Package 'RSiena'

May 13, 2018

Type Package

Title Siena - Simulation Investigation for Empirical Network Analysis

Version 1.2-12

Date 2018-05-12

Author Ruth Ripley, Krists Boitmanis, Tom A.B. Snijders, Felix Schoenenberger

Depends R (>= 2.15.0), utils

Imports Matrix, tcltk, lattice, parallel, MASS, methods

Suggests xtable, network, tools, codetools

SystemRequirements GNU make, tcl/tk 8.5, Tktable

Maintainer Tom A.B. Snijders <tom.snijders@nuffield.ox.ac.uk>

Description The main purpose of this package is to perform simulation-based estimation of stochastic actor-oriented models for longitudinal network data collected as panel data. Dependent variables can be single or multivariate networks, which can be directed, non-directed, or two-mode. There are also functions for testing parameters and checking goodness of fit. An overview of these models is given in Tom A.B. Snijders (2017), Stochastic Actor-Oriented Models for Network Dynamics, Annual Review of Statistics and Its Application, 4, 343-363 <doi: 10.1146/annurev-statistics-060116-054035>.

License GPL (>= 3)

LazyLoad yes

LazyData yes

Biarch yes

NeedsCompilation yes

BuildResaveData no

URL http://www.stats.ox.ac.uk/~snijders/siena

Repository CRAN

Date/Publication 2018-05-13 14:58:05 UTC

R topics documented:

RSiena-package *Simulation Investigation for Empirical Network Analysis*

Description

Fits statistical models to longitudinal sets of networks, and to longitudinal sets of networks and behavioral variables. Not only one-mode networks but also two-mode networks and multivariate networks are allowed. The models are stochastic actor-oriented models.

Package "RSienaTest" is the development version, and is distributed through R-Forge, see http://r-forge.r-project.org/R/?group_id=461. Package "RSiena" is the official release.

Details

The main flow of operations of this package is as follows.

Data objects can be created from matrices and vectors using sienaDependent, coCovar, varCovar, coDyadCovar, etc., and finally sienaDataCreate.

Effects are selected using an sienaEffects object, which can be created using getEffects and may be further specified by includeEffects, setEffect, and includeInteraction.

Control of the estimation algorithm requires a sienaAlgorithm object that defines the settings (parameters) of the algorithm, and which can be created by sienaAlgorithmCreate.

Function siena07 is used to fit a model.

A general introduction to the method is available in the tutorial paper Snijders, van de Bunt, and Steglich (2010). Next to the help pages, more detailed help is available in the manual (see below) and a lot of information is at the website (also see below).

Package:	RSiena
Type:	Package
Version:	1.2-12
Date:	2018-05-12
Depends:	R (>= 3.0.0)
Imports:	Matrix
Suggests:	tcltk, network, codetools, lattice, MASS, parallel, xtable, tools, utils
SystemRequirements:	GNU make, tcl/tk 8.5, Tktable
License:	GPL-2
LazyData:	yes
NeedsCompilation:	yes
BuildResaveData:	no

Author(s)

Ruth Ripley, Krists Boitmanis, Tom Snijders, Felix Schoenenberger. Contributions by Josh Lospinoso, Charlotte Greenan, Christian Steglich, Johan Koskinen, Mark Ortmann, Nynke Niezink, Natalie Indlekofer, Christoph Stadtfeld, and Robert Hellpap.

Maintainer: Tom A.B. Snijders <tom.snijders@nuffield.ox.ac.uk>

References

- Schweinberger, Michael, and Snijders, Tom A.B. (2007). Markov models for digraph panel data: Monte Carlo-based derivative estimation. *Computational Statistics and Data Analysis* 51, 4465-4483.
- Snijders, Tom A.B. (2001). The statistical evaluation of social network dynamics. *Sociological Methodology*, 31, 361-395.
- Snijders, Tom A.B. (2017). Stochastic Actor-Oriented Models for Network Dynamics. *Annual Review of Statistics and Its Application*, 4, 343-363.
- Snijders, Tom A.B., van de Bunt, Gerhard G., and Steglich, Christian E.G. (2010). Introduction to actor-based models for network dynamics. *Social Networks*, 32, 44-60.
- Snijders, Tom A.B., Steglich, Christian E.G., and Schweinberger, Michael (2007). Modeling the co-evolution of networks and behavior. Pp. 41-71 in *Longitudinal models in the behavioral and related sciences*, edited by Kees van Montfort, Han Oud and Albert Satorra; Lawrence Erlbaum.
- Steglich, Christian E.G., Snijders, Tom A.B., and Pearson, Michael A. (2010). Dynamic networks and behavior: Separating selection from influence. *Sociological Methodology*, 40, 329-393.
- The manual: http://www.stats.ox.ac.uk/~snijders/siena/RSiena_Manual.pdf
- The website: http://www.stats.ox.ac.uk/~snijders/siena/.

See Also

siena07

Examples

```
mynet1 <- sienaDependent(array(c(tmp3, tmp4), dim=c(32, 32, 2)))
mydata <- sienaDataCreate(mynet1)
myeff <- getEffects(mydata)
myeff <- includeEffects(myeff, transTrip)
myeff
myalgorithm <- sienaAlgorithmCreate(nsub=3, n3=200)
ans <- siena07(myalgorithm, data=mydata, effects=myeff, batch=TRUE)
summary(ans)
```

`allEffects` *Internal data frame used to construct effect objects.*

Description

This data frame is used internally to construct effect objects.

Usage

```
data(allEffects)
```

Format

A data frame with values for the following 23 variables.

`effectGroup` a character vector

`effectName` a character vector

`functionName` a character vector

`shortName` a character vector

`endowment` a logical vector

`interaction1` a character vector

`interaction2` a character vector

`type` a character vector

`basicRate` a logical vector

`include` a logical vector

`randomEffects` a logical vector

`fix` a logical vector

`test` a logical vector

`timeDummy` a character vector, default ","

`initialValue` a numeric vector

`parm` a numeric vector

`functionType` a character vector

`period` a character vector

`rateType` a character vector

`untrimmedValue` a numeric vector

`effect1` a logical vector

`effect2` a logical vector

`effect3` a logical vector

`interactionType` a character vector

`local` a logical vector

`setting` Settings name: ” (no settings), ’primary’, ’universal’ or the name of the defining covariate.

Details

Used to define effects. Not for general user use.

References

See http://www.stats.ox.ac.uk/~snijders/siena/

`coCovar` *Function to create a constant covariate object*

Description

This function creates a constant covariate object from a vector.

Usage

```
coCovar(val, centered=TRUE, nodeSet="Actors", imputationValues=NULL)
```

Arguments

`val`	Vector of covariate values
`centered`	Boolean: if TRUE, then the mean value is subtracted.
`nodeSet`	Name of node set: character string. If the entire data set contains more than one node set, then the node sets must be specified in all data objects.
`imputationValues`	Vector of covariate values of same length as `val`, to be used for imputation of `NA` values (if any) in `val`. Must not contain any `NA`.

Details

When part of a Siena data object, the covariate is associated with the node set `nodeSet` of the Siena data object.
If there are any `NA` values in `val`, and `imputationValues` is given, then the corresponding elements of `imputationValues` are used for imputation. If `imputationValues` is `NULL`, imputation is by the mean value. In both cases, cases with imputed values are not used for calculating target statistics (see the manual).

Value

Returns the covariate as an object of class "coCovar", in which form it can be used as an argument to `sienaDataCreate`.

Author(s)

Ruth Ripley

References

See http://www.stats.ox.ac.uk/~snijders/siena/

See Also

sienaDataCreate, varCovar, coDyadCovar, varDyadCovar

Examples

```
myconstCovar <- coCovar(s50a[,1])
```

---

coDyadCovar *Function to create a constant dyadic covariate object.*

Description

This function creates a constant dyadic covariate object from a matrix.

Usage

```
coDyadCovar(val, centered=TRUE, nodeSets=c("Actors", "Actors"),
    sparse=is(val,"dgTMatrix"), type=c("oneMode", "bipartite"))
```

Arguments

val	Matrix of covariate values. May be sparse, of type "dgTMatrix".
centered	Boolean: if TRUE, then the mean value is subtracted.
nodeSets	The name of the node sets with which this covariate is associated. If the entire data set contains more than one node set, then the node sets must be specified in all data objects.
sparse	Boolean: whether a sparse matrix or not.
type	oneMode or bipartite: whether the matrix refers to a one-mode or a bipartite (two-mode) network.

Details

When part of a Siena data object, the covariate is assumed to be associated with the node sets named in nodeSets of the Siena data object. The name of the associated node sets will only be checked when the Siena data object is created.

Value

Returns the covariate as an object of class "coDyadCovar", in which form it can be used as an argument to sienaDataCreate.

Author(s)

Ruth Ripley

References

See http://www.stats.ox.ac.uk/~snijders/siena/

See Also

sienaDataCreate, varDyadCovar, coCovar, varCovar

Examples

```
mydyadvar <- coDyadCovar(s503)
```

---

edit.sienaEffects *Allow editing of a sienaEffects object if a gui is available.*

Description

Interactive editor for an effects object. A wrapper to edit.data.frame.

Usage

```
## S3 method for class 'sienaEffects'
edit(name, ...)
```

Arguments

name	An object of class sienaEffects
...	For extra arguments (none used at present)

Details

Will be invoked by fix(name) for an object of class sienaEffects.

Value

The updated object. There is no backup copy, and the edits cannot be undone.

Author(s)

Ruth Ripley

References

See http://www.stats.ox.ac.uk/~snijders/siena/

See Also

getEffects

Examples

```
mynet1 <- sienaDependent(array(c(s501, s502, s503), dim=c(50, 50, 3)))
mybeh <- sienaDependent(s50a, type="behavior")
mycovar <- coCovar(rnorm(50))
mydyadcovar <- coDyadCovar(matrix(as.numeric(rnorm(2500) > 2), nrow=50))
mydata <- sienaDataCreate(mynet1, mybeh, mycovar, mydyadcovar)
myeff <- getEffects(mydata)
## Not run:
fix(myeff)

## End(Not run)
```

---

effectsDocumentation *Function to create a table of documentation of effect names, short names etc.*

Description

Produces a table of the shortnames and other information for effects, either in html or latex.

Usage

```
effectsDocumentation(effects = NULL, type = "html", display = (type=="html"),
    filename = ifelse(is.null(effects), "effects", deparse(substitute(effects))))
```

Arguments

effects	A Siena effects object, or NULL.
type	Type of output required. Valid options are "html" or "latex".
display	Boolean: should the output be displayed after creation. Only applicable to html output.
filename	Character string denoting file name.

Details

If `effects=NULL`, the `allEffects` object is written to a table, either latex or html. This table presents all the available effects present in this version of RSiena, not delimited by a particular data set. The default file name is "effects.tex" or "effects.pdf", respectively. The latter file is also shown in the browser when requesting
`RShowDoc("effects", package="RSiena")`

The table lists all effects, with their name, shortName, whether an endowment (and creation) effect exists, the value of an effect parameter - if any -, and the interactionType (which can be empty or: "ego" or "dyadic" for dependent network variables; "OK" for dependent behavior variables). The latter is important for knowing how the effects can be used in interaction effects. (See `includeInteraction`).

If an existing effects object is specified for `effects`, then all available effects in this effects object are listed. This table lists the name (i.e., dependent variable), effect name, shortName, type (rate/evaluation/endowment/creation), the variables defined as `interaction1` and `interaction2` (see `includeEffects`) that specify this effect, the value of an effect parameter - if any -, and the interactionType. The default root file name is the name of the input effects object.

Value

Nothing returned. Output files are created in the current working directory.

Author(s)

Ruth Ripley, Tom A.B. Snijders

References

See http://www.stats.ox.ac.uk/~snijders/siena/

See Also

`getEffects`, `includeEffects`, `summary.sienaEffects`, `includeInteraction`.

Examples

```
## Not run: effectsDocumentation()
```

---

getEffects *Function to create a Siena effects object*

Description

Creates a basic list of effects for all dependent variables in the input siena object.

Usage

```
getEffects(x, nintn = 10, behNintn=4, getDocumentation=FALSE)
```

Arguments

x	an object of class 'siena" or 'sienaGroup"
nintn	Number of user-defined network interactions that can later be created.
behNintn	Number of user-defined behavior interactions that can later be created.
getDocumentation	Flag to allow documentation of internal functions, not for use by users.

Details

Creates a data frame of effects for use in siena model fits. Note that the class of the return object may be lost if the data.frame is edited using `fix`. See `fix` and `edit.data.frame`.

Value

An object of class `sienaEffects` or `sienaGroupEffects`: this is a data frame, each part of which relates to one dependent variable in the input object, with columns

name	name of the dependent variable
effectName	name of the effect
functionName	name of the function
shortName	short name for the effect
interaction1	second variable to define the effect, if any
interaction2	third variable to define the effect, if any
type	"eval", "endow", "creation", or "rate"
basicRate	boolean: whether a basic rate parameter
include	boolean: include in the model to be fitted or not
randomEffects	boolean: random or fixed effect. Currently not used.
fix	boolean: fix parameter value or not
test	boolean: test parameter value or not
timeDummy	comma separated list of periods, or "all", or "," for none – which time dummy interacted parameters should be included?
initialValue	starting value for estimation, also used for `fix` and `test`.
parm	parameter values
functionType	"objective" or "rate"
period	period for basic rate parameters
rateType	"Structural", "covariate", "diffusion"
untrimmedValue	Used to store initial values which could be trimmed
effect1	Used to indicate effect number in user-specified interactions
effect2	Used to indicate effect number in user-specified interactions
effect3	Used to indicate effect number in user-specified interactions
interactionType	Defines "dyadic" or "ego" or "OK" effects, used in `includeInteraction`

local	whether a local effect; used for the option localML in sienaAlgorithmCreate
effectFn	here NULL, but could be replaced by a function later
statisticFn	here NULL, but could be replaced by a function later
netType	Type of dependent variable: "oneMode", "behavior", or "bipartite"
groupName	name of relevant group data object
group	sequential number of relevant group data object in total
effectNumber	a unique identifier of the row

The combination of name, shortName, interaction1, interaction2, and type uniquely identifies any effect other than basic rate effects and user-specified interaction effects. For the latter, effect1, effect2 and effect3 are also required for the identification. The combination name, shortName, period and group uniquely identifies a basic rate effect.

A list of all effects in a given effects object (e.g., myeff), including their names of dependent variables, effect names, short names, and values of interaction1 and interaction2 (if any), is obtained by executing effectsDocumentation(myeff).

Author(s)

Ruth Ripley

References

See http://www.stats.ox.ac.uk/~snijders/siena/

See Also

sienaDataCreate, sienaDataCreate, includeEffects, setEffect

Examples

```
mynet1 <- sienaDependent(array(c(s501, s502, s503), dim=c(50, 50, 3)))
mybeh <- sienaDependent(s50a, type="behavior")
mycovar <- coCovar(rnorm(50))
mydyadcovar <- coDyadCovar(matrix(as.numeric(rnorm(2500) > 2), nrow=50))
mydata <- sienaDataCreate(mynet1, mybeh, mycovar, mydyadcovar)
myeff <- getEffects(mydata)
myeff
```

---

hn3401 *Network data: excerpt from "Dutch Social Behavior Data Set" of Chris Baerveldt.*

Description

Matrices N3401, N3403, N3404, N3406, and HN3401, HN3403, HN3404, HN3406 are two waves of networks for four schools (numbered 1, 3, 4, 6): there is a tie from pupil i to pupil j if i says that he/she receives and/or gives emotional support from/to pupil j. The data are part of a larger data set (see source below) and were collected under the direction of Chris Baerveldt.

Format

Adjacency matrices for the network at two time points. The matrices with name N... are the first wave, those with name HN... are the second wave.

Source

http://www.stats.ox.ac.uk/~snijders/siena/CB_data.zip

References

Houtzager, B. & Baerveldt, C. (1999). Just like Normal. A Social Network Study of the Relation between Petty Crime and the Intimacy of Adolescent Friendships. *Social Behavior and Personality* 27(2), 177-192.

Snijders, Tom A.B, and Baerveldt, Chris (2003). A Multilevel Network Study of the Effects of Delinquent Behavior on Friendship Evolution. *Journal of Mathematical Sociology* 27, 123-151.

See http://www.stats.ox.ac.uk/~snijders/siena/BaerveldtData.html

Examples

```
mynet <- sienaDependent(array(c(N3401, HN3401), dim=c(45, 45, 2)))
mydata <- sienaDataCreate(mynet)
```

---

includeEffects *Function to include effects in a Siena model*

Description

This function can be used for model specification by modifying a Siena effects object.

Usage

```
includeEffects(myeff, ..., include = TRUE, name = myeff$name[1], type = "eval",
 interaction1 = "", interaction2 = "", fix=FALSE, test=FALSE, character=FALSE)
```

Arguments

myeff	a Siena effects object as created by getEffects
...	short names to identify the effects which should be included or excluded.
include	Boolean. default TRUE, but can be switched to FALSE to turn off an effect.
name	Name of network for which effects are being included. Defaults to the first in the effects object.
type	Type of effects to be included: "eval", "endow", "creation", or "rate".
interaction1	Name of siena object where needed to completely identify the effects e.g. covariate name or behavior variable name.

interaction2	Name of siena object where needed to completely identify the effects e.g. covariate name or behavior variable name.
fix	Boolean. Are the effects to be fixed at the value stored in myeff$initialValue or not.
test	Boolean. Are the effects to be tested or not (requires fix).
character	Boolean: are the effect names character strings or not.

Details

The effects indicated by the arguments ..., type, and (if necessary) interaction1 and interaction2 are included or excluded from the model specified by the effects object. The names interaction1 and interaction2 do not refer to interactions between effects, but to dependence of effects on other variables in the data set. The arguments should identify the effects completely.

The value of myeff$initialValue can be set by function setEffect.

A list of all effects available in a given effects object (e.g., myeff), including their names of dependent variables, effect names, short names, and values of interaction1 and interaction2 (if any), is obtained by executing effectsDocumentation(myeff).

The function includeEffects operates as an interface setting the "include" column on selected rows of the effects object, to the value requested (TRUE or FALSE).

Value

An updated version of the input effects object, with the include, test, and fix columns for one or more rows updated. Details of the rows altered will be printed.

Author(s)

Ruth Ripley

References

See http://www.stats.ox.ac.uk/~snijders/siena/

See Also

getEffects, setEffect, includeInteraction, effectsDocumentation

Examples

```
mynet1 <- sienaDependent(array(c(s501, s502, s503), dim=c(50, 50, 3)))
mybeh  <- sienaDependent(s50a, type="behavior")
mydata <- sienaDataCreate(mynet1, mybeh)
myeff <- getEffects(mydata)
myeff <- includeEffects(myeff, transTrip, balance)
myeff <- includeEffects(myeff, avAlt, name="mybeh", interaction1="mynet1")
myeff
```

includeInteraction *Function to create user-specified interactions for a Siena model.*

Description

This function allows the user to include or exclude an interaction effect in a Siena effects object.

Usage

```
includeInteraction(myeff, ..., include = TRUE, name = myeff$name[1],
    type = "eval", interaction1 = rep("", 3), interaction2 = rep("", 3),
    fix = FALSE, test = FALSE, parameter = 0, random = FALSE,
    character = FALSE, verbose = TRUE)
```

Arguments

myeff	a Siena effects object as created by getEffects
...	2 or 3 short names to identify the effects which should be interacted.
include	Boolean. default TRUE, but can be switched to FALSE to turn off an interaction.
name	Name of dependent variable (network or behavior) for which interactions are being defined. Defaults to the first in the effects object.
type	Type of effects to be interacted.
interaction1	Vector of Siena objects where needed to completely identify the effect e.g. covariate name or behavior variable name. Trailing blanks may be omitted.
interaction2	Vector of siena objects where needed to completely identify the effect e.g. covariate name or behavior variable name. Trailing blanks may be omitted.
fix	Boolean. Are the effects to be fixed at the value stored in myeff$initialValue or not.
test	Boolean. Are the effects to be tested or not (requires fix).
parameter	Value of internal effect parameter of this interaction effect. Default 0. If NULL, no change is made.
random	For specifying that the interaction effect will vary randomly; not relevant for RSiena at this moment. Boolean required. Default FALSE.
character	Boolean: are the effect names character strings or not.
verbose	Boolean: should the print of altered effects be produced.

Details

The details provided should uniquely identify up to three effects. If so, an interaction effect will be created and included or not in the model.
Whether interactions between two or three given effects can be created depends on their interactionType (which can be, for dependent network variables, empty, ego, or dyadic; and for dependent behavioral variables, empty or OK). Consult the section on Interaction Effects in the manual for this. The

interactionType is shown in the list of effects obtained from the function effectsDocumentation. From the point of view of model building it is usually advisable, when including an interaction effect in a model, also to include the corresponding main effects. This is however not enforced by includeInteraction().

Interaction effects are constructed from effects with shortName unspInt (for networks) and behUnspInt (for behavior) by specifying their elements effect1 and effect2, and possibly effect3.

The number of possible user-specified interaction effects is limited by the parameters nintn (for dependent network variables) and behNintn (for dependent behavior variables) in the call of getEffects, which determine the numbers of effects with shortNames unspInt and behUnspInt.

The input names interaction1 and interaction2 do not themselves refer to created interactions, but to dependence of the base effects on other variables in the data set. They are used to completely identify the effects.

A list of all effects in a given effects object (e.g., myeff), including their names of dependent variables, effect names, short names, and values of interaction1 and interaction2 (if any), is obtained by executing effectsDocumentation(myeff).

