

Package ‘MoNAn’

August 29, 2023

Type Package

Title Mobility Network Analysis

Version 0.1.2

Date 2023-08-15

Description Implements the method to analyse weighted mobility networks or distribution networks as outlined in:

Block, P., Stadtfeld, C., & Robins, G. (2022) <[doi:10.1016/j.socnet.2021.08.003](https://doi.org/10.1016/j.socnet.2021.08.003)>.

The purpose of the model is to analyse the structure of mobility, incorporating exogenous predictors pertaining to individuals and locations known from classical mobility analyses, as well as modelling emergent mobility patterns akin to structural patterns known from the statistical analysis of social networks.

Depends R (>= 4.2)

License GPL (>= 3)

Encoding UTF-8

LazyData true

Imports snowfall (>= 1.84-6.2), methods

RoxygenNote 7.2.3

Suggests knitr, rmarkdown

NeedsCompilation no

Author Per Block [cre, aut, cph] (<<https://orcid.org/0000-0002-7583-2392>>),
Christoph Stadtfeld [aut] (<<https://orcid.org/0000-0002-2704-2134>>),
Nico Keiser [aut]

Maintainer Per Block <block@soziologie.uzh.ch>

Repository CRAN

Date/Publication 2023-08-29 17:00:08 UTC

R topics documented:

autoCorrelationTest	2
createAlgorithm	3

createEdgelist	5
createEffectsObject	6
createNetwork	7
createNodeSet	8
createNodeVariable	9
createProcessState	10
createWeightedCache	11
estimateMobilityNetwork	12
extractTraces	14
getMultinomialStatistics	15
gofMobilityNetwork	16
mobilityData	17
myOutcomeObjects	18
scoreTest	18
simulateMobilityNetworks	19

Index	22
--------------	-----------

autoCorrelationTest	<i>autoCorrelationTest</i>
---------------------	----------------------------

Description

The `autoCorrelationTest` indicates the degree to which the values of the dependent variable of consecutive draws from the chain in phase 3 are correlated. Here lower values are better. Values above 0.5 are very problematic and indicate that a higher thinning is needed.

Usage

```
autoCorrelationTest(ans)
```

Arguments

ans	An object of class "result.monan" resulting from an estimation with the function <code>estimateMobilityNetwork()</code> .
-----	---

Value

A number indicating the auto-correlation.

Examples

```
# regression diagnostics
autoCorrelationTest(myResDN)
```

createAlgorithm *createAlgorithm*

Description

Specifies the algorithm used in the estimation based on characteristics of the state and the effects.

Usage

```
createAlgorithm(
  state,
  effects,
  multinomialProposal = FALSE,
  burnInN1 = NULL,
  thinningN1 = NULL,
  iterationsN1 = NULL,
  gainN1 = 0.1,
  burnInN2 = NULL,
  thinningN2 = NULL,
  initialIterationsN2 = 50,
  nsubN2 = 4,
  initGain = 0.2,
  burnInN3 = NULL,
  thinningN3 = NULL,
  iterationsN3 = 500,
  allowLoops = NULL
)
```

Arguments

state	An object of class "processState.monan" that contains all relevant information about the outcome in the form of an edgelist, the nodesets, and covariates.
effects	An object of class "effectsList.monan" that specifies the model.
multinomialProposal	How should the next possible outcome in the simulation chains be sampled? If TRUE, fewer simulation steps are needed, but each simulation step takes considerably longer. Defaults to FALSE.
burnInN1	The number of simulation steps before the first draw in Phase 1. A recommended value is at least $n_Individuals * n_locations$ if <code>multinomialProposal = FALSE</code> , and at least $n_Individuals$ if <code>multinomialProposal = TRUE</code> which is set as default.
thinningN1	The number of simulation steps between two draws in Phase 1. A recommended value is at least $0.5 * n_Individuals * n_locations$ if <code>multinomialProposal = FALSE</code> , and at least $n_Individuals$ if <code>multinomialProposal = TRUE</code> which is set as default.

