
  - sigma1, sigma2: the corresponding set of parameters of the dispersal kernel of each Thomas model.

  The parent points and their descendant points are generated the same as the Type B model.

Value

sim.cppm returns an object of class "sim.cpp" containing the following components which has print and plot methods.

parents — a list containing two components named "n" and "xy", which are the number and the matrix of (x,y) coordinates of simulated parent points, respectively. For "TypeB", xy [1:n[1], 1:2] and the remainder are generated from (mu1, nu, sigma1) and (mu2, nu, sigma2), respectively. For "TypeC", xy[1:n[1], 1:2] and the remainder are generated from (mu1, nu1, sigma1) and (mu2, nu2, sigma2), respectively.

offspring — a list containing two components named "n" and "xy", which are the number and the matrix of (x,y) coordinates of simulated descendant points, respectively. For "TypeB", xy [1:n[1], 1:2] and the remainder are generated from (mu1, nu, sigma1) and (mu2, nu, sigma2), respectively. For "TypeC", xy[1:n[1], 1:2] and the remainder are generated from (mu1, nu1, sigma1) and (mu2, nu2, sigma2), respectively.

References

Tanaka, U., Ogata, Y. and Katsura, K. (2008) Simulation and estimation of the Neyman-Scott type spatial cluster models. *Computer Science Monographs* **34**, 1-44. The Institute of Statistical Mathematics, Tokyo. https://www.ism.ac.jp/editsec/csm/.

Examples

```
## Thomas Model
pars <- c(mu = 50.0, nu = 30.0, sigma = 0.03)
t.sim <- sim.cppm("Thomas", pars, seed = 117)
t.sim
plot(t.sim)

## Inverse-Power Type Model
pars <- c(mu = 50.0, nu = 30.0, p = 1.5, c = 0.005)
ip.sim <- sim.cppm("IP", pars, seed = 353)
ip.sim
plot(ip.sim)

## Type A Model
pars <- c(mu = 50.0, nu = 30.0, a = 0.3, sigma1 = 0.005, sigma2 = 0.1)
a.sim <- sim.cppm("TypeA", pars, seed = 575)
a.sim
plot(a.sim)

## Type B Model
pars <- c(mu1 = 10.0, mu2 = 40.0, nu = 30.0, sigma1 = 0.01, sigma2 = 0.03)
b.sim <- sim.cppm("TypeB", pars, seed = 257)
b.sim
plot(b.sim, parents.distinct = TRUE)

## Type C Model
pars <- c(mu1 = 5.0, mu2 = 9.0, nu1 = 30.0, nu2 = 150.0,
               sigma1 = 0.01, sigma2 = 0.05)
c.sim <- sim.cppm("TypeC", pars, seed = 555)
c.sim
plot(c.sim, parents.distinct = FALSE)
```

Index