To set attributes of interaction effects in the effects object, function setEffect can also be used with short name unspInt or behUnspInt, and further using parameters effect1 and effect2, and possibly effect3.

Value

An updated version of the input effects object; if include, containing the interaction effect between "effect1" and "effect2" and possibly "effect3"; if not, without this interaction effect. The shortName of the interaction effect is "unspInt" for network effects and "behUnspInt" for behavior effects.
If verbose=TRUE, details of the fields altered will be printed.

Author(s)

Ruth Ripley, Tom Snijders

References

See http://www.stats.ox.ac.uk/~snijders/siena/

See Also

getEffects, setEffect, includeEffects, effectsDocumentation

Examples

```
mynet <- sienaDependent(array(c(s501, s502, s503), dim=c(50, 50, 3)))
alc <- varCovar(s50a)
mydata <- sienaDataCreate(mynet, alc)
myeff <- getEffects(mydata)
myeff <- includeEffects(myeff, transTrip)
myeff <- includeEffects(myeff, egoX, altX, simX, interaction1="alc")
myeff <- includeInteraction(myeff, recip, simX, interaction1=c("", "alc"))
myeff <- includeInteraction(myeff, transTrip, egoX, interaction1=c("", "alc"))
myeff
```

```
# How to set the effect parameter of an interaction:
myeff <- getEffects(mydata)
myeff <- setEffect(myeff, gwespFF, parameter=69)
myeff <- includeInteraction(myeff, recip, gwespFF, parameter=69)
myeff
```

`includeTimeDummy` *Function to include time dummy effects in a Siena model*

Description

This function specifies time heterogeneity for selected effects in a Siena model, by interacting them with time dummies, without explicitly using time-dependent covariates.

Usage

```
includeTimeDummy(myeff, ..., timeDummy="all", name=myeff$name[1], type="eval",
              interaction1="", interaction2="", include=TRUE, character=FALSE)
```

Arguments

`myeff`	A Siena effects object as created by `getEffects`.
`...`	Short names to identify the effects for which interactions with time dummies should be included or excluded. This function cannot be used for regular interaction effects.
`timeDummy`	Character string. Either "all" or the periods for which to create dummies (from 1 to (number of waves - 1)), space delimited.
`include`	Boolean. default TRUE, but can be switched to FALSE to turn off an effect.
`name`	Name of dependent network or behavioral variable for which effects are being included. Defaults to the first in the effects object.
`type`	Type of dummy effects to be interacted.
`interaction1`	Name of variable where needed to completely identify the effects e.g. covariate name or behavior variable name.
`interaction2`	Name of variable where needed to completely identify the effects e.g. covariate name or behavior variable name.
`character`	Boolean: are the effect names character strings or not

Details

The arguments (`...`, `name`, `interaction1`, `interaction2`) should identify the effects completely. See `includeEffects` and `effectsDocumentation` for more information about this.

This function operates by setting the `timeDummy` column on selected rows of a Siena effects object, thereby specifying interactions of the specified effect or effects with dummy variables for the specified periods. The `timeDummy` column of `myeff` will be set to include the values requested if `include=TRUE`, and to exclude them for `include=FALSE`.

For an effects object in which the `timeDummy` column of some of the included effects includes some or all period numbers, interactions of those effects with ego effects of time dummies for the indicated periods will also be estimated by `siena07`. For the outdegree effect this is just the ego effect of the time dummies. If ... does not include the outdegree effect, then still this ego effect will be created, but its parameter will be fixed to 0.

An alternative to the use of `includeTimeDummy` is to define time-dependent actor covariates (dummy variables or other functions of wave number that are the same for all actors), include these in the data set through `sienaDataCreate`, and include interactions of other effects with ego effects of these time-dependent actor covariates by `includeInteraction`. This is illustrated in an example in the help file for `sienaTimeTest`. Using `includeTimeDummy` is easier; on the other hand, using self-defined interactions with time-dependent variables gives more control (e.g., it will allow to specify linear time dependence and test time heterogeneity for interaction effects).

Value

An updated version of `myeff`, with the `timeDummy` column for one or more rows updated. Details of the rows altered will be printed.

Author(s)

Josh Lospinoso

References

See http://www.stats.ox.ac.uk/~snijders/siena/ for general information on RSiena.

See Also

`sienaTimeTest`, `getEffects`, `includeEffects`, `effectsDocumentation`.

Examples

```
## Not run:
## Estimate a restricted model
myalgorithm <- sienaAlgorithmCreate(nsub=4, n3=1000)
mynet1 <- sienaDependent(array(c(s501, s502, s503), dim=c(50, 50, 3)))
mydata <- sienaDataCreate(mynet1)
myeff <- getEffects(mydata)
myeff <- includeEffects(myeff, transTrip, balance)
myeff
(ans <- siena07(myalgorithm, data=mydata, effects=myeff))

## Conduct the score type test to assess whether heterogeneity is present.
tt <- sienaTimeTest(ans)
summary(tt)

## Suppose that we wish to include a time dummy.
## Since there are three waves, the number of periods is two.
## This means that only one time dummy can be included for
## the interactions. The default is for period 2;
## an equivalent model, but with different parameters
```

```
## (that can be transformed into each other) is obtained
## when the dummies are defined for period 1.
myeff <- includeTimeDummy(myeff, density, recip, timeDummy="2")
myeff       # Note the \code{timeDummy} column.
(ans2 <- siena07(myalgorithm, data=mydata, effects=myeff))

## Re-assess the time heterogeneity
tt2 <- sienaTimeTest(ans2)
summary(tt2)

## And so on..

## End(Not run)

## A demonstration of RateX heterogeneity.
## Note that rate interactions are not implemented in general,
## but they are for Rate x coCovar.
## Not run:
myalgorithm <- sienaAlgorithmCreate(nsub=4, n3=1000)
mynet1 <- sienaDependent(array(c(s501, s502, s503), dim=c(50, 50, 3)))
myccov <- coCovar(s50a[,1])
mydata <- sienaDataCreate(mynet1, myccov)
myeff <- getEffects(mydata)
myeff <- includeEffects(myeff, transTrip, balance)
myeff <- includeTimeDummy(myeff, RateX, type="rate",
            interaction1="myccov")
myeff
(ans <- siena07(myalgorithm, data=mydata, effects=myeff))

## End(Not run)
```

`iwlsm` *Function to fit an iterated weighted least squares model.*

Description

Fits an iterated weighted least squares model.

Usage

```
iwlsm(x, ...)

## S3 method for class 'formula'
iwlsm(formula, data, weights, ses, ..., subset, na.action,
    method = c("M", "MM", "model.frame"),
    wt.method = c("inv.var", "case"),
    model = TRUE, x.ret = TRUE, y.ret = FALSE, contrasts = NULL)

## Default S3 method:
```

```
iwlsm(x, y, weights, ses, ..., w = rep(1/nrow(x), nrow(x)),
    init = "ls", psi = psi.iwlsm,
    scale.est = c("MAD", "Huber", "proposal 2"), k2 = 1.345,
    method = c("M", "MM"), wt.method = c("inv.var", "case"),
    maxit = 20, acc = 1e-4, test.vec = "resid", lqs.control = NULL)

psi.iwlsm(u, k, deriv = 0, w, sj2, hh)
```

Arguments

formula	a formula of the form `y ~ x1 + x2 + ...`.
data	data frame from which variables specified in `formula` are preferentially to be taken.
weights	a vector of prior weights for each case.
subset	An index vector specifying the cases to be used in fitting.
ses	Estimated variance of the responses. Will be paseed to `psi` as sj2
na.action	A function to specify the action to be taken if NAs are found. The 'factory-fresh' default action in R is `na.omit`, and can be changed by `options(na.action=)`.
x	a matrix or data frame containing the explanatory variables.
y	the response: a vector of length the number of rows of `x`.
method	Must be "M". (argument not used here).
wt.method	are the weights case weights (giving the relative importance of case, so a weight of 2 means there are two of these) or the inverse of the variances, so a weight of two means this error is half as variable? This will not work at present.
model	should the model frame be returned in the object?
x.ret	should the model matrix be returned in the object?
y.ret	should the response be returned in the object?
contrasts	optional contrast specifications: se `lm`.
w	(optional) initial down-weighting for each case. Will not work at present.
init	(optional) initial values for the coefficients OR a method to find initial values OR the result of a fit with a `coef` component. Known methods are `"ls"` (the default) for an initial least-squares fit using weights `w*weights`, and `"lts"` for an unweighted least-trimmed squares fit with 200 samples. Probably not functioning.
psi	the psi function is specified by this argument. It must give (possibly by name) a function `g(x, ..., deriv, w)` that for `deriv=0` returns psi(x)/x and for `deriv=1` returns some value. Extra arguments may be passed in via `...`.
scale.est	method of scale estimation: re-scaled MAD of the residuals (default) or Huber"s proposal 2 (which can be selected by either `"Huber"` or `"proposal 2"`).
k2	tuning constant used for Huber proposal 2 scale estimation.
maxit	the limit on the number of IWLS iterations.
acc	the accuracy for the stopping criterion.

test.vec	the stopping criterion is based on changes in this vector.
...	additional arguments to be passed to iwlsm.default or to the psi function.
lqs.control	An optional list of control values for lqs.
u	numeric vector of evaluation points.
k	tuning constant. Not used.
deriv	0 or 1: compute values of the psi function or of its first derivative. (Latter not used).
sj2	Estimated variance of the responses
hh	Diagonal values of the hat matrix

Details

This function is very slightly adapted from rlm in packages MASS. It alternates between weighted least squares and estimation of variance on the basis of a common variance. The function psi.iwlsm calculates the weights for the next iteration. Used by siena08 to combine estimates from different sienaFits.

Value

An object of class "iwlsm" inheriting from "lm". Note that the df.residual component is deliberately set to NA to avoid inappropriate estimation of the residual scale from the residual mean square by "lm" methods.

The additional components not in an lm object are

s	the robust scale estimate used
w	the weights used in the IWLS process
psi	the psi function with parameters substituted
conv	the convergence criteria at each iteration
converged	did the IWLS converge?
wresid	a working residual, weighted for "inv.var" weights only.

Note

The function has been changed as little as possible, but has only been used with default arguments. The other options have been retained just in case they may prove useful.

Author(s)

Ruth Ripley

References

Venables, W. N. and Ripley, B. D. (2002) *Modern Applied Statistics with S*. Fourth edition. Springer.
See also http://www.stats.ox.ac.uk/~snijders/siena/

See Also

siena08, sienaMeta, sienaFit

Examples

```
## Not run:
##not enough data here for a sensible example, but shows the idea.
myalgorithm <- sienaAlgorithmCreate(nsub=2, n3=100)
mynet1 <- sienaDependent(array(c(s501, s502), dim=c(50, 50, 2)))
mynet2 <- sienaDependent(array(c(s502, s503), dim=c(50, 50, 2)))
mydata1 <- sienaDataCreate(mynet1)
mydata2 <- sienaDataCreate(mynet2)
myeff1 <- getEffects(mydata1)
myeff2 <- getEffects(mydata2)
myeff1 <- setEffect(myeff1, transTrip, fix=TRUE, test=TRUE)
myeff2 <- setEffect(myeff2, transTrip, fix=TRUE, test=TRUE)
myeff1 <- setEffect(myeff1, cycle3, fix=TRUE, test=TRUE)
myeff2 <- setEffect(myeff2, cycle3, fix=TRUE, test=TRUE)
ans1 <- siena07(myalgorithm, data=mydata1, effects=myeff1, batch=TRUE)
ans2 <- siena07(myalgorithm, data=mydata2, effects=myeff2, batch=TRUE)
meta <- siena08(ans1, ans2)
metadf <- split(meta$thetadf, meta$thetadf$effects)[[1]]
metalm <- iwlsm(theta ~ tconv, metadf, ses=se^2)

## End(Not run)
```

---

maxlikefn *A ML version of FRAN*

Description

A function to be called as "FRAN". All in R. very slow. work in progress.

Usage

```
maxlikefn(z, x, INIT = FALSE, TERM = FALSE, data,
effects = NULL, nstart = 1000, pinsdel = 0.6,
pperm = 0.3, prelins = 0.1, multfactor=2, promul = 0.1,
promul0 = 0.5, pdiaginsdel = 0.1, fromFiniteDiff = FALSE,
noSamples = 1, sampInterval = 50, int = 1)
```

Arguments

z	control object, passed in automatically in Siena07
x	model object, passed in automatically in Siena07
INIT	if TRUE, do initial processing. May be required to set up z
TERM	if TRUE, do end processing.

data	A siena object
effects	list of data frames as returned by getEffects
nstart	Number of MH steps at the start, after making the chain
pinsdel	Probability of insert/delete step
pperm	Probability of permutation step. (set to zero in startup phase.)
prelins	Insertion probability in InsDelPermute
multfactor	Factor controlling number of MH steps. Will be read from the model in preference, and that is easier to alter! But I don"t want to alter that program yet..
promul	Probability of choosing a random single multiple in InsDelPermute in start up phase.
promul0	Probability of choosing a random single multiple in InsDelPermute not in startup phase
pdiaginsdel	Probability of insertion or deletion of a diagonal link.
fromFiniteDiff	Should always be FALSE
noSamples	Number of chains to be returned
sampInterval	If multiple chains are returned, the number of steps between each
int	Number of parallel MCMC chains to pursue.

Details

This can be used for the element FRAN of the model object. The arguments with no defaults must be passed in on the call to siena07. Also you must set the option `maxlike=TRUE` in the call to `sienaAlgorithmCreate()`

Value

Depends on the call. If `INIT` or `initC` or `TERM` are true, returns `z`, the control object. Otherwise, returns a list containing:

fra	Simulated scores
dff	2nd deriv, not phase 2
OK	could be set to FALSE if serious error has occurred

Author(s)

Ruth Ripley

References

See http://www.stats.ox.ac.uk/~snijders/siena/

See Also

siena07

Examples

```
## Not run:
mynet1 <- sienaDependent(array(c(tmp3, tmp4), dim=c(32, 32, 2)))
mydata <- sienaDataCreate(mynet1)
myeff<- getEffects(mydata)
myalgor<- sienaAlgorithmCreate(nsub=2, n3=100, maxlike=TRUE)
ans<- siena07(myalgor, data=mydata, effects=myeff,  batch=TRUE)

## End(Not run)
```

---

n3401 *Network data: excerpt from "Dutch Social Behavior Data Set" of Chris Baerveldt.*

Description

Matrices N3401, N3403, N3404, N3406, and HN3401, HN3403, HN3404, HN3406 are two waves of networks for four schools (numbered 1, 3, 4, 6): there is a tie from pupil i to pupil j if i says that he/she receives and/or gives emotional support from/to pupil j. The data are part of a larger data set (see source below) and were collected under the direction of Chris Baerveldt.

Format

Adjacency matrices for the network at two time points. The matrices with name N... are the first wave, those with name HN... are the second wave.

Source

http://www.stats.ox.ac.uk/~snijders/siena/CB_data.zip

References

Houtzager, B. & Baerveldt, C. (1999). Just like Normal. A Social Network Study of the Relation between Petty Crime and the Intimacy of Adolescent Friendships. *Social Behavior and Personality* 27(2), 177-192.

Snijders, Tom A.B, and Baerveldt, Chris (2003). A Multilevel Network Study of the Effects of Delinquent Behavior on Friendship Evolution. *Journal of Mathematical Sociology* 27, 123-151.

See http://www.stats.ox.ac.uk/~snijders/siena/BaerveldtData.html

Examples

```
mynet <- sienaDependent(array(c(N3401, HN3401), dim=c(45, 45, 2)))
mydata <- sienaDataCreate(mynet)
```

plot.sienaTimeTest *Functions to plot assessment of time heterogeneity of parameters*

Description

Plot method for sienaTimeTest objects.

Usage

```
## S3 method for class 'sienaTimeTest'
plot(x, pairwise=FALSE, effects,
 scale=.2, plevels=c(.1, .05, .025), ...)
```

Arguments

x	A sienaTimeTest object returned by sienaTimeTest.
pairwise	A Boolean value corresponding to whether the user would like a pairwise plot of the simulated statistics to assess correlation among the effects (pairwise=TRUE), or a plot of the estimates across waves in order to assess graphically the results of the score type test.
effects	A vector of integers corresponding to the indices given in the sienaTimeTest output for effects which are to be plotted.
scale	A positive number corresponding to the number of standard deviations on one step estimates to use for computing the maximum and minimum of the plotting range. We recommend experimenting with this number when the y-axes of the plots are not satisfactory. Smaller numbers shrink the axes.
plevels	A list of three decimals indicating the gradients at which to draw the confidence interval bars.
...	For extra arguments. The Lattice parameter layout can be used to control the layout of the graphs.

Details

The pairwise=TRUE plot may be used to assess whether effects are highly correlated. This information may be important when considering forward-model selection, since highly correlated effects may have highly correlated one-step estimates, particularly since the individual score type tests are not orthogonalized against the scores and deviations of yet-unestimated dummies. For example, reciprocity and outdegree may have highly correlated statistics as indicated by a strong, positive correlation coefficient. When considering whether to include dummy terms, it may be a good idea to include, e.g., outdegree, estimate the parameter, and see whether reciprocity dummies remain significant after method of moments estimation of the updated model–as opposed to including both outdegree and reciprocity.

The pairwise=FALSE plot displays the most of the information garnered from sienaTimeTest in a graphical fashion. For each effect, the method of moments parameter estimate for the base period (i.e. wave 1) is given as a blue, horizontal reference line. One step estimates are given for all of

the parameters by dots at each wave. The dots are colored black if the parameter has been included in the model already (i.e. has been estimated via method of moments), or red if they have not been included. Confidence intervals are given based on pivots given at `pvalues`. Evidence of time heterogeneity is suggested by points with confidence intervals not overlapping with the base period.

Value

None

Author(s)

Josh Lospinoso

References

See http://www.stats.ox.ac.uk/~snijders/siena/ for general information on RSiena.

See Also

siena07, sienaTimeTest, xyplot

Examples

```
## Not run:
myalgorithm <- sienaAlgorithmCreate(nsub=4, n3=500)
mynet1 <- sienaDependent(array(c(s501, s502, s503, s501, s503, s502), dim=c(50, 50, 6)))
mydata <- sienaDataCreate(mynet1)
myeff <- getEffects(mydata)
myeff <- includeEffects(myeff, transTrip, balance)
myeff <- includeTimeDummy(myeff, recip, timeDummy="2,3,5")
myeff <- includeTimeDummy(myeff, balance, timeDummy="4")
myeff <- includeTimeDummy(myeff, density, timeDummy="all")
ansp <- siena07(myalgorithm, data=mydata, effects=myeff, batch=TRUE)
ttp <- sienaTimeTest(ansp)

## Pairwise plots show
plot(ttp, pairwise=TRUE)

## Time test plots show
plot(ttp, effects=1:3) ## default layout
plot(ttp, effects=1:3, layout=c(3,1))

## End(Not run)
```

print.sienaEffects *Print methods for Siena effects objects*

Description

Print the major columns of the effects object. Or all, with any non atomic columns listed separately.

Usage

```
## S3 method for class 'sienaEffects'
print(x, fileName = NULL, includeOnly=TRUE,
expandDummies = FALSE, includeRandoms = FALSE, dropRates=FALSE, ...)
## S3 method for class 'sienaEffects'
summary(object, fileName = NULL,
includeOnly=TRUE, expandDummies = FALSE, ...)
## S3 method for class 'summary.sienaEffects'
print(x, fileName = NULL, ...)
```

Arguments

object	An object of class sienaEffects.
x	An object of class sienaEffects or summary.sienaEffects as appropriate.
fileName	Character string denoting file name if file output desired.
includeOnly	Boolean. If TRUE, only effects with the include flag TRUE will be printed.
expandDummies	Interpret the timeDummy column and show any effects which would be added by sienaTimeFix.
includeRandoms	Boolean. If TRUE, also the randomEffects column will be printed.
dropRates	Boolean. If TRUE, do not print the rows for basic rate effects.
...	For extra arguments (none used at present).

Value

The function print.sienaEffects prints details of the main columns of the selected rows of the effects object.

The function summary.sienaEffects checks the rows for valid printing via print.data.frame and excludes any that will fail. The OK columns are printed first, followed by any others.

Output from either can be directed to a file by using the argument filename.

Author(s)

Ruth Ripley, modifications by Tom Snijders.

References

See http://www.stats.ox.ac.uk/~snijders/siena/

See Also

sienaTimeTest, effectsDocumentation

Examples

```
mynet1 <- sienaDependent(array(c(s501, s502, s503), dim=c(50, 50, 3)))
mybeh <- sienaDependent(s50a, type="behavior")
mycovar <- coCovar(rnorm(50))
mydyadcovar <- coDyadCovar(matrix(as.numeric(rnorm(2500) > 2), nrow=50))
mydata <- sienaDataCreate(mynet1, mybeh, mycovar, mydyadcovar)
myeff <- getEffects(mydata)
myeff
summary(myeff)
```

---

print.sienaMeta *Methods for processing sienaMeta objects*

Description

print, summary, and plot methods for sienaMeta objects.