<code>iterationsN1</code>	The number of draws taken in Phase 1. A recommended value is at least $4 * n_effects$ which is set as default. If the value is too low, there will be an error in Phase 1.
<code>gainN1</code>	The size of the updating step after Phase 1. A conservative value is 0, values higher than 0.25 are courageous. Defaults to 0.1.
<code>burnInN2</code>	The number of simulation steps before the first draw in Phase 1. A recommended value is at least $n_Individuals * n_locations$ if <code>multinomialProposal = FALSE</code> , and at least $n_Individuals$ if <code>multinomialProposal = TRUE</code> which is set as default.
<code>thinningN2</code>	The number of simulation steps between two draws in Phase 2. A recommended value is at least $0.5 * n_Individuals * n_locations$ if <code>multinomialProposal = FALSE</code> , and at least $n_Individuals$ if <code>multinomialProposal = TRUE</code> which is set as default.
<code>initialIterationsN2</code>	The number of draws taken in subphase 1 of Phase 2. For first estimations, a recommended value is around 50 (default to 50). Note that in later subphases, the number of iterations increases. If this is a further estimation to improve convergence, higher values (100+) are recommended.
<code>nsubN2</code>	Number of subphases in Phase 2. In case this is the first estimation, 4 subphases are recommended and set as default. If convergence in a previous estimation was close, then 1-2 subphases should be enough.
<code>initGain</code>	The magnitude of parameter updates in the first subphase of Phase 2. Values of around 0.2 (default) are recommended.
<code>burnInN3</code>	The number of simulation steps before the first draw in Phase 3. A recommended value is at least $3 * n_Individuals * n_locations$ if <code>multinomialProposal = FALSE</code> , and at least $3 * n_Individuals$ if <code>multinomialProposal = TRUE</code> which is set as default.
<code>thinningN3</code>	The number of simulation steps between two draws in Phase 3. A recommended value is at least $n_Individuals * n_locations$ if <code>multinomialProposal = FALSE</code> , and at least $2 * n_Individuals$ if <code>multinomialProposal = TRUE</code> which is set as default. In case this value is too low, the outcome might erroneously indicate a lack of convergence.
<code>iterationsN3</code>	Number of draws in Phase 3. Recommended are at the very least 500 (default). In case this value is too low, the outcome might erroneously indicate a lack of convergence.
<code>allowLoops</code>	Logical: can individuals/resources stay in their origin?

Value

An object of class "algorithm.monan".

See Also

[createProcessState\(\)](#), [createEffectsObject\(\)](#), [estimateMobilityNetwork\(\)](#)

Examples

```
# define algorithm based on state and effects characteristics
myAlg <- createAlgorithm(myState, myEffects, multinomialProposal = FALSE)
```

createEdgelist	<i>createEdgelist</i>
----------------	-----------------------

Description

Creates an edgelist object, which is the standard format of the outcome to be modelled by MoNAn.

Usage

```
createEdgelist(e1, nodeSet = c("location", "location", "individuals"))
```

Arguments

e1	An edgelist in the form of a matrix with two columns and N rows. The first column indicates the origin of a person/resource, the second row the destination. Each row represents one observation.
nodeSet	The nodesets of the edgelists. This is a vector with three entries referencing the names of the nodesets of locations and individuals of the form c(location, location, individuals).

Value

An object of class "edgelist.monan".

See Also

[createProcessState\(\)](#)

Examples

```
# create an object of class edgelist.monan
transfers <- createEdgelist(mobilityEdgelist, c("organisations", "organisations", "people"))
```

createEffectsObject *createEffectsObject*

Description

Specifies the model with endogenous and exogenous predictors. The predictors in the model are called “effects”.

Usage

```
createEffectsObject(effectInit, checkProcessState = NULL)
```

Arguments

effectInit A list of "effects", where each effect to be included is specified as a further list that contains the effect name and the additional parameters it needs. Effects without further parameters only contain the effect name (e.g., loops).

checkProcessState
 For internal use only.

Value

An object of class "effectsList.monan".

Examples

```
# create an effects object
myEffects <- createEffectsObject(
  list(
    list("loops"),
    list("reciprocity_min"),
    list("dyadic_covariate", attribute.index = "sameRegion"),
    list("alter_covariate", attribute.index = "size"),
    list("resource_covar_to_node_covar",
        attribute.index = "region",
        resource.attribute.index = "sex"
    ),
    list("loops_resource_covar", resource.attribute.index = "sex")
  )
)
```

createNetwork	<i>createNetwork</i>
---------------	----------------------

Description

Defines a network between locations, generally to be used as a predictor in the model. NOTE: The outcome variable of the model is not defined as a network, but as an edgelist!

Usage

```
createNetwork(  
  m,  
  isSymmetric = FALSE,  
  isBipartite = FALSE,  
  nodeSet = c("actors", "actors")  
)
```

Arguments

<code>m</code>	A square matrix containing the network data.
<code>isSymmetric</code>	Currently not in use.
<code>isBipartite</code>	Currently not in use.
<code>nodeSet</code>	Which nodeset are the nodes of the network. Usually this will be the locations in the data.

Value

An object of class "network.monan".

See Also

[createProcessState\(\)](#), [createEdgelist\(\)](#)

Examples

```
# create an object of class network.monan  
sameRegion <- outer(orgRegion, orgRegion, "==") * 1  
sameRegion <- createNetwork(sameRegion, nodeSet = c("organisations", "organisations"))
```

createNodeSet	<i>createNodeSet</i>
---------------	----------------------

Description

Determines and names the nodesets of individuals and locations that make up the mobility network.

Usage

```
createNodeSet(x = NULL, isPresent = NULL, considerWhenSampling = NULL)
```

Arguments

x	Either a single number indicating how many items are in this nodeset or a vector from 1:n_items.
isPresent	Currently not in use.
considerWhenSampling	A boolean/logical vector of the length of the nodeset. Only in use in special cases. If the nodeset indicates a location, considerWhenSampling indicates whether the location is a possible destination, or is only an origin (e.g. a training facility). Entries in the vector of locations that cannot be a destination are FALSE. If the nodeset indicates mobile individuals, considerWhenSampling indicates whether their mobility should be modelled or whether it is structurally determined, that is, their mobility is exogenously defined and does not follow the same logic as the mobility of everybody else.

Value

An object of class "nodeSet.monan".

See Also

[createProcessState\(\)](#)

Examples

```
# create an object of class nodeSet.monan
people <- createNodeSet(1:nrow(mobilityEdgelist))
organisations <- createNodeSet(length(orgRegion))
```

createNodeVariable *createNodeVariable*

Description

Assigns a covariate to one nodeset, i.e., an exogenous characteristic of mobile individuals/resources or locations.

Usage

```
createNodeVariable(  
  values,  
  range = NULL,  
  nodeSet = "actors",  
  addSame = FALSE,  
  addSim = FALSE  
)
```

Arguments

values	A vector assigning the covariate value to each element of the nodeset.
range	The range of possible values, currently not in use.
nodeSet	The nodeset to which the covariate applies.
addSame	Will the variable be used to model categorical homophily (e.g., with the same_covariate effect)? In this case, addSame needs to be set to TRUE.
addSim	Will the variable be used to model continuous homophily (e.g., with the sim_covariate effect)? In this case, addSim needs to be set to TRUE.

Value

An object of class "nodeVar.monan".

See Also

[createProcessState\(\)](#)

Examples

```
# create an object of class nodeVar.monan  
region <- createNodeVariable(orgRegion, nodeSet = "organisations")  
size <- createNodeVariable(orgSize, nodeSet = "organisations", addSim = TRUE)  
sex <- createNodeVariable(indSex, nodeSet = "people")
```

createProcessState *createProcessState*

Description

Creates the "Process state", i.e., a MoNAn object that stores all information about the data that will be used in the estimation. This includes the outcome variable (edgelist), the nodesets, and all covariates.

Usage

```
createProcessState(elements, dependentVariable)
```

Arguments

elements A named list of the outcome variable (edgelist), the nodesets, and all covariates that contain the information about the data that will be used in the estimation.

dependentVariable The name of the outcome variable (edgelist) as specified under "elements". This indicates what outcome the researcher is interested in.

Value

An object of class "processState.monan".