Usage

```
## S3 method for class 'sienaMeta'
print(x, file=FALSE, ...)

## S3 method for class 'sienaMeta'
summary(object, file=FALSE, extra=TRUE, ...)

## S3 method for class 'summary.sienaMeta'
print(x, file=FALSE, extra=TRUE, ...)

## S3 method for class 'sienaMeta'
plot(x,  ..., layout=c(2,2))
```

Arguments

object	An object of class sienaMeta
x	An object of class sienaMeta, or summary.sienaMeta as appropriate
file	Boolean: if TRUE, sends output to file named x$projname.out. If FALSE, output is to the terminal.
extra	Boolean: if TRUE, prints more information
layout	the vector giving number of rows and columns in the arrangement of the several panels in a rectangular array, possibly spanning multiple pages
...	For extra arguments (none used at present)

Value

The function `print.sienaMeta` prints details of the merged estimates of the meta-analysis carried out by `siena08`, with test statistics. See the help page for `siena08` for what is produced by this function.

The function `summary.sienaMeta` prints details as for the `print` method, but also details of the `sienaFit` objects included.

Output from either can be directed to a file by using the argument `file`. It will be appended to any existing file of the same name: `projname.out` where `projname` is the value of the argument to `siena08`.

The function `plot.sienaMeta` plots estimates against standard errors for each effect, with reference lines added at the two-sided significance threshold 0.05. It returns an object of class `trellis`, of the `lattice`.package. Effects for which a score test was requested are not plotted.

Author(s)

Ruth Ripley, Tom Snijders

References

T. A. B. Snijders and Chris Baerveldt. "Multilevel network study of the effects of delinquent behavior on friendship evolution". *Journal of Mathematical Sociology*, 27: 123–151, 2003.

See also the Siena manual and http://www.stats.ox.ac.uk/~snijders/siena/

See Also

`siena08`

Examples

```
## Not run:
# A meta-analysis for three groups does not make much sense
# for generalizing to a population of networks,
# but it the Fisher combinations of p-values are meaningful.
# But using three groups shows the idea.

Group1 <- sienaDependent(array(c(N3401, HN3401), dim=c(45, 45, 2)))
Group3 <- sienaDependent(array(c(N3403, HN3403), dim=c(37, 37, 2)))
Group4 <- sienaDependent(array(c(N3404, HN3404), dim=c(33, 33, 2)))
dataset.1 <- sienaDataCreate(Friends = Group1)
dataset.3 <- sienaDataCreate(Friends = Group3)
dataset.4 <- sienaDataCreate(Friends = Group4)
OneAlgorithm <- sienaAlgorithmCreate(projname = "SingleGroups")
effects.1 <- getEffects(dataset.1)
effects.3 <- getEffects(dataset.3)
effects.4 <- getEffects(dataset.4)
effects.1 <- includeEffects(effects.1, transTrip)
effects.1 <- setEffect(effects.1, cycle3, fix=TRUE, test=TRUE)
effects.3 <- includeEffects(effects.3, transTrip)
effects.3 <- setEffect(effects.3, cycle3, fix=TRUE, test=TRUE)
```

```
effects.4 <- includeEffects(effects.4, transTrip)
effects.4 <- setEffect(effects.4, cycle3, fix=TRUE, test=TRUE)
ans.1 <- siena07(OneAlgorithm, data=dataset.1, effects=effects.1, batch=TRUE)
ans.3 <- siena07(OneAlgorithm, data=dataset.3, effects=effects.3, batch=TRUE)
ans.4 <- siena07(OneAlgorithm, data=dataset.4, effects=effects.4, batch=TRUE)
ans.1
ans.3
ans.4
(meta <- siena08(ans.1, ans.3, ans.4))
summary(meta)
# For specifically presenting the Fisher combinations:
# First determine the number of estimated effects:
(neff <- sum(sapply(meta, function(x){ifelse(is.list(x),!is.null(x$cjplus),FALSE)})))
Fishers <- t(sapply(1:neff,
        function(i){c(meta[[i]]$cjplus, meta[[i]]$cjminus,
                        meta[[i]]$cjplusp, meta[[i]]$cjminusp, 2*meta[[i]]$n1 )}))
Fishers <- as.data.frame(Fishers, row.names=names(meta)[1:neff])
names(Fishers) <- c('Fplus', 'Fminus', 'pplus', 'pminus', 'df')
Fishers
# For plotting:
plo <- plot(meta, layout = c(3,1))
plo
plo[3]

## End(Not run)
```

`print01Report` *Function to produce the Siena01 report from R objects*

Description

Prints a report of a Siena data object and its default effects.

Usage

```
print01Report(data, modelname = "Siena", getDocumentation=FALSE)
```

Arguments

`data`	a Siena data object
`modelname`	Character string used to name the output file "modelname.out"
`getDocumentation`	Flag to allow documentation of internal functions, not for use by users.

Details

First deletes any file of the name "modelname.out", then prints a new one.

Value

No value returned.

Author(s)

Ruth Ripley

References

See http://www.stats.ox.ac.uk/~snijders/siena/

Examples

```
mynet1 <- sienaDependent(array(c(s501, s502, s503), dim=c(50, 50, 3)))
mydata <- sienaDataCreate(mynet1)
## Not run:
print01Report(mydata, modelname="mydescription")

## End(Not run)
```

---

s50 *Network data: excerpt from "Teenage Friends and Lifestyle Study" data.*

Description

An excerpt of the network, alcohol consumption, and smoking data for 50 randomly chosen girls from the Teenage Friends and Lifestyle Study data set. Useful as a small example of network and behaviour, for which models can be fitted quickly, and for which there are no missing values.

Format

Adjacency matrix for the network at time points 1, 2, 3; 50 by 3 matrices of alcohol consumption and smoking data for the three time points.

Source

http://www.stats.ox.ac.uk/~snijders/siena/s50_data.zip

References

West, P. and Sweeting, H. (1995) Background Rationale and Design of the West of Scotland 11-16 Study. Working Paper No. 52. MRC Medical Sociology Unit Glasgow.

See http://www.stats.ox.ac.uk/~snijders/siena/s50_data.htm

See Also

s501, s502, s503, s50a, s50s

Examples

```
mynet <- sienaDependent(array(c(s501, s502, s503), dim=c(50, 50, 3)))
mybeh <- sienaDependent(s50a, type="behavior")
mydata <- sienaDataCreate(mynet, mybeh)
mydata
```

---

s501 *Network 1 data: excerpt from "Teenage Friends and Lifestyle Study" data.*

Description

First timepoint network data from an excerpt of 50 girls from the Teenage Friends and Lifestyle Study data set. Useful as a small example of network and behaviour, for which models can be fitted quickly.

Format

The adjacency matrix for the network at time point 1.

Source

http://www.stats.ox.ac.uk/~snijders/siena/s50_data.zip

References

West, P. and Sweeting, H. (1995) Background Rationale and Design of the West of Scotland 11-16 Study. Working Paper No. 52. MRC Medical Sociology Unit Glasgow.

See http://www.stats.ox.ac.uk/~snijders/siena/s50_data.htm

See Also

s502, s503, s50a, s50s

---

s502 *Network 2 data: excerpt from "Teenage Friends and Lifestyle Study" data.*

Description

Second timepoint network data from an excerpt of 50 girls from the Teenage Friends and Lifestyle Study data set. Useful as a small example of network and behaviour, for which models can be fitted quickly.

Format

The adjacency matrix for the network at time point 2.

Source

http://www.stats.ox.ac.uk/~snijders/siena/s50_data.zip

References

West, P. and Sweeting, H. (1995) Background Rationale and Design of the West of Scotland 11-16 Study. Working Paper No. 52. MRC Medical Sociology Unit Glasgow.

See http://www.stats.ox.ac.uk/~snijders/siena/s50_data.htm

See Also

s501, s503, s50a, s50s, s50

---

s503 *Network 3 data: excerpt from "Teenage Friends and Lifestyle Study" data.*

Description

Second timepoint network data from an excerpt of 50 girls from the Teenage Friends and Lifestyle Study data set. Useful as a small example of network and behaviour, for which models can be fitted quickly.

Format

Adjacency matrix for the network at time point 3.

Source

http://www.stats.ox.ac.uk/~snijders/siena/s50_data.zip

References

West, P. and Sweeting, H. (1995) Background Rationale and Design of the West of Scotland 11-16 Study. Working Paper No. 52. MRC Medical Sociology Unit Glasgow.

See http://www.stats.ox.ac.uk/~snijders/siena/s50_data.htm

See Also

s501, s502, s50a, s50s

Examples

```
mynet <- sienaDependent(array(c(s501, s502, s503), dim=c(50, 50, 3)))
mybeh <- sienaDependent(s50a, type="behavior")
mydata <- sienaDataCreate(mynet, mybeh)
```

---

s50a *Alcohol use data: excerpt from "Teenage Friends and Lifestyle Study" data*

Description

Data from an excerpt of 50 girls from the Teenage Friends and Lifestyle Study data set. Useful as a small example of network and behaviour, for which models can be fitted quickly.

Format

A matrix of variables relating to the use of alcohol for the actors in the network. Three columns, one for each time point. Coding is 1–5, high values indicating higher consumption.

Source

http://www.stats.ox.ac.uk/~snijders/siena/s50_data.zip

References

West, P. and Sweeting, H. (1995) Background Rationale and Design of the West of Scotland 11-16 Study. Working Paper No. 52. MRC Medical Sociology Unit Glasgow.

See http://www.stats.ox.ac.uk/~snijders/siena/s50_data.htm

See Also

s501, s502, s503, s50s

Examples

```
mynet <- sienaDependent(array(c(s501, s502, s503), dim=c(50, 50, 3)))
mybeh <- sienaDependent(s50a, type="behavior")
mydata <- sienaDataCreate(mynet, mybeh)
mydata
```

s50s — *Smoking data: excerpt from "Teenage Friends and Lifestyle Study" data*

Description

Data from an excerpt of 50 girls from the Teenage Friends and Lifestyle Study data set. Useful as a small example of network and behaviour, for which models can be fitted quickly.

Format

A matrix of variables relating to the smoking habits for the actors in the network. Three columns, one for each time point. Coding is 1–3: 1 = no smoking, 2 = moderate smoking, 3 = serious smoking.

Source

http://www.stats.ox.ac.uk/~snijders/siena/s50_data.zip

References

West, P. and Sweeting, H. (1995) Background Rationale and Design of the West of Scotland 11-16 Study. Working Paper No. 52. MRC Medical Sociology Unit Glasgow.

See http://www.stats.ox.ac.uk/~snijders/siena/s50_data.htm

See Also

s501, s502, s503, s50a

Examples

```
mynet <- sienaDependent(array(c(s501, s502, s503), dim=c(50, 50, 3)))
myvar <- varCovar(s50s)
mydata <- sienaDataCreate(mynet, myvar)
mydata
```

setEffect — *Function to set various columns in an effects object in a Siena model.*

Description

This function provides an interface to change various columns of a selected row of a Siena effects object.

Usage

```
setEffect(myeff, shortName, parameter = NULL, fix = FALSE,
test = FALSE, random=FALSE, initialValue = 0, timeDummy = ",", include = TRUE,
name = myeff$name[1], type = "eval", interaction1 = "",
interaction2 = "", effect1=0, effect2=0, effect3=0,
period=1, group=1, character=FALSE)
```

Arguments

myeff	a Siena effects object as created by getEffects
shortName	A short name (all with or all without quotes) to identify the effect which should be changed.
parameter	Value of internal effect parameter. If NULL, no change is made.
fix	For fixing effects. Boolean required. Default FALSE.
test	For testing effects by score-type tests. Boolean required. Default FALSE.
random	For specifying that effects will vary randomly; not relevant for RSiena at this moment. Boolean required. Default FALSE.
initialValue	Initial value required. Default 0.
timeDummy	string: Comma delimited string of which periods to dummy. Alternatively, use includeTimeDummy.
include	Boolean. default TRUE, but can be switched to FALSE to turn off an effect.
name	Name of dependent variable (network or behavior) for which effects are being modified. Defaults to the first in the effects object.
type	Character string indicating the type of the effect to be changed : currently "rate", "eval", "endow", or "creation". Default "eval".
interaction1	Name of siena object where needed to completely identify the effect e.g. covariate name or behavior variable name.
interaction2	Name of siena object where needed to completely identify the effect e.g. covariate name or behavior variable name.
effect1	Only for shortName=unspInt, which means this is a user-defined interaction effect: effect1 is a natural number indicating the first component of the interaction effect; the number is the one listed when applying print() to myeff.
effect2	Only for shortName=unspInt: second component of interaction effect (see effect1).
effect3	Only for shortName=unspInt: third component of interaction effect, if any (see effect1).
period	Number of period if basic rate. Use numbering within groups.
group	Number of group if basic rate.
character	Boolean: is the short name a character string or not.

Details

The arguments shortName, name, type, interaction1, interaction2, effect1, effect2, and effect3 should identify the effects completely. The column elements of the output effects object for parm, fix, test, randomEffects, initialValue, and timeDummy will be set to the values requested.

Value

An object of class sienaEffects or sienaGroupEffects. This will be an updated version of the input effects object, with one row updated. Details of the row altered will be printed.

Author(s)

Ruth Ripley

References

See http://www.stats.ox.ac.uk/~snijders/siena/

See Also

getEffects, includeEffects, includeInteraction.

Examples

```
mynet <- sienaDependent(array(c(s501, s502, s503), dim=c(50, 50, 3)))
mybeh <- sienaDependent(s50a, type="behavior")
mydata <- sienaDataCreate(mynet, mybeh)
myeff <- getEffects(mydata)
# Set the initial rate parameter for one period:
myeff <- setEffect(myeff, Rate, initialValue=1.5, name="mybeh",
                   type="rate", period=2)
# Specify an effect parameter:
myeff <- setEffect(myeff, outTrunc, parameter=1)
myeff
# Now request the interaction between reciprocity and alcohol similarity
# to be tested without estimation.
myeff <- includeInteraction(myeff, recip, simX, interaction1=c("", "mybeh"))
myeff
# The following code uses the fact that the interaction effect is number 6.
# In practice one would just look up the numbers of effect1 and effect2
# from the response to \code{myeff} and fill them in.
(eff1 <- myeff[myeff$include,]$effect1[6])
(eff2 <- myeff[myeff$include,]$effect2[6])
myeff <- setEffect(myeff, unspInt, fix=TRUE, test=TRUE,
                                    effect1=eff1, effect2=eff2)
myeff
```

siena07 *Function to estimate parameters in a Siena model*

Description

Estimates parameters in a Siena model using method of moments, based on direct simulation, conditional or otherwise; or using Maximum Likelihood by MCMC simulation. Estimation is done using a Robbins-Monro algorithm. Note that the data and particular model to be used must be passed in using named arguments as the ..., and the specification for the algorithm must be passed on as x, which is a sienaAlgorithm object as produced by sienaAlgorithmCreate (see examples).

Usage

```
siena07(x, batch=FALSE, verbose=FALSE, silent=FALSE,
        useCluster=FALSE, nbrNodes=2, initC=TRUE,
        clusterString=rep("localhost", nbrNodes), tt=NULL,
        parallelTesting=FALSE, clusterIter=!x$maxlike,
        clusterType=c("PSOCK", "FORK"), cl=NULL, ...)
```

Arguments

x	A control object, of class sienaAlgorithm
batch	Desired interface: FALSE gives a gui (graphical user interface implemented as a tcl/tk screen), TRUE gives a small (if verbose=FALSE) amount of printout to the console.
verbose	Produces various output to the console if TRUE.
silent	Produces no output to the console if TRUE, even if batch mode.
useCluster	Boolean: whether to use a cluster of processes (useful if multiple processors are available).
nbrNodes	Number of processes to use if useCluster is TRUE.
initC	Boolean: set to TRUE if the simulation will use C routines (currently always needed). Only for use if using multiple processors, to ensure all copies are initialised correctly. Ignored otherwise, so is set to TRUE by default.
clusterString	Definitions of clusters. Default set up to use the local machine only.
tt	A tcltk toplevel window. Used if called from the model options screen.
parallelTesting	Boolean. If TRUE, sets up random numbers to parallel those in Siena 3.
clusterIter	Boolean. If TRUE, multiple processes execute complete iterations at each call. If FALSE, multiple processes execute a single wave at each call.
clusterType	Either "PSOCK" or "FORK". On Windows, must be "PSOCK". On a single non-Windows machine may be "FORK", and subprocesses will be formed by forking. If "PSOCK", subprocesses are formed using R scripts.

cl	An object of class c("SOCKcluster", "cluster") (see Details).
...	Arguments for the simulation function, see simstats0c: usually, data and effects, as in the examples below; possibly also prevAns if a previous reasonable provisional estimate was obtained for a similar model; possibly also returnDeps if the simulated dependent variables (networks, behaviour) should be returned; possibly also returnChains if the simulated sequences (chains) of ministeps should be returned; this will produce a very big file.

Details

siena07 runs a Robbins-Monro algorithm for parameter estimation according to the Method of Moments using the three-phase implementation in Snijders (2001) and Snijders, Steglich and Schweinberger (2007), with (if findiff=FALSE in the sienaAlgorithm object) derivative estimation as in Schweinberger and Snijders (2007). Phase 1 does a few iterations to estimate the derivative matrix of the targets with respect to the parameter vector. Phase 2 does the estimation. Phase 3 runs a simulation to estimate standard errors and check convergence of the model. The simulation function is called once for each iteration in these phases and also once to initialise the model fitting and once to complete it. Unless in batch mode, displays a tcl/tk screen to allow interruption and to show progress.

If maxlike=TRUE in the sienaAlgorithm object, estimation is done by Maximum Likelihood implemented as in Snijders, Koskinen and Schweinberger (2010) also using the three-phase Robbins-Monro algorithm.

It is necessary to check that convergence has been achieved. The rule of thumb is that the all t-ratios for convergence should be in absolute value less than 0.1 and the overall maximum convergence ratio should be less than 0.25. If this was not achieved, the result can be used to start another estimation run from the estimate obtained, using the parameter prevAns as illustrated in the example below. (This parameter is inserted through '...' into the function initializeFRAN.)

In the case of using multiple processors, there are two options for telling siena07 to use them. By specifying the options useCluster, nbrNodes, clusterString and initC, siena07 will create a cluster object that will be used by the parallel package. After finishing the estimation procedure, siena07 will automatically stop the cluster. Alternatively, instead of having the function to create a cluster, the user may provide its own by specifying the option cl, similar to what the boot function does in the **boot** package. By using the option cl the user may be able to create more complex clusters (see examples below).

Parameters can be tested against zero by dividing the estimate by its standard error and using an approximate standard normal null distribution. Further, functions Wald.RSiena and Multipar.RSiena are available for multi-parameter testing.
Parameters specified in includeEffects or setEffect with fix=TRUE, test=TRUE will not be estimated; score tests of their hypothesized values are reported in the output file specified in the control (algorithm) object. These tests can be obtained also using score.Test .

Value

Returns an object of class sienaFit, some parts of which are:

OK	Boolean indicating successful termination

`termination` Character string, values: "OK", "Error", or "UserInterrupt". "UserInterrupt" indicates that the user asked for early termination before phase 3.

`f` Various characteristics of the data and model definition.

`requestedEffects` The included effects in the effects object.

`effects` The included effects in the effects object to which are added the main effects of the requested interaction effects, if any.

`theta` Estimated value of theta.

`covtheta` Estimated covariance matrix of theta; this is not available if the sienaAlgorithm object `x` was produced with `simOnly=TRUE`.

`se` Vector of standard errors of estimated theta.

`dfra` Matrix of estimated derivatives.

`sf` Matrix of deviations from targets in phase 3.

`sf2` Array of periodwise deviations from simulations in phase 3. Not included if `lessMem=FALSE` in `sienaAlgorithmCreate`.

`tconv` t-statistics for convergence.

`tmax` maximum absolute t-statistic for convergence for non-fixed parameters.

`tconv.max` overall maximum convergence ratio.

`targets` Observed statistics; for ML, zero vector.

`targets2` Observed statistics by wave, starting with second wave; for ML, zero matrix.

`ssc` Score function contributions for each wave for each simulation in phase 3. Not included if finite difference method is used or if `lessMem=FALSE` in `sienaAlgorithmCreate`.

`scores` Score functions, added over waves, for each simulation in phase 3. Only included if `lessMem=FALSE` in `sienaAlgorithmCreate`.

`sims` If `returnDeps=TRUE`: list of simulated dependent variables (networks, behaviour). Networks are given as a list of edgelists, one for each period.

`chain` If `returnChains = TRUE`: list, or data frame, of simulated chains of ministeps. The chain has the structure `chain[[run]][[depvar]][[period]][[ministep]]`.

`Phase3nits` Number of iterations actually performed in phase 3.

Writes text output to the file named "projname.out", where projname is defined in the sienaAlgorithm object `x`.

Author(s)

Ruth Ripley, Tom Snijders

References

- Schweinberger, Michael, and Snijders, Tom A.B. (2007). Markov models for digraph panel data: Monte Carlo-based derivative estimation. *Computational Statistics and Data Analysis* 51, 4465-4483.
- Snijders, Tom A.B. (2001). The statistical evaluation of social network dynamics. *Sociological Methodology*, 31, 361-395.

- Snijders, Tom A.B. (2017). Stochastic Actor-Oriented Models for Network Dynamics. *Annual Review of Statistics and Its Application*, 4, 343-363.
- Snijders, Tom A. B., Koskinen, Johan, and Schweinberger, Michael (2010). Maximum likelihood estimation for social network dynamics. *Annals of Applied Statistics*, 4, 567-588.
- Snijders, Tom A.B., Steglich, Christian E.G., and Schweinberger, Michael (2007). Modeling the co-evolution of networks and behavior. Pp. 41-71 in *Longitudinal models in the behavioral and related sciences*, edited by Kees van Montfort, Han Oud and Albert Satorra; Lawrence Erlbaum.
- Steglich, Christian E. G., Snijders, Tom A. B., and Pearson, Michael A. (2010). Dynamic networks and behavior: Separating selection from influence. Sociological Methodology, 40, 329-393.