See Also

[createEdgelist\(\)](#), [createNodeSet\(\)](#), [createNodeVariable\(\)](#), [createNetwork\(\)](#)

Examples

```
# Create a process state out of the mobility data objects:
# create objects (which are later combined to the process state)
transfers <- createEdgelist(mobilityEdgelist,
  nodeSet = c("organisations", "organisations", "people")
)
people <- createNodeSet(1:nrow(mobilityEdgelist))
organisations <- createNodeSet(1:length(orgRegion))
sameRegion <- outer(orgRegion, orgRegion, "==") * 1
sameRegion <- createNetwork(sameRegion,
  nodeSet = c("organisations", "organisations")
)
region <- createNodeVariable(orgRegion, nodeSet = "organisations")
size <- createNodeVariable(orgSize, nodeSet = "organisations", addSim = TRUE)
sex <- createNodeVariable(indSex, nodeSet = "people")

# combine created objects to the process state
myState <- createProcessState(list(
  transfers = transfers,
```

```
people = people,  
organisations = organisations,  
sameRegion = sameRegion,  
region = region,  
size = size,  
sex = sex),  
dependentVariable = "transfers")
```

`createWeightedCache` *createWeightedCache*

Description

Creates a necessary internal object used in simulating the chains in the simulation and estimation of the model. In case variables of the individuals in the data are included in the state, they need to be explicitly mentioned in the creation of the cache under “resourceCovariates”.

Usage

```
createWeightedCache(processState, resourceCovariates = NULL)
```

Arguments

`processState` The process state that provides the data basis for creating the cache.
`resourceCovariates`
 A vector of resource covariates that will be used in the model specification.

Value

A cache object provided as a list.

See Also

[createProcessState\(\)](#)

Examples

```
# create cache object  
myCache <- createWeightedCache(myState, resourceCovariates = c("sex"))
```

```
estimateMobilityNetwork  
    estimateMobilityNetwork
```

Description

The core function of the package in which the model for the analysis of mobility tables is estimated.

Usage

```
estimateMobilityNetwork(  
  state,  
  cache,  
  effects,  
  algorithm,  
  initialParameters = NULL,  
  prevAns = NULL,  
  parallel = FALSE,  
  cpus = 1,  
  verbose = FALSE,  
  returnDeps = FALSE,  
  fish = FALSE  
)  
  
estimateDistributionNetwork(  
  state,  
  cache,  
  effects,  
  algorithm,  
  initialParameters = NULL,  
  prevAns = NULL,  
  parallel = FALSE,  
  cpus = 1,  
  verbose = FALSE,  
  returnDeps = FALSE,  
  fish = FALSE  
)  
  
## S3 method for class 'result.monan'  
print(x, covMat = FALSE, ...)
```

Arguments

state	An object of class "processState.monan" which contains all relevant information about the outcome in the form of an edgelist, the nodesets, and covariates.
cache	A cache object created from the same state object that is used in the estimation.

effects	An object of class "effectsList.monan" that specifies the model.
algorithm	An object of class "algorithm.monan" which specifies the algorithm used in the estimation.
initialParameters	Starting values for the parameters. Using starting values, e.g., from a multinomial logit model (see getMultinomialStatistics()) increases the chances of model convergence in the first run of the estimation considerably.
prevAns	If a previous estimation did not yield satisfactory convergence, the outcome object of that estimation should be specified here to provide new starting values for the estimation.
parallel	Logical: computation on multiple cores?
cpus	Number of cores for computation in case parallel = TRUE.
verbose	Logical: display information about estimation progress in the console?
returnDeps	Logical: should the simulated values of Phase 3 be stored and returned? This is necessary to run GoF tests. Note that this might result in very large objects.
fish	Logical: display a fish?
x	An object of class "result.monan".
covMat	Logical: indicating whether the covariance matrix should be printed.
...	For internal use only.

Value

The function `estimateMobilityNetwork` returns an object of class "result.monan" that contains the estimates, standard errors, and convergence statistics. Furthermore, the covariance matrix used to calculate the standard errors is included, which also shows collinearity between effects. In case `returnDeps = TRUE`, the simulations of Phase 3 are included, too.