- Information about the implementation of the algorithm is in http://www.stats.ox.ac.uk/~snijders/siena/Siena_algorithms.pdf.
- Further see http://www.stats.ox.ac.uk/~snijders/siena/ .

See Also

sienaAlgorithmCreate, Wald.RSiena, Multipar.RSiena, score.Test.

There are print, summary and xtable methods for sienaFit objects: xtable, print.sienaFit.

Examples

```
myalgorithm <- sienaAlgorithmCreate(nsub=2, n3=100, seed=123)
# nsub=2 and n3=100 is used here for having a brief computation, not for practice.
mynet1 <- sienaDependent(array(c(tmp3, tmp4), dim=c(32, 32, 2)))
mydata <- sienaDataCreate(mynet1)
myeff <- getEffects(mydata)
ans <- siena07(myalgorithm, data=mydata, effects=myeff, batch=TRUE)

# or for non-conditional estimation ----------------------------------------------
## Not run:
model <- sienaModelCreate(nsub=2, n3=100, cond=FALSE, seed=123)
ans <- siena07(myalgorithm, data=mydata, effects=myeff, batch=TRUE)

## End(Not run)

# or if a previous "on track" result ans was obtained ---------------------------
## Not run:
ans1 <- siena07(myalgorithm, data=mydata, effects=myeff, prevAns=ans, batch=TRUE)

## End(Not run)

# Running in multiple processors ------------------------------------------------
## Not run:
# Not tested because dependent on presence of processors
# Find out how many processors there are
library(parallel)
(n.clus <- detectCores() - 1)
# number of cores; 1 is subtracted to keep time for other processes
```

```
ans2 <- siena07(myalgorithm, data=mydata, effects=myeff, batch=TRUE,
                useCluster=TRUE, nbrNodes=n.clus, initC=TRUE)

# Suppose 8 processors are going to be used.
# Loading the parallel package and creating a cluster
# with 8 processors (this should be equivalent)

library(parallel)
cl <- makeCluster(n.clus)

ans3 <- siena07(myalgorithm, data=mydata, effects=myeff, batch=TRUE, cl = cl)

# Notice that now -siena07- perhaps won't stop the cluster for you.
# stopCluster(cl)

# You can create even more complex clusters using several computers. In this
# example we are creating a cluster with 3*8 = 24 processors on three
# different machines.
cl <- makePSOCKcluster(
    rep(c('localhost', 'machine2.website.com' , 'machine3.website.com'), 8),
    user='myusername', rshcmd='ssh -p PORTNUMBER')

ans4 <- siena07(myalgorithm, data=mydata, effects=myeff, batch=TRUE, cl = cl)
stopCluster(cl)

## End(Not run)
# Accessing simulated networks for ML ------------------------------------------
# The following is an example for accessing the simulated networks for ML,
# which makes sense only if there are some missing tie variables;
# observed tie variables are identically simulated at the moment of observation,
# missing tie variable are imputed in a model-based way.
mat1 <- matrix(c(0,0,1,1,
                 1,0,0,0,
                 0,0,0,1,
                 0,1,0,0),4,4, byrow=TRUE)
mat2 <- matrix(c(0,1,1,1,
                 1,0,0,0,
                 0,0,0,1,
                 0,0,1,0),4,4, byrow=TRUE)
mat3 <- matrix(c(0,1,0,1,
                 1,0,0,0,
                 0,0,0,0,
                 NA,1,1,0),4,4, byrow=TRUE)
mats <- array(c(mat1,mat2,mat3), dim=c(4,4,3))
net <- sienaDependent(mats, allowOnly=FALSE)
sdat <- sienaDataCreate(net)
alg <- sienaAlgorithmCreate(maxlike=TRUE, nsub=3, n3=100, seed=1234)
effs <- getEffects(sdat)
## Not run:
(ans <- siena07(alg, data=sdat, effects=effs, returnDeps=TRUE, batch=TRUE))
# See manual Section 9.1 for information about the following functions
edges.to.adj <- function(x,n){
# create empty adjacency matrix
```

```
    adj <- matrix(0, n, n)
# put edge values in desired places
    adj[x[, 1:2]] <- x[, 3]
    adj
}
the.edge <- function(x,n,h,k){
    edges.to.adj(x,n)[h,k]
}
# Now show the results
n <- 4
ego <- rep.int(1:n,n)
alter <- rep(1:n, each=n)
ones <- sapply(1:n^2, function(i)
    {mean(sapply(ans$sims, function(x){the.edge(x[[1]][[2]],n,ego[i],alter[i])}))})
cbind(ego,alter,ones)
matrix(ones,n,n)

## End(Not run)
```

`siena08` *Function to perform a meta analysis of a collection of Siena fits.*

Description

Estimates a meta analysis based on a collection of Siena fits.

Usage

```
siena08(..., projname = "sienaMeta", bound = 5, alpha = 0.05, maxit=20)
```

Arguments

`...`	names of `sienaFit` objects, returned from `siena07`. They will be renamed if entered in format `newname=oldname`. It is also allowed to give for ... a list of `sienaFit` objects.
`projname`	Base name of report file if required
`bound`	Upper limit of standard error for inclusion in the meta analysis.
`alpha`	1 minus confidence level of confidence intervals.
`maxit`	Number of iterations of iterated least squares procedure.

Details

A meta analysis is performed as described in the Siena manual, section "Meta-analysis of Siena results". This consists of three parts: an iterated weighted least squares modification of the method described in the reference below; maximum likelihood estimates and confidence intervals based on profile likelihoods under normality assumptions; and Fisher combinations of left-sided and right-sided p-values. These are produced for all effects separately.

Note that the corresponding effects must have the same effect name in each model fit. This implies that at least covariates and behavior variables must have the same name in each model fit.

Value

An object of class sienaMeta. There are print, summary and plot methods for this class. This object contains at least the following.

thetadf	Data frame containing the coefficients, standard errors and score test results
projname	Name for any output file to be produced by the print method
bound	Estimates with standard error above this value were excluded from the calculations
scores	Object of class by indicating, for each effect in the models, whether score test information was present.
requestedEffects	The requestedEffects component of the first sienaFit object in ....
theta	The vector of ML estimates mu.ml (see below).
se	The vector of standard errors of the ML estimates mu.ml.se (see below).

Then for each effect, there is a list with at least the following.

cor.est	Spearman rank correlation coefficient between estimates and their standard errors.
cor.pval	p-value for above
regfit	Part of the result of the fit of iwlsm.
regsummary	The summary of the fit, which includes the coefficient table.
Tsq	test statistic for effect zero in every model
pTsq	p-value for above
tratio	test statistics that mean effect is 0
ptratio	p-value for above
Qstat	Test statistic for variance of effects is zero
pttilde	p-value for above
cjplus	Test statistic for at least one theta strictly greater than 0
cjminus	Test statistic for at least one theta strictly less than 0
cjplusp	p-value for cjplus
cjminusp	p-value for cjminus
mu.ml	ML estimate of population mean
mu.ml.se	standard error of ML estimate of population mean
sigma.ml	ML estimate of population standard deviation
mu.confint	confidence interval for population mean based on profile likelihood
sigma.confint	confidence interval for population standard deviation based on profile likelihood
n1	Number of fits on which the meta analysis is based
cjplus	Test statistic for combination of right one-sided Fisher combination tests
cjminus	Test statistic for combination of left one-sided Fisher combination tests

cjplusp	p-value for cjplus
cjminusp	p-value for cjminus
scoreplus	Test statistic for combination of right one-sided p-values from score tests
scoreminus	Test statistic for combination of left one-sided p-values from score tests
scoreplusp	p-value for scoreplus
scoreminusp	p-value for scoreminus
ns	Number of fits on which the score test analysis is based

Author(s)

Ruth Ripley, Tom Snijders

References

T. A. B. Snijders and Chris Baerveldt. Multilevel network study of the effects of delinquent behavior on friendship evolution. *Journal of Mathematical Sociology*, 27: 123-151, 2003.

See also the manual (Section 11.2) and http://www.stats.ox.ac.uk/~snijders/siena/

See Also

sienaMeta, iwlsm, siena07

Examples

```
## Not run:
# A meta-analysis for three groups does not make much sense
# for generalizing to a population of networks,
# but the Fisher combinations of p-values are meaningful.
# However, using three groups does show the idea.

Group1 <- sienaDependent(array(c(N3401, HN3401), dim=c(45, 45, 2)))
Group3 <- sienaDependent(array(c(N3403, HN3403), dim=c(37, 37, 2)))
Group4 <- sienaDependent(array(c(N3404, HN3404), dim=c(33, 33, 2)))
dataset.1 <- sienaDataCreate(Friends = Group1)
dataset.3 <- sienaDataCreate(Friends = Group3)
dataset.4 <- sienaDataCreate(Friends = Group4)
OneAlgorithm <- sienaAlgorithmCreate(projname = "SingleGroups", seed=1234)
effects.1 <- getEffects(dataset.1)
effects.3 <- getEffects(dataset.3)
effects.4 <- getEffects(dataset.4)
effects.1 <- includeEffects(effects.1, transTrip)
effects.1 <- setEffect(effects.1, cycle3, fix=TRUE, test=TRUE)
effects.3 <- includeEffects(effects.3, transTrip)
effects.3 <- setEffect(effects.3, cycle3, fix=TRUE, test=TRUE)
effects.4 <- includeEffects(effects.4, transTrip)
effects.4 <- setEffect(effects.4, cycle3, fix=TRUE, test=TRUE)
ans.1 <- siena07(OneAlgorithm, data=dataset.1, effects=effects.1, batch=TRUE)
ans.3 <- siena07(OneAlgorithm, data=dataset.3, effects=effects.3, batch=TRUE)
ans.4 <- siena07(OneAlgorithm, data=dataset.4, effects=effects.4, batch=TRUE)
```

```
ans.1
ans.3
ans.4
(meta <- siena08(ans.1, ans.3, ans.4))
# For specifically presenting the Fisher combinations:
# First determine the components of meta with estimated effects:
which.est <- sapply(meta, function(x){ifelse(is.list(x),!is.null(x$cjplus),FALSE)})
Fishers <- t(sapply(1:sum(which.est),
        function(i){c(meta[[i]]$cjplus, meta[[i]]$cjminus,
                      meta[[i]]$cjplusp, meta[[i]]$cjminusp, 2*meta[[i]]$n1 )}))
Fishers <- as.data.frame(Fishers, row.names=names(meta)[which.est])
names(Fishers) <- c('Fplus', 'Fminus', 'pplus', 'pminus', 'df')
Fishers
round(Fishers,4)

## End(Not run)
```

`sienaAlgorithmCreate` *Function to create an object containing the algorithm specifications for parameter estimation in RSiena*

Description

Creates an object with specifications for the algorithm for parameter estimation in RSiena. `sienaAlgorithmCreate()` and `sienaModelCreate()` are identical functions; the second name was used from the start of the `RSiena` package, but the first name indicates more precisely the purpose of this function.

Usage

```
sienaAlgorithmCreate(fn, projname = "Siena", MaxDegree = NULL, Offset = NULL,
     useStdInits = FALSE, n3 = 1000, nsub = 4, n2start = NULL,
     dolby=TRUE, maxlike = FALSE, diagonalize=0.2*!maxlike,
     condvarno = 0, condname = "", firstg = 0.2, reduceg = 0.5,
     cond = NA, findiff = FALSE, seed = NULL,
     pridg=0.05, prcdg=0.05, prper=0.2, pripr=0.3, prdpr=0.3,
     prirms=0.05, prdrms=0.05, maximumPermutationLength=40,
     minimumPermutationLength=2, initialPermutationLength=20,
     modelType=NULL, behModelType=NULL, mult=5, simOnly=FALSE, localML=FALSE,
     truncation=5, doubleAveraging=0, standardizeVar=(diagonalize<1),
     lessMem=FALSE)

sienaModelCreate(fn, projname = "Siena", MaxDegree = NULL, Offset = NULL,
     useStdInits = FALSE, n3 = 1000, nsub = 4, n2start = NULL,
     dolby=TRUE, maxlike = FALSE, diagonalize=0.2*!maxlike,
     condvarno = 0, condname = "", firstg = 0.2, reduceg = 0.5,
     cond = NA, findiff = FALSE, seed = NULL,
     pridg=0.05, prcdg=0.05, prper=0.2, pripr=0.3, prdpr=0.3,
```

```
prirms=0.05, prdrms=0.05, maximumPermutationLength=40,
minimumPermutationLength=2, initialPermutationLength=20,
modelType=NULL, behModelType=NULL, mult=5, simOnly=FALSE, localML=FALSE,
truncation=5, doubleAveraging=0, standardizeVar=(diagonalize<1),
lessMem=FALSE)
```

Arguments

`fn`	Function to do one simulation in the Robbins-Monro algorithm. Not to be touched.
`projname`	Character string name of project; the output file will be called projname.out. No embedded spaces!!!
`MaxDegree`	Named vector of maximum degree values for corresponding networks. Allows to restrict the model to networks with degrees not higher than this maximum. Names should be the names of all dependent network variables, in the same order as in the Siena data set. Default as well as value 0 imply no restrictions.
`Offset`	Named vector of offset values for symmetric networks with `modelType = 3` (M.1). Names should be the names of all dependent network variables, in the same order as in the Siena data set. Default `NULL` implies values 0.
`useStdInits`	Boolean. If TRUE, the initial values in the effects object will be ignored and default values used instead. If FALSE, the initial values in the effects object will be used.
`n3`	Number of iterations in phase 3. For regular use with the Method of Moments, n3=1000 mostly suffices. For use in publications and for Maximum Likelihood, at least n3=3000 is advised. Sometimes much higher values are required for stable estimation of standard errors.
`nsub`	Number of subphases in phase 2.
`n2start`	Minimum number of interations in subphase 1 of phase 2; default is `2.52*(p+7)`, where `p` = number of estimated parameters.
`dolby`	Boolean. Should there be noise reduction by regression on augmented data score. In most cases dolby=TRUE yields better convergence, but takes some extra computing time; if convergence is problematic, however, dolby=FALSE may be tried. Just use whatever works best.
`maxlike`	Whether to use maximum likelihood method or Method of Moments estimation.
`diagonalize`	Number between 0 and 1 (bounds included), values outside this interval will be truncated; for diagonalize=0 the complete estimated derivative matrix will be used for updates in the Robbins-Monro procedure; for diagonalize=1 only the diagonal entries will be used; for values between 0 and 1, the weighted average will be used with weight diagonalize for the diagonalized matrix. Has no effect for ML estimation. Higher values are more stable, lower values potentially more efficient. Default: for ML estimation, diagonalize=0; for MoM estimation, diagonalize = 1.0.
`condvarno`	If `cond` (conditional simulation), the sequential number of the network or behavior variable on which to condition.

`condname` If conditional, the name of the dependent variable on which to condition. Use one or other of `condname` or `condvarno` to specify the variable.

`firstg` Initial value of scaling ("gain") parameter for updates in the Robbins-Monro procedure.

`reduceg` Reduction factor for scaling ("gain") parameter for updates in the Robbins-Monro procedure (MoM only).

`cond` Boolean. Only relevant for Method of Moments simulation/estimation. If TRUE, use conditional simulation; if FALSE, unconditional simulation. If missing, decision is deferred until `siena07`, when it is set to TRUE if there is only one dependent variable, FALSE otherwise.

`findiff` Boolean: If TRUE, estimate derivatives using finite differences. If FALSE, use scores.

`seed` Integer. Starting value of random seed. Not used if parallel testing.

`pridg` Real number. Probability used in Metropolis-Hastings routine in ML estimation. See Siena_Algorithms.pdf.

`prcdg` Real number. Probability used in Metropolis-Hastings routine in ML estimation. See Siena_Algorithms.pdf.

`prper` Real number. Probability used in Metropolis-Hastings routine in ML estimation. See Siena_Algorithms.pdf.

`pripr` Real number. Probability used in Metropolis-Hastings routine in ML estimation. See Siena_Algorithms.pdf.

`prdpr` Real number. Probability used in Metropolis-Hastings routine in ML estimation. See Siena_Algorithms.pdf.

`prirms` Real number. Probability used in Metropolis-Hastings routine in ML estimation. See Siena_Algorithms.pdf.

`prdrms` Real number. Probability used in Metropolis-Hastings routine in ML estimation. See Siena_Algorithms.pdf.

`maximumPermutationLength`
Maximum length of permutation in steps in ML estimation.

`minimumPermutationLength`
Minimum length of permutation in steps in ML estimation.

`initialPermutationLength`
Initial length of permutation in steps in ML estimation.

`modelType` Named vector indicating the type of model to be fitted for dependent network variables. Possible values are:
1=directed, 2:6 for symmetric networks only: 2=dictatorial forcing (D.1), 3=Initiative model with reciprocal confirmation (M.1), 4=Pairwise dictatorial forcing model (D.2), 5=Pairwise mutual model (M.2), 6=Pairwise joint model (C.2).
Names should be the names of all dependent network variables, in the same order as in the Siena data set.
See Snijders and Pickup (2016) for the meanings of these models.
Default `NULL` implies 1 for directed or two-mode, 2 for symmetric.

`behModelType` Named vector indicating the type of model to be fitted for behavioral dependent variables. Possible values are:
1=standard (restricted), 2=absorbing.
Names should be the names of all dependent behavioral variables, in the same order as in the Siena data set.
Default `NULL` implies values 1.

`mult` Multiplication factor for maximum likelihood and Bayes. Number of steps per iteration is set to this multiple of the total distance between the observations at start and finish of the wave. Decreasing `mult` below a certain value has no further effect.

`simOnly` Logical: If TRUE, then the calculation of the covariance matrix and standard errors of the estimates at the end of Phase 3 of the estimation algorithm in function siena07 is skipped. This is suitable if nsub=0 and `siena07` is used only for the purpose of simulation.

`localML` Logical: If TRUE, and `maxlike`, then calculations are sped up for models with all local effects.

`truncation` Used for step truncation in the Robbins Monro algorithm (applied to deviate/(standard deviation)).

`doubleAveraging`
subphase after which double averaging is used in the Robbins Monro algorithm, which probably increases algorithm efficiency.

`standardizeVar` Logical: whether to limit deviations used in Robbins-Monro updates to unit variances.

`lessMem` Logical: whether to reduce storage during operation of `siena07`, and of the object produced, by leaving out arrays by iteration and by period of simulated statistics `sf2` and scores `ssc`.
if `lessMem=TRUE`, it will be impossible to run `sienaTimeTest` or `sienaGOF` on the object produced by `siena07`.

Details

Model specification is done via this object for `siena07`. This function creates an object with the elements required to control the Robbins-Monro algorithm. Those not available as arguments can be changed manually where desired.
Further information about the implementation of the algorithm is in
http://www.stats.ox.ac.uk/~snijders/siena/Siena_algorithms.pdf.

Value

Returns an object of class `sienaAlgorithm` containing values implied by the parameters.

Author(s)

Ruth Ripley and Tom A.B. Snijders

References

- For the model types:
  Tom A. B. Snijders and Mark Pickup, Stochastic Actor-Oriented Models for Network Dynamics. In: Jennifer N. Victor, Mark Lubell and Alexander H. Montgomery, Oxford Handbook of Political Networks. Oxford University Press, 2016.
- Also see http://www.stats.ox.ac.uk/~snijders/siena/

See Also

siena07, simstats0c.

Examples

```
myAlgorithm <- sienaAlgorithmCreate(projname="NetworkDyn")
StdAlgorithm <- sienaAlgorithmCreate(projname="NetworkDyn", useStdInits=TRUE)
CondAlgorithm <- sienaAlgorithmCreate(projname="NetworkDyn", condvarno=1, cond=TRUE)
Max10Algorithm <- sienaAlgorithmCreate(projname="NetworkDyn", MaxDegree=c(mynet=10),
     behModelType=c(mynet=1))
Beh2Algorithm <- sienaAlgorithmCreate(projname="NetBehDyn", behModelType=c(mybeh=2))
# where mynet is the name of the network object created by sienaDependent(),
# and mybeh the name of the behavior object created by the same function.
```

---

sienaCompositionChange

Functions to create a Siena composition change object

Description

Used to create a list of events describing the changes over time of a Siena actor set

Usage

```
sienaCompositionChange(changelist, nodeSet = "Actors", option = 1)
sienaCompositionChangeFromFile(filename, nodeSet = "Actors",
    fileobj=NULL, option = 1)
```

Arguments

changelist	A list with an entry for each actor in the node set. Each entry a vector of numbers (may be as characters) indicating intervals during which the corresponding actor was present. Each entry must have an even number of digits. The actor is assumed to be present from the first to the second, third to fourth, etc., time points.
filename	Name of file containing change information. One line per actor, each line a series of space delimited numbers indicating intervals.
fileobj	The result of readLines on filename.

`nodeSet` Character string containing the name of a Siena node set. If the entire data set contains more than one node set, then the node sets must be specified in all data objects.

`option` Integer controlling the processing of the tie variables for the actors not currently present. Values (default is 1)

1. 0 before entry, final value carried forward after leaving, and used for calculating statistics in Method of Moments estimation
2. 0 before entry, missing after (final value carried forward, but treated as missing)
3. missing whenever not in the network. Previous values will be used where available, but always treated as missing values.
4. Convert to structural zeros (not available at present).

Details

If there is a composition change object for the first node set in the data object, then this will be used in estimation by the Method of Moments to make actors active (able to send and receive ties) only for the time intervals when this is indicated in the composition change object. This is done according to the procedure of Huisman and Snijders (2003). See the manual for further details.
For bipartite networks, composition change objects for the second node set have no effect and will lead to an error message.
For `M` waves, time starts at 1 and ends at `M`; so all numbers must be between 1 and the number of waves (bounds included). Intervals are treated as closed at each end. For example, an entry `(2, 4)` means that the actor corresponding to this entry arrived at wave 2 and left at wave 4, but did give valid date for both of these waves. An entry `(1.01, 2.99)` means that the actor arrived just after wave 1 and left just before wave 3, and gave valid data only for wave 2. An entry `(1, 2), (3.5, 4)` means that the actor was there at the start and left at wave 2 (giving valid data for wave 2), came back halfway between waves 3 and 4, and gave valid data still at wave 4; if there would be more than 4 waves in the data set, this entry would also mean that the actor left at wave 4.
For data sets including a composition change object, estimation by Method of Moments is forced to be unconditional, overriding the specification in the `sienaAlgorithm` object.

Value

An object of class "compositionChange", a list of numeric vectors, with attributes:

`NodeSet` Name of node set

`Option` Option

Author(s)

Ruth Ripley

References

- Huisman, M. E. and Snijders, T. A. B. (2003). Statistical analysis of longitudinal network data with changing composition. *Sociological Methods & Research*, 32, 253-287.
- The manual: http://www.stats.ox.ac.uk/~snijders/siena/RSiena_Manual.pdf
- Further see http://www.stats.ox.ac.uk/~snijders/siena/

See Also

sienaNodeSet, sienaDataCreate

Examples

```
  clist <- list(c(1, 3), c(1.4, 2.5))
  #or
  clist <- list(c("1", "3"), c("1.4", "2.5"))

  compChange <- sienaCompositionChange(clist)

  s50net <- sienaDependent(array(c(s501, s502, s503), dim=c(50, 50, 3)))
  s50list <- rep(list(c(1,3)), 50)
  # This is a trivial composition change: all actors are present in all waves.
  compChange <- sienaCompositionChange(s50list)
  s50data <- sienaDataCreate(s50net, compChange)
  s50data

  ## Not run:
  filedata <- c("1 3", "1.4 2.5")
  write.table(filedata, "cc.dat",row.names=FALSE, col.names=FALSE,
          quote=FALSE)
  ## file will be
  ## 1 3
  ## 1.4 2.5
  compChange <- sienaCompositionChangeFromFile("cc.dat")

## End(Not run)
```

---

sienaDataConstraint *Function to change the values of the constraints between networks.*

Description

This function allows the user to change the constraints of "higher", "disjoint" and "atLeastOne" for a specified pair of networks in a Siena data object.

Usage

```
sienaDataConstraint(x, net1, net2,
          type = c("higher", "disjoint", "atLeastOne"), value = FALSE)
```

Arguments

x	Siena Data Object; maybe a group object?
net1	name of first network
net2	name of second network
type	one of "higher", "disjoint", "atleastOne". Default is "higher".
value	Boolean giving the value.

Details

The value of the appropriate attribute is set to the value requested.

Value

Updated Siena data object.

Author(s)

Ruth Ripley

References

See http://www.stats.ox.ac.uk/~snijders/siena/

See Also

sienaDataCreate

Examples

```
nowFriends <- sienaDependent(array(c(s501, s502, s503), dim=c(50, 50, 3)))
ever <- array(c(s501, s502, s503), dim=c(50, 50, 3))
ever[,,2] <- pmax(ever[,,1], ever[,,2])
ever[,,3] <- pmax(ever[,,2], ever[,,3])
everFriends <- sienaDependent(ever)
# Note: this data set serves to illustrate this function,
# but it is not an appropriate data set for estimation by siena07,
# because everFriends (for the three waves together) depends deterministically
# on nowFriends (for the three waves together).
nowOrEver <- sienaDataCreate(nowFriends, everFriends)
attr(nowOrEver, "higher")
nowOrEver
nowOrEver.unconstrained <-
   sienaDataConstraint(nowOrEver, everFriends, nowFriends, "higher", FALSE)
nowOrEver.unconstrained
attr(nowOrEver.unconstrained, "higher")
```

---

sienaDataCreate *Function to create a Siena data object*

Description

Creates a Siena data object from input dependent variables (networks and possibly behavioural variables), covariates, and composition change objects.

Usage

```
sienaDataCreate(..., nodeSets=NULL, getDocumentation=FALSE)
```

Arguments

...	objects of class sienaDependent, coCovar,varCovar,coDyadCovar, varDyadCovar, sienaCompositionChange
nodeSets	list of Siena node sets. Default is the single node set named "Actors", length equal to the number of rows in the first object of class "sienaDependent". If the entire data set contains more than one node set, then the node sets must have been specified in the creation of all data objects.
getDocumentation	Flag to allow documentation of internal functions, not for use by users.

Details

The function checks that the objects fit, that there is at least one network, and adds various attributes to each dependent variable describing the data. If there is more than one nodeSet they must all be specified. Function print01Report will give a basic description of the data object and is a check useful, e.g., for diagnosing problems.

Value

An object of class "siena" which is designed to be used in a siena model fit. The components of the object are.

nodeSets	List of node sets involved
observations	Integer indicating number of waves of data
depvars	List of networks and behavior variables
cCovars	List of constant covariates
vCovars	List of changing covariates
dycCovars	List of constant dyadic covariates
dyvCovars	List of changing dyadic covariates
compositionChange	List of composition change objects corresponding to the node sets

Author(s)

Ruth Ripley

References

See http://www.stats.ox.ac.uk/~snijders/siena/

See Also

sienaDependent, coCovar, varCovar, coDyadCovar, varDyadCovar, sienaCompositionChange, sienaGroupCreate, sienaDataConstraint, sienaNodeSet, print01Report

Examples

```
mynet <- sienaDependent(array(c(s501, s502, s503), dim=c(50, 50, 3)))
mybeh <- sienaDependent(s50a, type="behavior")
mydata <- sienaDataCreate(mynet, mybeh)
## And for a two-mode network
mynet1 <- sienaDependent(array(c(s501, s502), dim=c(50, 50, 2)), nodeSet="senders")
senders <- sienaNodeSet(50, nodeSetName="senders")
receivers <- sienaNodeSet(30, nodeSetName="receivers")
mynet2 <- sienaDependent(array(c(s501[,1:30], s502[,1:30]), dim=c(50, 30, 2)),
      nodeSet=c("senders", "receivers"))
(mydata <- sienaDataCreate(mynet1, mynet2, nodeSets=list(senders, receivers)))
## Not run:
print01Report(mydata, modelname = "mydescription")

## End(Not run)
```

`sienaDependent` *Function to create a dependent variable for a Siena model*

Description

Creates a Siena dependent variable: either a network, created from a matrix or array or list of sparse matrix of triples; or a behavior variable, created from a matrix.
`sienaDependent()` and `sienaNet()` are identical functions; the second name was used from the start of the `RSiena` package, but the first name indicates more precisely the purpose of this function.

Usage

```
sienaDependent(netarray, type=c("oneMode", "bipartite", "behavior"),
nodeSet="Actors", sparse=is.list(netarray), allowOnly=TRUE)

sienaNet(netarray, type=c("oneMode", "bipartite", "behavior"),
nodeSet="Actors", sparse=is.list(netarray), allowOnly=TRUE)
```

Arguments

`netarray`	matrix (type="behavior" only) or (for the other types) array of values or list of sparse matrices of type "dgTMatrix".
`type`	type of dependent variable, default "oneMode".
`nodeSet`	character string naming the appropriate node set. For a bipartite network, a vector containing 2 character strings: "rows" first, then "columns".
`sparse`	logical: TRUE indicates the data is in sparse matrix format, FALSE otherwise.
`allowOnly`	logical: If TRUE, it will be detected when between any two consecutive waves the changes are non-decreasing or non-increasing, and if this is the case, this will also be a constraint for the simulations between these two waves. This is done by means of the internal parameters `uponly` and `downonly`. If FALSE,

the parameters uponly and downonly always are set to FALSE, and changes in dependent variables will not be constrained to be non-decreasing or non-increasing. This also will imply that some effects are excluded because they are superfluous in such constrained situations. This will be reported in the output of print01Report.

For normal operation when this is the case for all periods, usually TRUE is the appropriate option. When it is only the case for some of the periods, and for data sets that will be part of a multi-group object created by sienaGroupCreate, FALSE usually is preferable.

Details

Adds attributes so that the array or list of matrices can be used in a Siena model fit.

Value

An object of class "sienaDependent". An array or (networks only) a list of sparse matrices with attributes:

netdims	Dimensions of the network or behavior variable: senders, receivers (1 for behavior), periods
type	oneMode, bipartite or behavior
sparse	Boolean: whether the network is given as a list of sparse matrices or not
nodeSet	Character string with name(s) of node set(s)
allowOnly	The value of the allowOnly parameter

Author(s)

Ruth Ripley and Tom A.B. Snijders

References

See http://www.stats.ox.ac.uk/~snijders/siena/ .

See Also

sienaDataCreate, sienaNodeSet, sienaDataConstraint

Examples

```
mynet1 <- sienaDependent(array(c(s501, s502, s503), dim=c(50, 50, 3)))
mybeh <- sienaDependent(s50a, type="behavior")
## note that the following example works although the node sets do not yet exist!
mynet3 <- sienaDependent(array(c(s501, s502, s503), dim=c(50, 50, 3)),
       type="bipartite", nodeSet=c("senders", "receivers"))
## sparse matrix input - create some RSiena edgelists first
library(Matrix)
tmps501 <- as(Matrix(s501), "dgTMatrix")
tmps502 <- as(Matrix(s502), "dgTMatrix")
tmps503 <- as(Matrix(s503), "dgTMatrix")
```

```
mymat1 <- cbind(tmps501@i + 1, tmps501@j + 1, 1, 1)
mymat2 <- cbind(tmps502@i + 1, tmps502@j + 1, 1, 2)
mymat3 <- cbind(tmps503@i + 1, tmps503@j + 1, 1, 3)
mymat <- rbind(mymat1, mymat2, mymat3)
library(Matrix)
## mymat includes all 3 waves
mymatlist <- by(mymat, mymat[, 4], function(x)
    spMatrix(50, 50, x[, 1], x[, 2], x[, 3]))
mynet4 <- sienaDependent(mymatlist)
## or alternatively
mymat1 <- mymat[mymat[, 4] == 1, ]
mymat2 <- mymat[mymat[, 4] == 2, ]
mymat3 <- mymat[mymat[, 4] == 3, ]
mymat1s <- spMatrix(50, 50, mymat1[, 1], mymat1[, 2], mymat1[, 3])
mymat2s <- spMatrix(50, 50, mymat2[, 1], mymat2[, 2], mymat2[, 3])
mymat3s <- spMatrix(50, 50, mymat3[, 1], mymat3[, 2], mymat3[, 3])
mynet4 <- sienaDependent(list(mymat1s, mymat2s, mymat3s))
```

`sienaFit.methods` *Methods for processing sienaFit objects, produced by* `siena07`.

Description

`print`, `summary`, and `xtable` methods for `sienaFit` objects.

Usage

```
## S3 method for class 'sienaFit'
print(x, tstat=TRUE, ...)

## S3 method for class 'sienaFit'
summary(object, ...)

## S3 method for class 'summary.sienaFit'
print(x, matrices=TRUE, ...)

## S3 method for class 'sienaFit'
xtable(x, caption = NULL, label = NULL, align = NULL,
                digits = NULL, display = NULL, ...)

siena.table(x, type="tex", file=paste(deparse(substitute(x)), ".", type,sep=""),
            vertLine=TRUE, tstatPrint=FALSE, sig=FALSE, d=3)
```

Arguments

`object`	An object of class `sienaFit`, produced by `siena07`.
`x`	An object of class `sienaFit`, or `summary.sienaFit` as appropriate.

matrices	Boolean: whether also to print in the summary the covariance matrix of the estimates, the derivative matrix of expected statistics X by parameters, and the covariance matrix of the statistics.
tstat	Boolean: if this is NULL, the t-statistics for convergence will not be added to the report.
type	Type of output to produce; must be either "tex" or "html".
file	Name of the file; defaults to the name of the sienaFit object.
vertLine	Boolean: add vertical lines separating the columns in siena.table.
tstatPrint	Boolean: add a column of significance t values (parameter estimate/standard error estimate) to siena.table.
sig	Boolean: adds symbols (daggers and asterisks) indicating significance levels for the parameter estimates to siena.table.
d	The number of decimals places used in siena.table.
caption	See documentation for xtable.
label	See documentation for xtable.
align	See documentation for xtable.
digits	See documentation for xtable.
display	See documentation for xtable
...	Add extra parameters for print.xtable here. e.g. type, file.

Value

The function print.sienaFit prints a table containing estimated parameter values, standard errors and (optionally) t-statistics for convergence.

The function summary.sienaFit prints a table containing estimated parameter values, standard errors and t-statistics for convergence together with the covariance matrix of the estimates, the derivative matrix of expected statistics X by parameters, and the covariance matrix of the expected statistics X.

The function xtable.sienaFit creates an object of class xtable.sienaFit which inherits from class xtable and passes an extra arguments to the print.xtable.

The function siena.table outputs a latex or html table of the estimates and standards errors of a sienaFit object. The table will be written to a file in the current directory and has a footnote reporting the maximum of the convergence t-ratios.

See the manual for how to import these tables easily into MS-Word.

Author(s)

Ruth Ripley, Charlotte Greenan

References

See http://www.stats.ox.ac.uk/~snijders/siena/

See Also

xtable, print.xtable, siena07

Examples

```
myalgorithm <- sienaAlgorithmCreate(nsub=2, n3=100)
mynet1 <- sienaDependent(array(c(tmp3, tmp4), dim=c(32, 32, 2)))
mydata <- sienaDataCreate(mynet1)
myeff <- getEffects(mydata)
ans <- siena07(myalgorithm, data=mydata, effects=myeff, batch=TRUE)
ans
summary(ans)
## Not run:
xtable(ans, type="html", file="ans.html")
siena.table(ans, type="html", tstat=TRUE, d=2)

## End(Not run)
```

---

sienaGOF *Functions to assess goodness of fit for SAOMs*

Description

The function sienaGOF assesses goodness of fit for a model specification as represented by an estimated sienaFit object created by siena07. This is done by simulations of auxiliary statistics, that differ from the statistics used for estimating the parameters. The auxiliary statistics must be given explicitly. The fit is good if the average values of the auxiliary statistics over many simulation runs are close to the values observed in the data. A Monte Carlo test based on the Mahalanobis distance is used to calculate frequentist p-values. Plotting functions can be used to diagnose bad fit. There are basic functions for calculating auxiliary statistics available out of the box, and the user is also permitted to create custom functions.

Usage

```
sienaGOF(sienaFitObject, auxiliaryFunction,
        period=NULL, verbose=FALSE, join=TRUE, twoTailed=FALSE,
        cluster=NULL, robust=FALSE, groupName="Data1",
        varName, ...)
## S3 method for class 'sienaGOF'
plot(x, center=FALSE, scale=FALSE, violin=TRUE, key=NULL,
        perc=.05, period=1, ...)
descriptives.sienaGOF(x, center=FALSE, scale=FALSE, perc=.05, key=NULL,
        period=1, showAll=FALSE)
```

Arguments

sienaFitObject	Results of class sienaFit, produced by a call to siena07 with returnDeps = TRUE.
auxiliaryFunction	Function to be used to calculate the auxiliary statistics; this can be a user-defined function, e.g. depending on the sna or igraph packages. See Examples and sienaGOF-auxiliary for more information on the signature of this function. The basic signature is function(index, data, sims, period, groupName, varName, ...), where index is the index of the simulated network, or NULL if the observed variable is needed; data is the observed data object from which the relevant variables are extracted; sims is the list of simulations returned from siena07; period is the index of the period; and ... are further arguments (like levls in the examples below and in sienaGOF-auxiliary).
period	Vector of period(s) to be used (may run from 1 to number of waves - 1). Has an effect only if join=FALSE.
verbose	Whether to print intermediate results. This may give some peace of mind to the user because calculations can take some time.
join	Boolean: should sienaGOF do tests on all of the periods individually (FALSE), or sum across periods (TRUE)?
twoTailed	Whether to use two tails for calculating p-values on the Monte Carlo test. Recommended for advanced users only, as it is probably only applicable in rare cases.
cluster	Optionally, a parallel or snow cluster to execute the auxiliary function calculations on.
robust	Whether to use robust estimation of the covariance matrix.
groupName	Name of group; relevant for multi-group data sets.
varName	Name of dependent variable.
x	Result from a call to sienaGOF.
center	Whether to center the statistics by median during plotting.
scale	Whether to scale the statistics by range during plotting. scale=TRUE makes little sense without also center=TRUE.
violin	Use violin plots (vs. box plots only)?
key	Keys in the plot for the levels of the auxiliary statistic (as given by parameter levls in the examples).
perc	1 minus confidence level for the confidence bands (two sided).
...	Other arguments; e.g. (1), levls for sienaGOF() where levls is a parameter for the auxiliary statistic as in sienaGOF-auxiliary; e.g. (2), main, xlab and/or ylab for plot.sienaGOF().
showAll	If FALSE, drops statistics with variance 0, like in the plot.

Details

This function is used to assess the goodness of fit of an estimated stochastic actor-oriented model for an arbitrarily defined multidimensional auxiliary statistic. It operates basically by comparing the observed values, at the ends of the periods, with the simulated values for the ends of the periods. The differences are assessed by combining the components of the auxiliary statistic using the Mahalanobis distance.

The function does not work properly for data sets that include a `sienaCompositionChange` object. If you wish to test the fit for such a data set, you need (for the purpose of fit assessment only) to replace the data set by a data set where absent actors are represented by structural zeros, and estimate the same model for this data set with the corresponding effects object, and use `sienaGOF` for this `sienaFit` object.

To achieve comparability between simulated and observed dependent variables, variables that are missing in the data at the start or end of a period are replaced by 0 (for tie variables) or NA (for behavior variables).
If there are any differences between structural values at the beginning and at the end of a period, these are dealt with as follows. For tie variables that have a structural value at the start of the period, this value is used to replace the observed value at the end of the period (for the goodness of fit assessment only). For tie variables that have a structural value at the end of the period but a free value value at the start of the period, the reference value for the simulated values is lacking; therefore, the simulated values at the end of the period then are replaced by the structural value at the end of the period (again, for the goodness of fit assessment only).

The auxiliary statistics documented in `sienaGOF-auxiliary` are calculated for the simulated dependent variables in Phase 3 of the estimation algorithm, returned in `sienaFitObject` because of having used `returnDeps = TRUE` in the call to `siena07`. These statistics should be chosen to represent features of the network that are not explicitly fit by the estimation procedure but can be considered important properties that the model at hand should represent well. Some examples are:

- Outdegree distribution
- Indegree distribution
- Distribution of the dependent behavior variable (if any).
- Distribution of geodesic distances
- Triad census
- Edgewise homophily counts
- Edgewise shared partner counts
- Statistics depending on the combination of network and behavioral variables.

The function is written so that the user can easily define other functions to capture some other relevant aspects of the network, behaviors, etc. This is further illustrated in the help page `sienaGOF-auxiliary`.

We recommend the following heuristic approach to model checking:

1. Check convergence of the estimation.
2. Assess time heterogeneity by `sienaTimeTest` and if there is evidence for time heterogeneity either modify the base effects or include time dummy terms.

3. Assess goodness of fit (primarily using `join=TRUE`) on auxiliary statistics, and if necessary refine the model.

The `print` function will display some useful information to help with model selection if some effects are set to FIX and TEST on the effects object. A rough estimator for the Mahalanobis distance that would be obtained at each proposed specification is given in the output. This can help guide model selection. This estimator is called the modified Mahalanobis distance (MMD). See Lospinoso (2012), the manual, or the references for more information.

The following functions are pre-fabricated for ease of use, and can be passed in as the `auxiliaryFunction` with no extra effort; see `sienaGOF-auxiliary` and the examples below.

- `IndegreeDistribution`
- `OutdegreeDistribution`
- `BehaviorDistribution`

Value

`sienaGOF` returns a result of class `sienaGOF`; this is a list of elements of class `sienaGofTest`; if `join=TRUE`, the list has length 1; if `join=FALSE`, each list element corresponds to a period analyzed; the list elements are themselves lists again, including the following elements:

- `sienaFitName` The name of sienaFitObject.
- `auxiliaryStatisticName` The name of auxiliaryFunction.
- `Observations` The observed values for the auxiliary statistics.
- `Simulations` The simulated auxiliary statistics.
- `ObservedTestStat` The observed Mahalanobis distance in the data.
- `SimulatedTestStat` The Mahalanobis distance for the simulations.
- `TwoTailed` Whether the p-value corresponds to a one- or two-tailed Monte Carlo test.
- `p` The p-value for the observed Mahalanobis distance in the permutation distribution of the simulated Mahalanobis distances.
- `Rank` Rank of the covariance matrix of the simulated auxiliary statistics.

In addition there are several attributes which give, for model specifications with fixed-and-tested effects, approximations to the expected Mahalanobis distance for model specifications where each of these effects would be added. This is reported in the `summary` method.
The `plot` method makes violin plots or box plots, with superimposed confidence bands, for the simulated distributions of all elements of the `auxiliaryFunction`, with the observed values indicated by red dots; but statistics with variance 0 are dropped.