The function `print.result.monan` prints the results from a monan estimation with three columns indicating the estimate, the standard error, and the convergence statistic.

See Also

[createProcessState\(\)](#), [createWeightedCache\(\)](#), [createEffectsObject\(\)](#), [createAlgorithm\(\)](#)

Examples

```
# estimate mobility network model

myResDN <- estimateMobilityNetwork(myState, myCache, myEffects, myAlg,
  initialParameters = NULL,
  # in case a pseudo-likelihood estimation was run, replace with
  # initialParameters = initEst,
  parallel = TRUE, cpus = 4,
  verbose = TRUE,
  returnDeps = TRUE,
  fish = FALSE
)
```

```
# check convergence
max(abs(myResDN$convergenceStatistics))

# view results
myResDN

myResDN
```

extractTraces

extractTraces

Description

This function shows the values of simulated statistics in Phase 3 for subsequent draws from the chain. Ideally, the plots show points randomly scattered around the red line, which indicates the statistics in the data.

Usage

```
extractTraces(ans, effects)

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

Arguments

ans	An object of class "result.monan" resulting from an estimation with the function estimateMobilityNetwork() .
effects	An object of class "effectsList.monan" used in the estimation.
x	An object of class "traces.monan".
...	Additional plotting parameters, use not recommended.

Value

The function `extractTraces` returns a list that includes (1) the observed statistics for all effects, (2) the distribution of statistics for all simulations and (3) effect names. It is recommended to use the plotting function to inspect the traces.

The function `plot.traces.monan` shows a scatter plot of the statistics of simulated networks from phase three of the estimation.

See Also

[createEffectsObject\(\)](#)

Examples

```
# regression diagnostics
traces <- extractTraces(myResDN, myEffects)

plot(traces)
```

```
getMultinomialStatistics
      getMultinomialStatistics
```

Description

One updating step in simulating the mobility network model can be expressed as a multinomial logit model. Extracting the statistics for such a model allows a straight-forward estimation of a multinomial logit model to get initial estimates for the full mobility model, which increases the chances of model convergence in the first run of the estimation considerably.

Usage

```
getMultinomialStatistics(state, cache, effects)
```

Arguments

state	An object of class "processState.monan" that stores all information to be used in the model.
cache	A cache object that contains intermediate information used for estimation.
effects	An object of class "effectsList.monan" for which the statistics of a multinomial model should be calculated.

Value

A data frame with $N * M$ rows (N = mobile individuals, M = number of locations) that specifies for each observation the statistics associated with moving to this location.

See Also

[createProcessState\(\)](#), [createWeightedCache\(\)](#), [createEffectsObject\(\)](#)

Examples

```
myStatisticsFrame <- getMultinomialStatistics(myState, myCache, myEffects)
```

gofMobilityNetwork *gofMobilityNetwork*

Description

Akin to ERGMs, goodness of fit testing is available to see whether auxiliary statistics are well captured by the model. The logic behind gof testing for network models is outlined in Hunter et al. (2008) and Lospinoso and Snijders (2019).

Usage

```
gofMobilityNetwork(ans, simulations, gofFunction, lvls = NULL)

gofDistributionNetwork(ans, simulations, gofFunction, lvls = NULL)

## S3 method for class 'gof.stats.monan'
plot(x, lvls, ...)
```

Arguments

ans	An object of class "result.monan" resulting from an estimation with the function estimateMobilityNetwork() .
simulations	The simulated outcomes with which the observed statistics are compared. Usually, they are stored in the ans\$deps, in case deps = TRUE was specified in the estimation.
gofFunction	A gof function that specifies which auxiliary outcome should be used, e.g., "getIndegree" or "getTieWeights".
lvls	The values for which the gofFunction should be calculated/plotted.
x	An object of class "gof.stats.monan".
...	Additional plotting parameters, use discouraged.

Value

The function `gofMobilityNetwork` returns a list containing (1) the observed values of the auxiliary statistics and (2) a list of the simulated values of the auxiliary statistics.

The function `plot.gof.stats.monan` returns violin plots of the gof tests with observed values superimposed in red.

References

Hunter, D. R., Goodreau, S. M., & Handcock, M. S. (2008). Goodness of fit of social network models. *Journal of the american statistical association*, 103(481), 248-258.

Lospinoso, J., & Snijders, T. A. (2019). Goodness of fit for stochastic actor-oriented models. *Methodological Innovations*, 12(3).

See Also

[getIndegree\(\)](#), [getTieWeights\(\)](#)

Examples