`descriptives.sienaGOF` returns a matrix giving numerical information about what is plotted in the `plot` method: maximum, upper percentile, mean, median, lower percentile, minimum, of the simulated distributions of the auxiliary statistics, and the observed values. If `center=TRUE` the median is subtracted from mean, median, and percentiles; if `scale=TRUE` these numbers are divided by (maximum - minimum).
If `showAll=FALSE`, statistics with variance 0 will be dropped.

Author(s)

Josh Lospinoso, modifications by Ruth Ripley and Tom Snijders

References

- See http://www.stats.ox.ac.uk/~snijders/siena/ for general information on RSiena.
- Lospinoso, J.A. and Snijders, T.A.B., "Goodness of fit for Stochastic Actor Oriented Models." Presentation given at Sunbelt XXXI, St. Petes Beach, Fl. 2011.
- Lospinoso, J.A. (2012). "Statistical Models for Social Network Dynamics." Ph.D. Thesis. University of Oxford: U.K.

See Also

siena07, sienaGOF-auxiliary, sienaTimeTest

Examples

```
mynet <- sienaDependent(array(c(s501, s502), dim=c(50, 50, 2)))
mybeh <- sienaDependent(s50a[,1:2], type="behavior")
mydata <- sienaDataCreate(mynet, mybeh)
myeff <- getEffects(mydata)
myeff <- includeEffects(myeff, transTrip)
myeff <- setEffect(myeff, cycle3, fix=TRUE, test=TRUE)
myeff <- setEffect(myeff, transTies, fix=TRUE, test=TRUE)
myalgorithm <- sienaAlgorithmCreate(nsub=1, n3=50)
# Shorter phases 2 and 3, just for example.
ans <- siena07(myalgorithm, data=mydata, effects=myeff, batch=TRUE, returnDeps=TRUE)
gofi <- sienaGOF(ans, IndegreeDistribution, verbose=TRUE, join=TRUE,
  varName="mynet")
summary(gofi)
plot(gofi)

## Not run:
# Illustration just for showing a case with two dependent networks;
# running time backwards is not meaningful!
mynet1 <- sienaDependent(array(c(s501, s502), dim=c(50, 50, 2)))
mynet2 <- sienaDependent(array(c(s503, s501), dim=c(50, 50, 2)))
mybeh <- sienaDependent(s50a[,1:2], type="behavior")
mydata <- sienaDataCreate(mynet1, mynet2, mybeh)
myeff <- getEffects(mydata)
myeff <- includeEffects(myeff, transTrip)
myeff <- includeEffects(myeff, recip, name="mynet2")
myeff <- setEffect(myeff, cycle3, fix=TRUE, test=TRUE)
myeff <- setEffect(myeff, transTies, fix=TRUE, test=TRUE)
myalgorithm <- sienaAlgorithmCreate(nsub=1, n3=50)
# Shorter phases 2 and 3, just for example.
ans <- siena07(myalgorithm, data=mydata, effects=myeff, batch=TRUE, returnDeps=TRUE)
gofi <- sienaGOF(ans, IndegreeDistribution, verbose=TRUE, join=TRUE,
  varName="mynet1")
summary(gofi)
plot(gofi)
```

```
## End(Not run)

## Not run:
(gofi.nc <- sienaGOF(ans, IndegreeDistribution, cumulative=FALSE,
    varName="mynet1"))
# cumulative is an example of "...".
plot(gofi.nc)
descriptives.sienaGOF(gofi.nc)

(gofi2 <- sienaGOF(ans, IndegreeDistribution, varName="mynet2"))
plot(gofi2)

(gofb <- sienaGOF(ans, BehaviorDistribution, varName = "mybeh"))
plot(gofb)

(gofo <- sienaGOF(ans, OutdegreeDistribution, varName="mynet1",
    levls=0:6, cumulative=FALSE))
# levls is another example of "...".
plot(gofo)

## End(Not run)

## A demonstration of using multiple processes
## Not run:
library(parallel)
(n.clus <- detectCores() - 1) # subtract 1 to keep time for other processes
myalgorithm.c <- sienaAlgorithmCreate(nsub=4, n3=1000, seed=12345)
(ans.c <- siena07(myalgorithm.c, data=mydata, effects=myeff, batch=TRUE,
    returnDeps=TRUE, useCluster=TRUE, nbrNodes=n.clus))
gofi.1 <- sienaGOF(ans.c, IndegreeDistribution, verbose=TRUE,
  varName="mynet1")
cl <- makeCluster(n.clus)
gofi.cl <- sienaGOF(ans.c, IndegreeDistribution, varName="mynet1",
  cluster=cl)
# compare simulation times
attr(gofi.1,"simTime")
attr(gofi.cl,"simTime")

## End(Not run)
```

`sienaGOF-auxiliary` *Auxiliary functions for goodness of fit assessment by* `sienaGOF`

Description

The functions given here are auxiliary to function `sienaGOF` which assesses goodness of fit for actor-oriented models.

The auxiliary functions are, first, some functions of networks or behavior (i.e., statistics) for which the simulated values for the fitted model are compared to the observed value; second, some extraction functions to extract the observed and simulated networks and/or behavior from the sienaFit object produced by siena07 with returnDeps=TRUE.

These functions are exported here mainly to enable users to write their own versions. At the end of this help page some non-exported functions are listed. These are not exported because they depend on packages that are not in the R base distribution; and to show templates for readers wishing to contruct their own functions.

Usage

```
OutdegreeDistribution(i, obsData, sims, period, groupName, varName,
         levls=0:8, cumulative=TRUE)

IndegreeDistribution(i, obsData, sims, period, groupName, varName,
         levls=0:8, cumulative=TRUE)

BehaviorDistribution(i, obsData, sims, period, groupName, varName,
         levls=NULL, cumulative=TRUE)

TriadCensus(i, obsData, sims, period, groupName, varName, levls=1:16)

mixedTriadCensus(i, obsData, sims, period, groupName, varName)

sparseMatrixExtraction(i, obsData, sims, period, groupName, varName)

networkExtraction(i, obsData, sims, period, groupName, varName)

behaviorExtraction(i, obsData, sims, period, groupName, varName)
```

Arguments

i	Index number of simulation to be extracted, ranging from 1 to length(sims); if NULL, the data observation will be extracted.
obsData	The observed data set to which the model was fitted; normally this is x$f where x is the sienaFit object for which the fit is being assessed.
sims	The simulated data sets to be compared with the observed data; normally this is x$sims where x is the sienaFit object for which the fit is being assessed.
period	Period for which data and simulations are used (may run from 1 to number of waves - 1).
groupName	Name of group; relevant for multi-group data sets; defaults in sienaGOF to "Data1".
varName	Name of dependent variable.
levls	Levels used as values of the auxiliary statistic. For BehaviorDistribution, this defaults to the observed range of values.
cumulative	Are the distributions to be considered as raw or cumulative (<=) distributions?

Details

The statistics should be chosen to represent features of the network that are not explicitly fit by the estimation procedure but can be considered important properties that the model at hand should represent well. The three given here are far from a complete set; they will be supplemented in due time by statistics depending on networks and behavior jointly. The examples below give a number of other statistics, using the packages sna and igraph.

The `levls` parameter must be adapted to the range of values that is considered important. For indegrees and outdegrees, the whole range should usually be covered. If the range is large, which could be the case, e.g., for indegrees of two-mode networks where the second mode has few nodes, think about the possibility of making a selection such as `levls=5*(0:20)` or `levls=c(0:4,5*(1:20))`; which in most cases will make sense only if `cumulative=TRUE`.

The method signature for the auxiliary statistics generally is `function(i, obsData, sims, period, groupName, varName, ...)`. For constructing new auxiliary statistics, it is helpful to study the code of `OutdegreeDistribution`, `IndegreeDistribution`, and `BehaviorDistribution` and of the example functions below.

`TriadCensus` returns the distribution of the Holland-Leinhardt triad census according to the algorithm by Batagelj and Mrvar (implementation by Parimalarangan Slota, and Madduri). An alternative is the `TriadCensus.sna` function mentioned below, from package `sna`, which gives the same results. Here the `levls` parameter can be used to exclude some triads, e.g., for non-directed networks.
The Batagelj-Mrvar algorithm is optimized for sparse, large graphs and may be much faster than the procedure implemented in `sna`. For dense graphs the `sna` procedure may be faster.

Value

`OutdegreeDistribution` returns a named vector, the distribution of the observed or simulated outdegrees for the values in `levls`.

`IndegreeDistribution` returns a named vector, the distribution of the observed or simulated indegrees for the values in `levls`.

`BehaviorDistribution` returns a named vector, the distribution of the observed or simulated behavioral variable for the values in `levls`.

`TriadCensus` returns a named vector, the distribution of the Holland-Leinhardt triad census according to the algorithm by Batagelj and Mrvar.

`mixedTriadCensus` returns a named vector, the distribution of the mixed triad census of Hollway, Lomi, Pallotti,and Stadtfeld (2017).

`sparseMatrixExtraction` returns the simulated network as a `dgCMatrix`; this is the "standard" class for sparse numeric matrices in the `Matrix` package. See the help file for `dgCMatrix-class`.
Tie variables for ordered pairs with a missing value for wave=period or period+1 are zeroed; note that this also is done in RSiena for calculation of target statistics. Tie variables that are structurally determined at the beginning of a period are used to replace observed values at the end of the period; tie variables that are structurally determined at the end, but not the beginning, of a period are used to replace simulated values at the end of the period.
To treat the objects returned by this function as regular matrices, it is necessary to attach the `Matrix` package in your session.

`networkExtraction` returns the network as an edge list of class "network" according to the network package (used for package sna). Missing values and structural values are treated as in `sparseMatrixExtraction`, see above.

`behaviorExtraction` returns the dependent behavior variable as an integer vector. Values for actors with a missing value for wave=period or period+1 are transformed to NA.

Author(s)

Josh Lospinoso, Tom Snijders

References

- Batagelj, V., and Mrvar, A. (2001). A subquadratic triad census algorithm for large sparse networks with small maximum degree. *Social networks*, 23, 237-243.
- See http://www.stats.ox.ac.uk/~snijders/siena/ for general information on RSiena.
- Holland, Paul W., and Leinhardt, Samuel (1976). Local structure in social networks. *Sociological Methodology*, 6, 1-45.
- Hollway, J., Lomi, A., Pallotti, F., and Stadtfeld, C. (2017) Multilevel social spaces: The network dynamics of organizational fields. *Network Science*, 5(2), 187-212.
- Lospinoso, J.A. and Snijders, T.A.B., (2011). "Goodness of fit for Stochastic Actor Oriented Models." Presentation given at Sunbelt XXXI, St. Petes Beach, FL.
- Lospinoso, J.A. (2012). "Statistical Models for Social Network Dynamics." Ph.D. Thesis. University of Oxford: U.K.
- Parimalarangan S., Slota, G.M., and Madduri, K. (2017). Fast parallel graph triad census and triangle counting on shared-memory platforms, 2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Lake Buena Vista, FL, pp. 1500-1509.

See Also

`siena07`, `sienaGOF`

Examples

```
### For use out of the box:

mynet1 <- sienaDependent(array(c(s501, s502), dim=c(50, 50, 2)))
mybeh <- sienaDependent(s50a[,1:2], type="behavior")
mydata <- sienaDataCreate(mynet1, mybeh)
myeff <- getEffects(mydata)
myeff <- includeEffects(myeff, transTies, cycle3)
# Shorter phases 2 and 3, just for example:
myalgorithm <- sienaAlgorithmCreate(nsub=1, n3=50, seed=1234)
(ans <- siena07(myalgorithm, data=mydata, effects=myeff, returnDeps=TRUE,
   batch=TRUE))

OutdegreeDistribution(NULL, ans$f, ans$sims, period=1, groupName="Data1",
  levls=0:7, varName="mynet1")
```

```
IndegreeDistribution(5, ans$f, ans$sims, period=1, groupName="Data1",
  varName="mynet1")
BehaviorDistribution(20, ans$f, ans$sims, period=1, groupName="Data1",
  varName="mybeh")
sparseMatrixExtraction(50, ans$f, ans$sims, period=1, groupName="Data1",
  varName="mynet1")
networkExtraction(40, ans$f, ans$sims, period=1, groupName="Data1",
  varName="mynet1")
behaviorExtraction(50, ans$f, ans$sims, period=1, groupName="Data1",
  varName="mybeh")

gofi <- sienaGOF(ans, IndegreeDistribution, verbose=TRUE, join=TRUE,
  varName="mynet1")
gofi
plot(gofi)

(gofo <- sienaGOF(ans, OutdegreeDistribution, verbose=TRUE, join=TRUE,
    varName="mynet1", cumulative=FALSE))
# cumulative is an example of "...".
plot(gofo)

(gofb <- sienaGOF(ans, BehaviorDistribution, varName = "mybeh",
    verbose=TRUE, join=TRUE, cumulative=FALSE))
plot(gofb)

(goftc <- sienaGOF(ans, TriadCensus, verbose=TRUE, join=TRUE,
    varName="mynet1"))
plot(goftc, center=TRUE, scale=TRUE)
# For this type of auxiliary statistics
# it is advisable in the plot to center and scale.
# note the keys at the x-axis.
descriptives.sienaGOF(goftc)
round(descriptives.sienaGOF(goftc, center=TRUE, scale=TRUE), 0)

## Not run:
### The mixed triad census for co-evolution of one-mode and two-mode networks:
actors <- sienaNodeSet(50, nodeSetName="actors")
activities <- sienaNodeSet(20, nodeSetName="activities")
onemodenet <- sienaDependent(array(c(s501, s502), dim=c(50, 50, 2)),
                             nodeSet="actors")
twomodenet <- sienaDependent(array(c(s502[1:50, 1:20], s503[1:50, 1:20]),
                                                        dim=c(50, 20, 2)),
                             type= "bipartite", nodeSet=c("actors", "activities"))
twodata <- sienaDataCreate(onemodenet, twomodenet,
                           nodeSets=list(actors, activities))
twoeff <- getEffects(twodata)
twoeff <- includeEffects(twoeff, outActIntn, name="onemodenet",
                             interaction1="twomodenet")
twoeff <- includeEffects(twoeff, outActIntn, name="twomodenet",
                             interaction1="onemodenet")
twoeff <- includeEffects(twoeff, from, name="onemodenet",
                             interaction1="twomodenet")
twoeff <- includeEffects(twoeff, to, name="twomodenet",
```

```
                              interaction1="onemodenet")
twoeff
# Shorter phases 2 and 3, just for example:
twoalgorithm <- sienaAlgorithmCreate(projname="twomode", nsub=2, n3=200, seed=1234)
(ans <- siena07(twoalgorithm, data=twodata, effects=twoeff, returnDeps=TRUE,
   batch=TRUE))
(gof.two <- sienaGOF(ans, mixedTriadCensus,
                        varName=c("onemodenet", "twomodenet"), verbose=TRUE))
plot(gof.two, center=TRUE, scale=TRUE)

## End(Not run)

## Not run:
### Here come some useful functions for building your own auxiliary statistics:
### First an extraction function.

# igraphNetworkExtraction extracts simulated and observed networks
# from the results of a siena07 run.
# It returns the network as an edge list of class "graph"
# according to the igraph package.
# Ties for ordered pairs with a missing value for wave=period or period+1
# are zeroed;
# note that this also is done in RSiena for calculation of target statistics.
# However, changing structurally fixed values are not taken into account.
igraphNetworkExtraction <- function(i, data, sims, period, groupName, varName) {
  require(igraph)
  dimsOfDepVar<- attr(data[[groupName]]$depvars[[varName]], "netdims")
  missings <- is.na(data[[groupName]]$depvars[[varName]][,,period]) |
    is.na(data[[groupName]]$depvars[[varName]][,,period+1])
  if (is.null(i)) {
    # sienaGOF wants the observation:
    original <- data[[groupName]]$depvars[[varName]][,,period+1]
    original[missings] <- 0
    returnValue <- graph.adjacency(original)
  }
  else
  {
    missings <- graph.adjacency(missings)
    #sienaGOF wants the i-th simulation:
    returnValue <- graph.difference(
      graph.empty(dimsOfDepVar) +
        edges(t(sims[[i]][[groupName]][[varName]][[period]][,1:2])),
      missings)
  }
  returnValue
}

### Then some auxiliary statistics.

# GeodesicDistribution calculates the distribution of non-directed
# geodesic distances; see ?sna::geodist
# The default for \code{levls} reflects that geodesic distances larger than 5
# do not differ appreciably with respect to interpretation.
```

```
# Note that the levels of the result are named;
# these names are used in the \code{plot} method.
GeodesicDistribution <- function (i, data, sims, period, groupName,
  varName, levls=c(1:5,Inf), cumulative=TRUE, ...) {
  x <- networkExtraction(i, data, sims, period, groupName, varName)
  require(network)
  require(sna)
  a <- sna::geodist(symmetrize(x))$gdist
  if (cumulative)
  {
    gdi <- sapply(levls, function(i){ sum(a<=i) })
  }
  else
  {
    gdi <- sapply(levls, function(i){ sum(a==i) })
  }
  names(gdi) <- as.character(levls)
  gdi
}

# Holland and Leinhardt Triad Census from sna; see ?sna::triad.census.
# For undirected networks, call this with levls=1:4
TriadCensus.sna <- function(i, data, sims, period, groupName, varName, levls=1:16){
  unloadNamespace("igraph") # to avoid package clashes
  require(network)
  require(sna)
  x <- networkExtraction(i, data, sims, period, groupName, varName)
  if (network.edgecount(x) <= 0){x <- symmetrize(x)}
  # because else triad.census(x) will lead to an error
  tc <- sna::triad.census(x)[levls]
  # names are transferred automatically
  tc
}

# Holland and Leinhardt Triad Census from igraph; see ?igraph::triad_census.
TriadCensus.i <- function(i, data, sims, period, groupName, varName){
  unloadNamespace("sna") # to avoid package clashes
  require(igraph)
  x <- igraphNetworkExtraction(i, data, sims, period, groupName, varName)
# suppressWarnings is used because else warnings will be generated
# when a generated network happens to be symmetric.
  setNames(suppressWarnings(triad_census(x)),
           c("003", "012", "102", "021D","021U", "021C", "111D", "111U",
           "030T", "030C", "201",  "120D", "120U", "120C", "210", "300"))
}

# CliqueCensus calculates the distribution of the clique census
# of the symmetrized network; see ?sna::clique.census.
CliqueCensus<-function (i, obsData, sims, period, groupName, varName, levls = 1:5){
  require(sna)
  x <- networkExtraction(i, obsData, sims, period, groupName, varName)
  cc0 <- sna::clique.census(x, mode='graph', tabulate.by.vertex = FALSE,
    enumerate=FALSE)[[1]]
```

```
  cc <- 0*levls
  names(cc) <- as.character(levls)
  levels.used <- as.numeric(intersect(names(cc0), names(cc)))
  cc[levels.used] <- cc0[levels.used]
  cc
}

# Distribution of Bonacich eigenvalue centrality; see ?igraph::evcent.
EigenvalueDistribution <- function (i, data, sims, period, groupName, varName,
  levls=c(seq(0,1,by=0.125)), cumulative=TRUE){
  require(igraph)
  x <- igraphNetworkExtraction(i, data, sims, period, groupName, varName)
  a <- igraph::evcent(x)$vector
  a[is.na(a)] <- Inf
  lel <- length(levls)
  if (cumulative)
  {
    cdi <- sapply(2:lel, function(i){sum(a<=levls[i])})
  }
  else
  {
    cdi <- sapply(2:lel, function(i){
      sum(a<=levls[i]) - sum(a <= levls[i-1])})
  }
  names(cdi) <- as.character(levls[2:lel])
  cdi
}

## Finally some examples of the three auxiliary statistics constructed above.
mynet1 <- sienaDependent(array(c(s501, s502, s503), dim=c(50, 50, 3)))
mybeh <- sienaDependent(s50a, type="behavior")
mydata <- sienaDataCreate(mynet1, mybeh)
myeff <- getEffects(mydata)
myeff <- includeEffects(myeff, transTrip, cycle3, nbrDist2)
myeff <- includeEffects(myeff, outdeg, name="mybeh", interaction1="mynet1")
myeff <- includeEffects(myeff,  outdeg, name="mybeh", interaction1="mynet1")
# Shorter phases 2 and 3, just for example:
myalgorithm <- sienaAlgorithmCreate(nsub=2, n3=200, seed=1234)
(ans2 <- siena07(myalgorithm, data=mydata, effects=myeff, returnDeps=TRUE,
   batch=TRUE))
gofc <- sienaGOF(ans2, EigenvalueDistribution, varName="mynet1",
  verbose=TRUE, join=TRUE)
plot(gofc)
descriptives.sienaGOF(gofc, showAll=TRUE)

goftc <- sienaGOF(ans2, TriadCensus, varName="mynet1", verbose=TRUE, join=TRUE)
plot(goftc, center=TRUE, scale=TRUE)
# For this type of auxiliary statistics
# it is advisable in the plot to center and scale.
# note the keys at the x-axis; these names are given by sna::triad.census
descriptives.sienaGOF(goftc)
round(descriptives.sienaGOF(goftc))
```

```
gofgd <- sienaGOF(ans2, GeodesicDistribution, varName="mynet1",
  verbose=TRUE, join=TRUE, cumulative=FALSE)
plot(gofgd)
# and without infinite distances:
gofgdd <- sienaGOF(ans2, GeodesicDistribution, varName="mynet1",
  verbose=TRUE, join=TRUE, levls=1:7, cumulative=FALSE)
plot(gofgdd)

## End(Not run)
```

`sienaGroupCreate` *Function to group together several Siena data objects*

Description

Creates an object of class "sienaGroup" from a list of Siena data objects.

Usage

```
sienaGroupCreate(objlist, singleOK = FALSE, getDocumentation=FALSE)
```

Arguments

`objlist`	List of objects of class "siena".
`singleOK`	Boolean: is it OK to only have one object?
`getDocumentation`	Flag to allow documentation of internal functions, not for use by users.

Details

This function creates a Siena group object from several Siena data objects ('groups'), all of which use networks, covariates and actor sets with the same names. The variables must correspond exactly between all data objects; the numbers of waves may differ. It can be used as data input to `siena07` for the multigroup option. Also used internally for convenience with a single Siena data object.

For later use in `siena07` or `sienaBayes`, it will often (but not always...) be helpful when creating the Siena data objects in `objlist` to use `allowOnly=FALSE` in the call of `sienaDependent`; see the help page for this function.

Value

An object of class "sienaGroup"; this is a list containing the input objects, with attributes:

`netnames`	names of the dependent variables in each set
`symmetric`	vector of booleans, one for each dependent variable. TRUE if all occurrences of the network are symmetric.
`structural`	vector of booleans, indicating whether structurally fixed values occur in this network

allUpOnly	vector of booleans, indicating whether changes are all upwards in all the occurrences of this network
allDownOnly	similar to previous, but for downward changes
anyUpOnly	vector of booleans, indicating whether changes are all upwards in any of the occurrences of this network
anyDownOnly	similar to previous, but for downward changes
types	vector of network types of the dependent variables
observations	Total number of periods to process
periodNos	Sequence of numbers of periods which are not skipped in multigroup processing
netnodeSets	list of names of the node sets corresponding to the dependent variables
cCovars	names of the constant covariates, if any
vCovars	names of the changing covariates, if any
dycCovars	names of the constant dyadic covariates, if any
dyvCovars	names of the changing dyadic covariates, if any
ccnodeSets	list of the names of the node sets corresponding to the constant covariates
cvnodeSets	list of the names of the node sets corresponding to the changing covariates
dycnodeSets	list of the names of the node sets corresponding to the constant dyadic covariates
dyvcnodeSets	list of the names of the node sets corresponding to the changing dyadic covariates
compositionChange	boolean: any composition change at all?
exooptions	named vector of composition change options for the node sets
names	Either from the input objects or "Data1", "Data2" etc
class	"sienaGroup" inheriting from "siena"
balmean	vector of means for balance calculations
bRange	vector of difference between maximum and minimum values for behavior variables, NA for other dependent variables
behRange	matrix of maximum and minimum values for behavior variables, NA for other dependent variables
bSim	vector of similarity means for behavior variables, NA for other dependent variables
bPoszvar	vector of booleans indicating positive variance for behavior variables. NA for other dependent variables
bMoreThan2	vector of booleans indicating whether the behavior variables take more than 2 distinct values
cCovarPoszvar	vector of booleans indicating positive variance for constant covariates
cCovarMoreThan2	vector of booleans indicating whether the constant covariates take more than 2 distinct values
cCovarRange	vector of difference between maximum and minimum values for constant covariates

cCovarRange2	matrix of maximum and minimum values for constant covariates
cCovarSim	vector of similarity means for constant covariates
cCovarMean	vector of means for constant covariates
vCovarRange	vector of difference between maximum and minimum values for changing covariates
vCovarSim	vector of similarity means for changing covariates
vCovarMoreThan2	vector of booleans indicating whether the changing covariates take more than 2 distinct values
vCovarPoszvar	vector of booleans indicating positive variance for changing covariates
vCovarMean	vector of means for changing covariates
dycCovarMean	vector of means for constant dyadic covariates
dycCovarRange	vector of ranges for constant dyadic covariates
dycCovarRange2	matrix of maximum and minimum values for constant dyadic covariates
dyvCovarRange	vector of ranges for changing dyadic covariates
dyvCovarMean	vector of means for changing dyadic covariates
anyMissing	vector of booleans, one for each dependent variable, indicating the presence of any missing values
netRanges	matrix of maximum and minimum values for dependent networks, NA for behavior variables

Author(s)

Ruth Ripley

References

See the Section on Multi-group Siena analysis in the manual available from http://www.stats.ox.ac.uk/~snijders/siena/.

See Also

sienaDataCreate

Examples

```
Group1 <- sienaDependent(array(c(N3401, HN3401), dim=c(45, 45, 2)))
Group3 <- sienaDependent(array(c(N3403, HN3403), dim=c(37, 37, 2)))
Group4 <- sienaDependent(array(c(N3404, HN3404), dim=c(33, 33, 2)))
Group6 <- sienaDependent(array(c(N3406, HN3406), dim=c(36, 36, 2)))
dataset.1 <- sienaDataCreate(Friends = Group1)
dataset.3 <- sienaDataCreate(Friends = Group3)
dataset.4 <- sienaDataCreate(Friends = Group4)
dataset.6 <- sienaDataCreate(Friends = Group6)
FourGroups <- sienaGroupCreate(list(dataset.1, dataset.3, dataset.4, dataset.6))
```

`sienaNodeSet` *Function to create a node set*

Description

Creates a Siena node set which can be used as the nodes in a siena network.

Usage

```
sienaNodeSet(n, nodeSetName="Actors", names=NULL)
```

Arguments

`n`	integer, size of set.
`nodeSetName`	character string naming the node set.
`names`	optional character string vector of length n of the names of the nodes.

Details

This function is important for data sets having more than one node set, but not otherwise.

Value

Returns a Siena node set, an integer vector, possibly with names, plus the attributes, class equal to "sienaNodeSet", and nodeSetName equal to the argument nodeSetName.

Author(s)

Ruth Ripley

References

See http://www.stats.ox.ac.uk/~snijders/siena/

Examples

```
students <- sienaNodeSet(50, "student")
```

sienaRI *Functions to assess the relative importance of effects at observation moments*

Description

The function sienaRI returns the relative importance of effects of a SAOM according to the measure of relative importance described in Section 3.1 of Indlekofer and Brandes (2013). The measure is based on the influence of effects on potential tie change or behavior change decisions of individual actors at the given observation moments. It takes the data as well as the complete model specification into account. Therefore, necessary arguments are the analysed data given as a siena data object as well as the complete model specification represented either by an estimated sienaFit object or by the triple consisting of a suitable parameter vector theta and the corresponding sienaAlgorithm and sienaEffects objects.
Entropy-based effect sizes as in Snijders (2004) and the within-ego standard deviations of change statistics are also computed. If getChangeStats=TRUE, the arrays of change statistics are stored in the sienaRI object.

Usage

```
sienaRI(data, ans=NULL, theta=NULL, algorithm=NULL, effects=NULL,
        getChangeStats=FALSE)
## S3 method for class 'sienaRI'
print(x, printSigma=FALSE,...)
## S3 method for class 'sienaRI'
plot(x, actors = NULL, col = NULL, addPieChart = FALSE,
              radius = 1, width = NULL, height = NULL, legend = TRUE,
              legendColumns = NULL, legendHeight = NULL,
              cex.legend = NULL, cex.names = NULL,...)
```

Arguments

data	siena data object representing the analyzed data and resulting from a call to sienaDataCreate.
ans	sienaFit object resulting from a call to siena07. The sienaFit object contains all necessary information on the model specification, in particular, the vector of parameter values ans$theta, the used algorithm for estimation ans$x, and information on included model effects ans$effects. If ans is a valid sienaFit object, the calculations of relative importances are based on ans$theta, ans$x, and ans$effects. Alternatively, the necessary information can be given directly as a suitable parameter vector theta, a sienaAlgorithm object, and a sienaEffects object. In this case, ans has to be unspecified (i.e., ans=NULL).
theta	Vector of parameter values of effects included in the model. Length of theta has to be equal to the number of included effects.
algorithm	sienaAlgorithm object as resulting from a call to sienaAlgorithmCreate. Works only for estimation by Method of Moments (i.e., if maxlike = FALSE).

effects	sienaEffects object specifying which effects are included the model. Note that sienaRI does not yet work for endowment or creation effect, i.e., included effects have to be of type eval (or rate).
getChangeStats	Boolean: If TRUE, an array of change statistics is added to the sienaRI object.
x	sienaRI object resulting from a call to sienaRI.
printSigma	Boolean: If TRUE, average within-ego standard deviations of change statistics ('sigma'), are included in the print.
actors	vector of integers: set of actors to be included in the plot; if NULL, all actors.
col	Colors used in the plot. If col=NULL a default color scheme is used.
addPieChart	Boolean: If TRUE, pie charts of aggregated relative importances for the complete set of actors will be added to the plot.
radius	Radius of pie charts. Only effective if addPieCharts = TRUE.
width	Width of the plot. If width=NULL a default value is used.
height	Height of the plot. If height=NULL a default value is used.
legend	Boolean: if TRUE a legend is added to the plot. x$effectNames are used as labels.
legendColumns	Number of columns in legend. If legendColumns=NULL a default value is used. Only effective if legend=TRUE.
legendHeight	Height of legend. If legendHeight=NULL a default value is used. Only effective if legend=TRUE.
cex.legend	Specifies the relative font size of legend labels.
cex.names	Specifies the relative font size of bar graph labels.
...	Other arguments.

Details

sienaRI takes the data as well as the complete model specification into account. Therefore, necessary arguments are the analyzed data given as a siena data object as well as the complete model specification represented either by an estimated sienaFit object or by the triple consisting of a suitable parameter vector theta and the corresponding sienaAlgorithm and sienaEffects objects.

A sienaFit object contains all necessary information on the model specification, in particular, the vector of parameter values ans$theta, the used algorithm for estimation ans$x, and information on included model effects ans$effects.

If ans is a valid sienaFit object, the calculations of relative importances are based on ans$theta, ans$x, and ans$effects. Alternatively, the necessary information can be given directly as a suitable parameter vector theta, a sienaAlgorithm object, and a sienaEffects object. In this case, ans has to be unspecified, i.e., ans=NULL.

Note that sienaRI works only with Method of Moments (i.e., for sienaAlgorithm objects with maxlike = FALSE) and that it does not yet work for endowment or creation effects (i.e., included effects have to be of type eval), and also not for models with interaction effects. It does not allow two-mode (bipartite) networks as dependent variables; but these can be represented as

one-mode networks using structural zeros. If the network is non-directed, the relative importances and entropy-based 'degrees of certainty' are calculated for `modelType=2` ('forcing'; see `sienaAlgorithmCreate`).

If there are any missing tie values in the network data set, they are imputed by initial zeros and Last Observation Carried Forward. Structural zeros and ones are replaced by `NA` and treated as impossible choices in the probability vectors and ignored in the standard deviations; but the change statistics for these dyads still are given in `changeStatistics` (if requested).

Value

If the model contains only one dependent variable, `sienaRI` returns an object of class `sienaRI`. Otherwise, it returns a list of objects of class `sienaRI`, each corresponding to one dependent variable.

A returned `sienaRI` object stores the expected relative importances of effects of one dependent variable at observation moments as defined in Section 3.1 of Indlekofer and Brandes (2013).

A `sienaRI` object is a list with the following components. For the components referred to as lists themselves, these are lists corresponding to the observation moments.

- `dependentVariable` the name of the corresponding dependent variable.
- `effectNames` the names of considered effects.
- `RIActors` a list that contains the expected relative importances of effects for each potential actor decision at observation moments. This is equation (3) in Indlekofer and Brandes (2013).
- `expectedRI` a list that contains the expected relative importances of effects aggregated over all actors for each network observation. These are the averages of the actor related values in `RIActors`. This is equation (4) in Indlekofer and Brandes (2013).
- `IActors` a list that contains the expected importances of effects for each potential actor decision at observation moments. This is the numerator of equation (3) in Indlekofer and Brandes (2013).
- `expectedI` is a list that contains the expected importances of effects aggregated over all actors in each observation. More precisely, it contains the averages of the actor related values in `IActors`.
- `absoluteSumActors` a list that contains the sum of the (unstandardized) L1-differences calculated for each potential actor decision at observation moments. This is the denominator of equation (3) in Indlekofer and Brandes (2013).
- `RHActors` a list that contains the degree of certainty in the potential ministep taken by an actor at the observation moments; this is R_H(i,x) of formula (6) in Snijders (2004). The mean over actors of these degrees of certainty, given by formula (7) in Snijders (2004), is printed by the `print` method for `sienaRI` objects.
- `sigma` a list of effects by ego matrices of the values of the within-ego standard deviations of the change statistics. Their averages (over egos) are printed if `printSigma=TRUE`.
- `changeStatistics` a list of arrays (effects by alters by egos) containing for each observation moment, the values of the change statistics for toggling the tie from actor to ego; this is produced only if `getChangeStats=TRUE`.

Author(s)

Natalie Indlekofer, some additions by Tom Snijders

References

- Indlekofer, Natalie and Brandes, Ulrik (2013). Relative Importance of Effects in Stochastic Actor-oriented Models. *Network Science*, 1 (3), 278-304.
- Snijders, Tom A.B. (2004). Explained Variation in Dynamic Network Models. *Mathematics and Social Sciences*, 168 (4), 31-41.

Examples

```
myalgorithm <- sienaAlgorithmCreate(nsub=2, n3=50)
mynet1 <- sienaDependent(array(c(tmp3, tmp4), dim=c(32, 32, 2)))
mydata <- sienaDataCreate(mynet1)
myeff <- getEffects(mydata)
myeff <- includeEffects(myeff, density, recip, transTies, nbrDist2)
ans <- siena07(myalgorithm, data=mydata, effects=myeff, batch=TRUE)

RI <- sienaRI(mydata, ans)
RI
## Not run:
plot(RI, addPieChart=TRUE)
plot(RI, actors=1:20, addPieChart=TRUE, radius=1.08)

## End(Not run)

myalgorithm <- sienaAlgorithmCreate(nsub=2, n3=50)
mynet2 <- sienaDependent(array(c(s502, s503), dim=c(50, 50, 2)))
mybeh <- sienaDependent(s50a[,2:3], type="behavior")
mydata2 <- sienaDataCreate(mynet2, mybeh)
myeff2 <- getEffects(mydata2)
myeff2 <- includeEffects(myeff2, density, recip, transTies)
ans2 <- siena07(myalgorithm, data=mydata2, effects=myeff2, batch=TRUE)
# Use only the parameters for the evaluation function:
theta.eval <- ans2$theta[myeff2$type[myeff2$include]=="eval"]
RI <- sienaRI(mydata2, theta=theta.eval, algorithm=myalgorithm,
              effects = myeff2)
RI
## Not run:
plot(RI[[2]], col = c("red", "green"), legend=FALSE)
plot(RI[[1]], addPieChart = TRUE, legendColumns=2)

## End(Not run)
```

---

sienaTimeTest *Functions to assess and account for time heterogeneity of parameters*

Description

Takes a sienaFit object estimated by Method of Moments, and tests for time heterogeneity by the addition of interactions with time dummy variables at waves m=2...(M-1). The test used is the score-type test of Schweinberger (2012). Tests for joint significance, parameter-wise significance, period-wise significance, individual significance, and one-step estimates of the unrestricted model parameters are returned in a list.

Usage

```
sienaTimeTest(sienaFit, effects=NULL, excludedEffects=NULL, condition=FALSE)
```

Arguments

sienaFit	A sienaFit object returned by siena07.
effects	Optional vector of effect numbers to test. Use the number on the print of the sienaFit object.
excludedEffects	Optional vector of effect numbers for which time heterogeneity is not to be tested. Use the number on the print of the sienaFit object.
condition	Whether to orthogonalize effect-wise score-type tests and individual significance tests against estimated effects and un-estimated dummy terms, or just against estimated effects.

Details

This test follows the score type test of Schweinberger (2012) as elaborated by Lospinoso et al. (2011) by using statistics already calculated at each wave to obtain vectors of partitioned moment functions corresponding to a restricted model (the model in the sienaFit object; used as null hypothesis) and an unrestricted model (which contains dummies for waves m=2...(M-1); used as alternative hypothesis).

condition=TRUE leads to a rough-and-easy approximation to controlling the mentioned tests also for the unestimated effects.

After assessing time heterogeneity, effects objects can be modified by adding numbers of all or some periods to the timeDummy column. This is facilitated by the includeTimeDummy function. For an effects object in which the timeDummy column of some of the included effects includes some or all period numbers, interactions of those effects with time dummies for the indicated periods will also be estimated.

An alternative to the use of includeTimeDummy is to define time-dependent actor covariates (dummy variables or other functions of wave number that are the same for all actors), include these in the data set through sienaAlgorithmCreate, and include interactions of other effects with ego effects of these time-dependent actor covariates by includeInteraction. This is illustrated in an example below. Using includeTimeDummy is easier; using self-defined interactions with time-dependent variables gives more control.

If you wish to use this function with sienaFit objects that use the finite differences method of derivative estimation, or which use maximum likelihood estimation, you must request the derivatives to be returned by wave using the byWave=TRUE option for siena07.

Effects leading to dummy interactions that are collinear with the model originally fitted, after excluding the effects mentioned, will be automatically excluded from the time heterogeneity testing.

If `sienaTimeTest` gives errors that there are too many collinear effects, run it with a smaller `effects` parameter. For example, if the model has 40 effects of which the first 8 are rate parameters and therefore uninteresting, and there is such an error message, try `effects=9:30`; if that still does not work, decrease the upper limit of 30, if it does work increase it, to find the largest possible set of effects for which heterogeneity assessment still is possible; then as a next step try the remaining effects in a similar way.

Value

`sienaTimeTest` returns a list containing many items, including the following:

`JointTest`	A chi-squared test for joint significance of the dummies.
`EffectTest`	A chi-squared test for joint significance across dummies for each separate effect.
`GroupTest`	A chi-squared test for joint significance across dummies; if `sienaFit` is a fit for a multi-group object then these refer to each group; else they refer to ecah period.
`IndividualTest`	A matrix displaying initial estimates, one-step estimates, and p-values for the individual interactions.

Author(s)

Josh Lospinoso, Tom Snijders

References

See http://www.stats.ox.ac.uk/~snijders/siena/ for general information on RSiena.

J.A. Lospinoso, M. Schweinberger, T.A.B. Snijders, and R.M. Ripley (2011). Assessing and Accounting for Time Heterogeneity in Stochastic Actor Oriented Models. *Advances in Data Analysis and Computation*, 5:147-176.

M. Schweinberger (2012). Statistical modeling of network panel data: Goodness-of-fit. *British Journal of Statistical and Mathematical Psychology*, 65:263-281.

See Also

`siena07`, `plot.sienaTimeTest`, `includeTimeDummy`

Examples

```
## Estimate a restricted model
myalgorithm <- sienaAlgorithmCreate(nsub=1, n3=50)
# Short estimation not for practice, just for having a quick demonstration
mynet1 <- sienaDependent(array(c(s501, s502, s503), dim=c(50, 50, 3)))
mydata <- sienaDataCreate(mynet1)
myeff <- getEffects(mydata)
myeff <- includeEffects(myeff, transTrip)
ans <- siena07(myalgorithm, data=mydata, effects=myeff, batch=TRUE)
```

```
## Conduct the score-type test to assess whether heterogeneity is present.
tt <- sienaTimeTest(ans)
summary(tt)

## Suppose that we wish to include time dummies.
## Add them in the following way:
myeff <- includeTimeDummy(myeff, recip, transTrip, timeDummy="2")
ans2 <- siena07(myalgorithm, data=mydata, effects=myeff, batch=TRUE)

## Re-assess the time heterogeneity
(tt2 <- sienaTimeTest(ans2))

## And so on..

## A demonstration of the plotting facilities, on a larger dataset:
## (Of course pasting these identical sets of three waves after each other
##  in a sequence of six is not really meaningful. It's just a demonstration.)

myalgorithm <- sienaAlgorithmCreate(nsub=2, n3=50, seed=1234)
mynet1 <- sienaDependent(array(c(s501, s502, s503, s501, s503, s502),
                               dim=c(50, 50, 6)))
mydata <- sienaDataCreate(mynet1)
myeff <- getEffects(mydata)
myeff <- includeEffects(myeff, transTrip, balance)
myeff <- includeTimeDummy(myeff, density, timeDummy="all")
myeff <- includeTimeDummy(myeff, recip, timeDummy="2,3,5")
myeff <- includeTimeDummy(myeff, balance, timeDummy="4")
## Not run:
(ansp <- siena07(myalgorithm, data=mydata, effects=myeff, batch=TRUE))
ttp <- sienaTimeTest(ansp)

## Pairwise plots show
plot(ttp, pairwise=TRUE)

## Time test plots show
plot(ttp, effects=1:4, dims=c(2,2))

## End(Not run)

## Instead of working with includeTimeDummy,
## you can also define time dummies explicitly;
## this may give more control and more clarity:
dum2 <- matrix(c(0,1,0,0,0), nrow=50, ncol=5, byrow=TRUE)
dum3 <- matrix(c(0,0,1,0,0), nrow=50, ncol=5, byrow=TRUE)
dum4 <- matrix(c(0,0,0,1,0), nrow=50, ncol=5, byrow=TRUE)
dum5 <- matrix(c(0,0,0,0,1), nrow=50, ncol=5, byrow=TRUE)
time2 <- varCovar(dum2)
time3 <- varCovar(dum3)
time4 <- varCovar(dum4)
time5 <- varCovar(dum5)
mydata <- sienaDataCreate(mynet1, time2, time3, time4, time5)
myeff <- getEffects(mydata)
```

```
myeff <- includeEffects(myeff, transTrip, balance)
## corresponding to includeTimeDummy(myeff, density, timeDummy="all"):
myeff <- includeEffects(myeff, egoX, interaction1='time2')
myeff <- includeEffects(myeff, egoX, interaction1='time3')
myeff <- includeEffects(myeff, egoX, interaction1='time4')
myeff <- includeEffects(myeff, egoX, interaction1='time5')
## corresponding to myeff <- includeTimeDummy(myeff, recip, timeDummy="2,3,5"):
myeff <- includeInteraction(myeff, egoX, recip, interaction1=c('time2', ''))
myeff <- includeInteraction(myeff, egoX, recip, interaction1=c('time3', ''))
myeff <- includeInteraction(myeff, egoX, recip, interaction1=c('time5', ''))
## corresponding to myeff <- includeTimeDummy(myeff, balance, timeDummy="4"):
myeff <- includeInteraction(myeff, egoX, balance, interaction1=c('time4', ''))
## Not run:
(anspp <- siena07(myalgorithm, data=mydata, effects=myeff, batch=TRUE))
## anspp contains identical results as ansp above.

## End(Not run)

## A demonstration of RateX heterogeneity. Note that rate interactions are
## not implemented in general, just for Rate x cCovar.
## Not run:
myalgorithm <- sienaAlgorithmCreate(nsub=4, n3=1000)
mynet1 <- sienaDependent(array(c(s501, s502, s503), dim=c(50, 50, 3)))
myccov <- coCovar(s50a[,1])
mydata <- sienaDataCreate(mynet1, myccov)
myeff <- getEffects(mydata)
myeff <- includeEffects(myeff, transTrip, balance)
myeff <- includeTimeDummy(myeff, RateX, type="rate", interaction1="myccov")
ans <- siena07(myalgorithm, data=mydata, effects=myeff, batch=TRUE)

## End(Not run)
```

`simstats0c` *Versions of FRAN*

Description

The functions to be called as "FRAN" by `siena07`. They call compiled C++.