```
# goodness of fit
myGofIndegree <- gofMobilityNetwork(ans = myResDN,
                                   simulations = myResDN$deps,
                                   gofFunction = getIndegree,
                                   lvls = 1:100)

myGofTieWeight <- gofMobilityNetwork(ans = myResDN,
                                    simulations = myResDN$deps,
                                    gofFunction = getTieWeights,
                                    lvls = 1:30)

plot(myGofIndegree, lvls = 20:70)
plot(myGofTieWeight, lvls = 1:15)
```

 mobilityData

Example Data for the MoNAn Package

Description

These are example data for the MoNAn package and can be used to estimate a mobility network. The raw example data is synthetic (i.e., made up). This fictitious example contains 17 organisations representing a labour market that are located in two regions (north and south). 742 workers are employed in these organisations at two time-points. Some are mobile while others work in the same organisation at both time-points. The following objects are provided for this purpose:

`mobilityEdgelist` The data frame indicates the origin at time 1 (first column) and the destination at time 2 (second column) for each of the 742 individuals between the 17 organisations. Note that some workers stay in their organisation, i.e. their origin equals their destination.

`orgRegion` Categorical characteristic describing whether the organisation is located on the northern (1) or southern (0) region.

`orgSize` Continuous measure representing the size of each organisation based on assets and revenue.

`indSex` Individual-level characteristics representing sex.

Format

`mobilityEdgelist` A data frame with 743 rows and 2 columns.

`orgRegion` An object with 17 values.

`orgSize` An object with 17 values.

`indSex` An object with 742 values.

myOutcomeObjects	<i>Exemplary Outcome Objects for the MoNAn Package</i>
------------------	--

Description

These are exemplary outcome objects for the MoNAn package and can be used in order not to run all precedent functions and thus save time. The following products are provided:

Format

myState An object of class "processState.monan" created by the function [createProcessState\(\)](#).
 myCache A "list" created by the function [createWeightedCache\(\)](#).
 myEffects An object of class "effectsList.monan" created by the function [createEffectsObject\(\)](#).
 myAlg An object of class "algorithm.monan" created by the function [createAlgorithm\(\)](#).
 myResDN An object of class "result.monan" created by the function [estimateMobilityNetwork\(\)](#).
 mySimDN An object of class "sims.monan" created by the function [simulateMobilityNetworks\(\)](#).

scoreTest	<i>scoreTest</i>
-----------	------------------

Description

Based on an estimated model, a score-type test is available that shows whether statistics representing non-included effects are well represented. If this is not the case, it is likely that including them will result in significant estimates.

Usage

```
scoreTest(ans, effects)

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

Arguments

ans	An object of class "result.monan" resulting from an estimation with the function estimateMobilityNetwork() .
effects	An object of class "effectsList.monan" in which the non included effects that should be tested are specified.
x	An object of class "scoretest.monan".
...	For internal use only.

Value

The function `scoreTest` returns basic values to calculate parametric and non-parametric p-values for each tested effect.

The function `print.scoretest.monan` shows parametric and non-parametric p-values for each tested effect.

See Also

[createEffectsObject\(\)](#)

Examples

```
# test whether other effects should be included
myEffects2 <- createEffectsObject(
  list(
    list("loops"),
    list("reciprocity_min"),
    list("dyadic_covariate", attribute.index = "sameRegion"),
    list("alter_covariate", attribute.index = "size"),
    list("resource_covar_to_node_covar", attribute.index = "region",
        resource.attribute.index = "sex"),
    list("loops_resource_covar", resource.attribute.index = "sex"),
    list("transitivity_min")
  )
)

test_ME.2 <- scoreTest(myResDN, myEffects2)

test_ME.2
```

simulateMobilityNetworks

simulateMobilityNetworks

Description

Simulates mobility networks for given data, effects, and parameters. This function is mainly interesting to explore the behavior of the model or to do counter-factual simulations.

Usage

```
simulateMobilityNetworks(
  state,
  cache,
  effects,
```

```

    parameters,
    allowLoops,
    burnin,
    thinning,
    nSimulations
)

simulateDistributionNetworks(
  state,
  cache,
  effects,
  parameters,
  allowLoops,
  burnin,
  thinning,
  nSimulations
)

```

Arguments

state	An object of class "processState.monan" that contains all relevant information about nodesets, and covariates. Further, an edgelist of the dependent variable needs to be specified with the initial mobility network as starting value for the simulation. For a large enough burn-in, any initial mobility network is allowed.
cache	A cache object created from the same state object that is used in the simulation.
effects	An object of class "effectsList.monan" that specifies the model.
parameters	The parameters associated with the effects that shall be used in the simulations.
allowLoops	Logical: can individuals/resources stay in their origin?
burnin	The number of simulation steps that are taken before the first draw of a network is taken. A number too small will mean the first draw is influenced by the initially specified network. A recommended value for the lower bound is $3 * n_Individuals * n_locations$.
thinning	The number of simulation steps that are taken between two draws of a network. A recommended value for the lower bound is $n_Individuals * n_locations$.
nSimulations	The number of mobility networks to be simulated.

Value

An object of class "sims.monan" with nSimulations entries, where each entry contains a further list with the state and the cache of the current simulation stored.

Examples

```

# simulate a mobility network
# note that thinning and burn-in values are for this example only
# in real cases, choose values approx. times 10

```

```
mySimDN <- simulateMobilityNetworks(  
  myState,  
  myCache,  
  myEffects,  
  parameters = c(2, 1, 1.5, 0.1, -1, -0.5),  
  allowLoops = TRUE,  
  burnin = 4500,  
  thinning = 1500,  
  nSimulations = 10  
)  
  
mySimDN[[1]]
```

Index

- * **datasets**
 - mobilityData, 17
 - myOutcomeObjects, 18
- autoCorrelationTest, 2
- createAlgorithm, 3
- createAlgorithm(), 13, 18
- createEdgelist, 5
- createEdgelist(), 7, 10
- createEffectsObject, 6
- createEffectsObject(), 4, 13–15, 18, 19
- createNetwork, 7
- createNetwork(), 10
- createNodeSet, 8
- createNodeSet(), 10
- createNodeVariable, 9
- createNodeVariable(), 10
- createProcessState, 10
- createProcessState(), 4, 5, 7–9, 11, 13, 15, 18
- createWeightedCache, 11
- createWeightedCache(), 13, 15, 18
- estimateDistributionNetwork
 - (estimateMobilityNetwork), 12
- estimateMobilityNetwork, 12
- estimateMobilityNetwork(), 2, 4, 14, 16, 18
- extractTraces, 14
- getIndegree(), 17
- getMultinomialStatistics, 15
- getMultinomialStatistics(), 13
- getTieWeights(), 17
- gofDistributionNetwork
 - (gofMobilityNetwork), 16
- gofMobilityNetwork, 16
- indSex (mobilityData), 17
- mobilityData, 17
- mobilityEdgelist (mobilityData), 17
- myAlg (myOutcomeObjects), 18
- myCache (myOutcomeObjects), 18
- myEffects (myOutcomeObjects), 18
- myOutcomeObjects, 18
- myResDN (myOutcomeObjects), 18
- mySimDN (myOutcomeObjects), 18
- myState (myOutcomeObjects), 18
- orgRegion (mobilityData), 17
- orgSize (mobilityData), 17
- plot.gof.stats.monan
 - (gofMobilityNetwork), 16
- plot.traces.monan (extractTraces), 14
- print.result.monan
 - (estimateMobilityNetwork), 12
- print.scoretest.monan (scoreTest), 18
- scoreTest, 18
- simulateDistributionNetworks
 - (simulateMobilityNetworks), 19
- simulateMobilityNetworks, 19
- simulateMobilityNetworks(), 18