Usage

```
simstats0c(z, x, data=NULL, effects=NULL, fromFiniteDiff=FALSE,
           returnDeps=FALSE, returnChains=FALSE, byWave=FALSE,
           returnDataFrame=FALSE, returnLoglik=FALSE)
maxlikec(z, x, data=NULL, effects=NULL,
           returnChains=FALSE, byGroup = FALSE, byWave=FALSE,
           returnDataFrame=FALSE, returnLoglik=FALSE,
           onlyLoglik=FALSE)
initializeFRAN(z, x, data, effects, prevAns = NULL, initC,
```

```
            profileData = FALSE, returnDeps = FALSE, returnChains =
            FALSE, byGroup = FALSE, returnDataFrame = FALSE,
            byWave = FALSE, returnLoglik = FALSE,  onlyLoglik = FALSE)
terminateFRAN(z, x)
```

Arguments

z	Control object, passed in automatically in siena07.
x	A sienaAlgorithm object, passed in automatically in siena07.
data	A sienaData object as returned by sienaDataCreate.
effects	A sienaEffects object as returned by getEffects.
fromFiniteDiff	Boolean used during calculation of derivatives by finite differences. Not for user use.
returnDeps	Boolean. Whether to return the simulated networks in Phase 3.
returnChains	Boolean. Whether to return the chains.
byWave	Boolean. Whether to return the finite difference or maximum likelihood derivatives by wave (uses a great deal of memory). Only necessary for sienaTimeTest
byGroup	Boolean. For internal use: allows different thetas for each group to be used in sienaBayes.
returnDataFrame	Boolean. Whether to return the chains as lists or data frames.
returnLoglik	Boolean. Whether to return the log likelihood of the simulated chain.
onlyLoglik	Boolean: whether to return just the likelihood for the simulated chain, plus details of steps accepted and rejected.
prevAns	An object of class "sienaFit" as returned by siena07, from which scaling information (derivative matrix and standard deviation of the deviations) will be extracted along with the latest version of the parameters which will be used as the initial values, unless the model requests the use of standard initial values. If the previous model is exactly the same as the current one, Phase 1 will be omitted. If not, any parameter estimates for effects which are included in the new model will be used as initial values, but phase 1 will still be carried out. If the results used as prevAns are a reasonable starting point, this will increase the efficiency of the algorithm.
initC	If TRUE, call is to setup the data and model in C++. For use with multiple processes only.
profileData	Boolean to force dumping of the data for profiling with sienaProfile.exe.

Details

Not for general users' use.
The name of simstats0c or maxlikec should be used for the element FRAN of the model object, the former when using estimation by forward simulation, the latter for maximum likelihood estimation. The arguments with no defaults must be passed in on the call to siena07. initializeFRAN and terminateFRAN are called in both cases.

Value

simstats0c returns a list containing:

fra	Simulated statistics.
sc	Scores with which to calculate the derivative (not phase 2 or if using finite differences or maximum likelihood).
dff	Contributions to the derivative if finite differences
ntim	For conditional processing, time taken.
feasible	Currently set to TRUE.
OK	Could be set to FALSE if serious error has occurred.
sims	A list of simulation results, one for each period. Each list consists of a list for each data object, each of which consists of a list for each network, each of which consists of a list for each period, each component of which is an edgelist in matrix form (the columns are from, to, value) (or vector for behavior variables). Only if returnDeps is TRUE.

maxlikec returns a list containing:

fra	Simulated scores.
dff	Simulated Hessians: stored as lower triangular matrices
ntim	NULL, compatibility only
feasible	Currently set to TRUE.
OK	Could be set to FALSE if serious error has occurred.
dff	Simulated Hessian
sims	NULL, for compatibility only
chain	A list of sampled chains, one for each period. Each list consists of a list for each data object, each of which consists of a list for each network, each of which consists of a list for each period, each component of which is a list or a data frame depending on the value of returnDataFrame. Only if returnChainss is TRUE.
accepts	Number of accepted MH steps by dependent variable (permute steps are counted under first dependent variable)
rejects	Number of rejected MH steps by dependent variable (permute steps are counted under first dependent variable)
aborts	Number of aborted MH steps counted under first dependent variable.
loglik	Loglikelihood of the simulations. Only if returnLoglik is TRUE. If onlyLoglik is TRUE, only loglik, accepts,rejects and aborts are returned.

initializeFRAN and terminateFRAN return the control object z.

Author(s)

Ruth Ripley

References

See http://www.stats.ox.ac.uk/~snijders/siena/

See Also

siena07

Examples

```
mynet1 <- sienaNet(array(c(tmp3, tmp4), dim=c(32, 32, 2)))
mydata <- sienaDataCreate(mynet1)
myeff <- getEffects(mydata)
myeff <- includeEffects(myeff, transTrip)
myalgorithm <- sienaAlgorithmCreate(fn=simstats0c, nsub=2, n3=100)
ans <- siena07(myalgorithm, data=mydata, effects=myeff,  batch=TRUE)
```

---

summary.iwlsm *Summary method for Iterative Weighted Least Squares Models*

Description

summary method for objects of class "iwlsm"

Usage

```
## S3 method for class 'iwlsm'
summary(object, method = c("XtX", "XtWX"),
 correlation = FALSE, ...)
```

Arguments

object	the fitted model. This is assumed to be the result of some fit that produces an object inheriting from the class iwlsm, in the sense that the components returned by the iwlsm function will be available.
method	Should the weighted (by the IWLS weights) or unweighted cross-products matrix be used?
correlation	logical. Should correlations be computed (and printed)?
...	arguments passed to or from other methods.

Details

This function is a method for the generic function summary() for class "iwlsm". It can be invoked by calling summary(x) for an object x of the appropriate class, or directly by calling summary.iwlsm(x) regardless of the class of the object.

Value

If printing takes place, only a null value is returned. Otherwise, a list is returned with the following components. Printing always takes place if this function is invoked automatically as a method for the `summary` function.

`correlation`	The computed correlation coefficient matrix for the coefficients in the model.
`cov.unscaled`	The unscaled covariance matrix; i.e, a matrix such that multiplying it by an estimate of the error variance produces an estimated covariance matrix for the coefficients.
`sigma`	The scale estimate.
`stddev`	A scale estimate used for the standard errors.
`df`	The number of degrees of freedom for the model and for residuals.
`coefficients`	A matrix with three columns, containing the coefficients, their standard errors and the corresponding t statistic.
`terms`	The terms object used in fitting this model.

Author(s)

Adapted by Ruth Ripley

References

Venables, W. N. and Ripley, B. D. (2002) *Modern Applied Statistics with S*. Fourth edition. Springer.
See also http://www.stats.ox.ac.uk/~snijders/siena/

See Also

summary

Examples

```
## Not run:
##not enough data here for a sensible example, but shows the idea.
myalgorithm <- sienaAlgorithmCreate(nsub=2, n3=100)
mynet1 <- sienaDependent(array(c(s501, s502), dim=c(50, 50, 2)))
mynet2 <- sienaDependent(array(c(s502, s503), dim=c(50, 50, 2)))
mydata1 <- sienaDataCreate(mynet1)
mydata2 <- sienaDataCreate(mynet2)
myeff1 <- getEffects(mydata1)
myeff2 <- getEffects(mydata2)
myeff1 <- setEffect(myeff1, transTrip, fix=TRUE, test=TRUE)
myeff2 <- setEffect(myeff2, transTrip, fix=TRUE, test=TRUE)
myeff1 <- setEffect(myeff1, cycle3, fix=TRUE, test=TRUE)
myeff2 <- setEffect(myeff2, cycle3, fix=TRUE, test=TRUE)
ans1 <- siena07(myalgorithm, data=mydata1, effects=myeff1, batch=TRUE)
ans2 <- siena07(myalgorithm, data=mydata2, effects=myeff2, batch=TRUE)
meta <- siena08(ans1, ans2)
metadf <- split(meta$thetadf, meta$thetadf$effects)[[1]]
metalm <- iwlsm(theta ~ tconv, metadf, ses=se^2)
```

```
summary(metalm)

## End(Not run)
```

---

tmp3 *van de Bunt's Freshman dataset, time point 3*

Description

Third timepoint of van de Bunt's freshman dataset.

Codes: 1 = best friendship; 2 = friendship; 3 = friendly relationship; 4 = neutral relationship; 5 = troubled relationship; 0 = unknown person.

Format

Adjacency matrix for the network at time point 3.

Source

vrnd32t3.dat from http://www.stats.ox.ac.uk/~snijders/siena/vdBunt_data.zip

References

Van de Bunt, G.G., M.A.J. van Duijn, and T.A.B. Snijders (1999). Friendship networks through time: An actor-oriented statistical network model. *Computational and Mathematical Organization Theory*, 5, 167-192.
Also see http://www.stats.ox.ac.uk/~snijders/siena/vdBunt_data.htm.

See Also

tmp4

---

tmp4 *van de Bunt's Freshman dataset, time point 4*

Description

Fourth timepoint of van de Bunt's freshman dataset.

Codes: 1 = best friendship; 2 = friendship; 3 = friendly relationship; 4 = neutral relationship; 5 = troubled relationship; 0 = unknown person.

Format

Adjacency matrix for the network at time point 4.

Source

vrnd32t4.dat from http://www.stats.ox.ac.uk/~snijders/siena/vdBunt_data.zip

References

Van de Bunt, G.G., M.A.J. van Duijn, and T.A.B. Snijders (1999). Friendship networks through time: An actor-oriented statistical network model. *Computational and Mathematical Organization Theory*, 5, 167-192.
Also see http://www.stats.ox.ac.uk/~snijders/siena/vdBunt_data.htm.

See Also

tmp3

`updateTheta` *A function to update the initial values of theta, and a function to update an effects object.*

Description

`updateTheta` copies the final values of any matching selected effects from a `sienaFit` object to a Siena effects object.
`updateSpecification` includes in a Siena effects object a set of effects that are included in another effects object.

Usage

```
updateTheta(effects, prevAns, varName=NULL)
updateSpecification(effects.to, effects.from, name.to=NULL, name.from=NULL)
```

Arguments

`effects`	Object of class `sienaEffects`.
`prevAns`	Object of class `sienaFit` as returned by `siena07`.
`varName`	Character string or vector of character strings; is this is not `NULL`, the update will only applied to this dependent variable / these dependent variables.
`effects.to`	Object of class `sienaEffects`.
`effects.from`	Object of class `sienaEffects`.
`name.to`	Character string, name of dependent variable in `object.to`.
`name.from`	Character string, name of dependent variable in `object.from`.

Details

The initial values of any selected effects in the input effects object which match an effect estimated in prevAns will be updated by updateTheta. If the previous run was conditional, the estimated rate parameters for the dependent variable on which the run was conditioned are added to the final value of theta. If varName is not NULL, this update is restricted to effects for the dependent variable/s specified by varName.
By updateSpecification, the effects included in effects.from are also included in effects.to; if name.to and/or name.from is specified, this is restricted to effects for those dependent variables. If name.to = "all" (should then not be used as variable name!), the effects for all dependent variables will be updated.
Correspondence between effects is defined by "name", "shortName" "type", "groupName", "interaction1", "inte and "effect3". This means that inclusion of user-defined interactions will be updated only if they were available (i.e., defined) already in effects.to.

Value

Updated effects object.

Note

Using updateTheta explicitly before calling siena07 rather than using it via the argument prevAns of siena07 will not permit the use of the previous derivative matrix. In most cases, using siena07 with prevAns will be more efficient.

Author(s)

Ruth Ripley, Tom A.B. Snijders

References

See http://www.stats.ox.ac.uk/~snijders/siena/

See Also

siena07, getEffects

Examples

```
## For updateTheta:
mynet1 <- sienaDependent(array(c(tmp3, tmp4), dim=c(32, 32, 2)))
mydata <- sienaDataCreate(mynet1)
myeff1 <- getEffects(mydata)
myeff1 <- includeEffects(myeff1, transTrip)
myalgorithm <- sienaAlgorithmCreate(nsub=1, n3=100)
ans <- siena07(myalgorithm, data=mydata, effects=myeff1, batch=TRUE)
ans$theta
(myeff <- updateTheta(myeff1, ans))
##
## For updateSpecification:
myeff2 <- getEffects(mydata)
```

```
myeff2 <- includeEffects(myeff2, inPop)
updateSpecification(myeff2, myeff1)
# Create (meaningless) two-dimensional dependent network
mynet1 <- sienaDependent(array(c(s501, s502), dim=c(50, 50, 2)))
mynet2 <- sienaDependent(array(c(s503, s501), dim=c(50, 50, 2)))
mydata12 <- sienaDataCreate(mynet1, mynet2)
myeff12 <- getEffects(mydata12)
myeff.new <- getEffects(mydata12)
(myeff12 <- includeEffects(myeff12, inPop, outPop, outAct))
# update myeff.new only for mynet1:
updateSpecification(myeff.new, myeff12)
# update myeff.new for all dependent networks:
(myeff.updated <- updateSpecification(myeff.new, myeff12, "all"))
# use multivariate effects object to update univariate effects object:
myeff1 <- getEffects(sienaDataCreate(mynet1))
updateSpecification(myeff1, myeff.updated)
```

`varCovar` *Function to create a changing covariate object.*

Description

This function creates a changing covariate object from a matrix.

Usage

```
varCovar(val, centered=TRUE, nodeSet="Actors", imputationValues=NULL)
```

Arguments

`val`	Matrix of covariate values, one row for each actor, one column for each period.
`centered`	Boolean: if TRUE, then the overall mean value is subtracted.
`nodeSet`	Character string containing the name of the associated node set. If the entire data set contains more than one node set, then the node sets must be specified in all data objects.
`imputationValues`	Mtrix of covariate values of same dimensions as `val`, to be used for imputation of `NA` values (if any) in `val`. Must not contain any `NA`.

Details

When part of a Siena data object, the covariate is assumed to be associated with node set nodeSet of the Siena data object.
If there are any `NA` values in `val`, and `imputationValues` is given, then the corresponding elements of `imputationValues` are used for imputation. If `imputationValues` is `NULL`, imputation is by the overall mean value. In both cases, cases with imputed values are not used for calculating target statistics (see the manual).

Value

Returns the covariate as an object of class "varCovar", in which form it can be used as an argument to sienaDataCreate.

Author(s)

Ruth Ripley

References

See http://www.stats.ox.ac.uk/~snijders/siena/

See Also

sienaDataCreate, coCovar, coDyadCovar, varDyadCovar

Examples

```
myvarCovar <- varCovar(s50a)
```

---

varDyadCovar *Function to create a changing dyadic covariate object.*

Description

This function creates a changing dyadic covariate object from an array.

Usage

```
varDyadCovar(val, centered=TRUE, nodeSets=c("Actors", "Actors"),
    sparse=is.list(val), type=c("oneMode", "bipartite"))
```

Arguments

val	Array of covariate values, third dimension is the time. Alternatively, a list of sparse matrices of type "dgTMatrix".
centered	Boolean: if TRUE, then the overall mean value is subtracted.
nodeSets	Names (character string) of the associated node sets. If the entire data set contains more than one node set, then the node sets must be specified in all data objects.
sparse	Boolean: whether sparse matrices or not.
type	oneMode or bipartite: whether the matrix refers to a one-mode or a bipartite (two-mode) network.

Details

When part of a Siena data object, the covariate is assumed to be associated with the node sets named NodeSets of the Siena data object. The names of the associated node sets will only be checked when the Siena data object is created.

Value

Returns the covariate as an object of class "varDyadCovar", in which form it can be used as an argument to `SienaDataCreate`.

Author(s)

Ruth Ripley

References

See http://www.stats.ox.ac.uk/~snijders/siena/

See Also

`sienaDataCreate`, `coDyadCovar`, `coCovar`, `varCovar`

Examples

```
mydyadvar <- varDyadCovar(array(c(s501, s502), dim=c(50, 50, 2)))
```

Wald *Wald and score tests for RSiena results*

Description

These functions compute Wald-type and score-type tests for results estimated by siena07.

Usage

```
Wald.RSiena(A, ans)

Multipar.RSiena(ans, ...)

score.Test(ans, test=ans$test)
```

Arguments

A	A k * p matrix, where p = ans$pp, the number of parameters in ans excluding the basic rate parameters used for conditional estimation.
ans	An object of class sienaFit, resulting from a call to siena07.
...	One or more integer numbers between 1 and p, specifying the tested effects (numbered as in print(ans); if conditional estimation was used, numbered as the 'Other parameters').
test	One or more integer numbers between 1 and p, or a logical vector of length p; these should specify the tested effects (numbered as described for the ...).

Details

The hypothesis tested by Wald.RSiena is $A\theta = 0$, where θ is the parameter estimated in the process leading to ans.

The hypothesis tested by Multipar.RSiena is that all parameters given in ... are 0. This is a special case of the preceding.

The tested effects for score.Test should have been specified in includeEffects or setEffect with fix=TRUE, test=TRUE, i.e., they should not have been estimated. The hypothesis tested by score.Test is that the tested parameters have the value indicated in the effects object used for obtaining ans.

These tests should be carried out only when convergence is adequate (overall maximum convergence ratio less than 0.25 and all *t*-ratios for convergence less than 0.1 in absolute value).

Value

A list of three elements:

- chisquare: The test statistic, assumed to have a chi-squared null distribution.
- df: The degrees of freedom.
- pvalue: The associated *p*-value.

Author(s)

Tom Snijders

References

See the manual and http://www.stats.ox.ac.uk/~snijders/siena/

See Also

siena07

Examples

```
mynet <- sienaDependent(array(c(s501, s502), dim=c(50, 50, 2)))
mydata <- sienaDataCreate(mynet)
myeff <- getEffects(mydata)
myalgorithm <- sienaAlgorithmCreate(nsub=1, n3=40, seed=123)
# nsub=1 and n3=40 is used here for having a brief computation,
# not for practice.
myeff <- includeEffects(myeff, transTrip, transTies)
myeff <- includeEffects(myeff, outAct, outPop, fix=TRUE, test=TRUE)
(ans <- siena07(myalgorithm, data=mydata, effects=myeff, batch=TRUE))
A <- matrix(0, 2, 6)
A[1, 3] <- 1
A[2, 4] <- 1
Wald.RSiena(A, ans)
# A shortcut for the above is:
Multipar.RSiena(ans, 3, 4)
# The following two are equivalent:
score.Test(ans, c(FALSE, FALSE, FALSE, FALSE, FALSE, TRUE))
score.Test(ans,6)
# Getting all 1-df score tests separately:
# (More identifying information for the effects may be added as necessary)
for (i in which(ans$test)){
   sct <- score.Test(ans,i)
   cat(ans$requestedEffects$effectName[i], '\n',
   '    chisquared =', round(sct$chisquare,1),
   '; p =', round(sct$pvalue,3), '\n')}
```

`xtable` *Access xtable in package xtable*

Description

Dummy function to allow access to xtable in package xtable

Usage

```
xtable(x, ...)
```

Arguments

`x`	`sienaFit` object
`...`	Other arguments for `xtable.sienaFit`

Value

Value returned from `xtable.sienaFit`

Author(s)

Ruth Ripley

References

http://www.stats.ox.ac.uk/~snijders/siena/

See Also

xtable.sienaFit

Examples

```
## The function is currently defined as
function (x, ...)
{
    xtable::xtable(x, ...)
  }
## Not run:
myalgorithm <- sienaAlgorithmCreate(nsub=2, n3=100)
mynet1 <- sienaDependent(array(c(tmp3, tmp4), dim=c(32, 32, 2)))
mydata <- sienaDataCreate(mynet1)
myeff <- getEffects(mydata)
ans <- siena07(myalgorithm, data=mydata, effects=myeff, batch=TRUE)
ans
summary(ans)
xtable(ans, type="html", file="ans.html")
## End(Not run)
```

Index