Package ‘multiSiena’

June 18, 2024

Encoding UTF-8

Type Package

Title multiSiena - Simulation Investigation for Multilevel Empirical Network Analysis

Version 1.2.26

Date 2024-06-11

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

Depends R (>= 3.5.0), RSiena (>= 1.4.13)

Imports Matrix,
parallel,
MASS,
methods,
xtable

Suggests codetools

SystemRequirements GNU make

Description This package is a multilevel extension of the RSiena package for
simulation-based estimation of stochastic actor-oriented models for
longitudinal network data collected as panel data.
The extension consists of the function sienaBayes and associated functions.
The purpose is to fit hierarchical Bayesian models with random effects
to sienaGroup data objects.
An extensive manual, scripts, and much further information is at the Siena
website http://www.stats.ox.ac.uk/~snijders/siena.

License GPL-3

LazyData yes

BuildResaveData no

Biarch yes

NeedsCompilation yes

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

R topics documented:

`multiSiena-package` *Simulation Investigation for Multilevel Empirical Network Analysis*

Description

Package `multiSiena` is based on `RSiena`, and has the extra functionality of function `sienaBayes`.

Details

This package is a multilevel extension of the `RSiena` package for simulation-based estimation of stochastic actor-oriented models for longitudinal network data collected as panel data. The extension consists of the function sienaBayes and associated functions. The purpose is to fit hierarchical Bayesian models random effects to `sienaGroup` data objects.

It was built on RSiena 1.4.13.

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:	multiSiena
Type:	Package
Version:	1.2.26
Date:	2024-06-11
Depends:	R (>= 3.5.0)
Imports:	Matrix, parallel, MASS, methods
Suggests:	codetools
SystemRequirements:	GNU make
License:	GPL-3
LazyData:	yes
BuildResaveData:	no
NeedsCompilation:	yes

Author(s)

Ruth Ripley, Tom Snijders, Johan Koskinen.

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

References

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

Koskinen, J.H. and T.A.B. Snijders (2007). Bayesian inference for dynamic social network data. *Journal of Statistical Planning and Inference*, 13, 3930-3938.

Koskinen, J.H. and T.A.B. Snijders (2023). Multilevel Longitudinal Analysis of Social Networks. Journal of the Royal Statistical Society, Series A, 186, 376-400, DOI: https://doi.org/10.1093/jrsssa/qnac009

See Also

RSiena, sienaBayes

---

bayesTest *Tests for sienaBayes results with print and plot methods*

Description

These functions compute tests based on sienaBayesFit objects resulting from sienaBayes. Print and plot methods are available for the results of the multi-parameter test.

Usage

```
simpleBayesTest(z, nfirst=z$nwarm+1, tested0=0,
                probs = c(0.025,0.975), ndigits=4)
multipleBayesTest(z, testedPar, nfirst=z$nwarm+1, tested0=0, ndigits=4)
## S3 method for class 'multipleBayesTest'
print(x, descriptives=FALSE, ...)
## S3 method for class 'multipleBayesTest'
plot(x, xlim=NULL, ylim=NULL, main=NULL, ...)
```

Arguments

z	A sienaBayesFit object, resulting from a call to sienaBayes.
nfirst	The first element of the MCMC chain used for calculating properties of the chain; this can be the first element for which it is assumed that convergence has occurred.
tested0	The value to be tested; for simpleBayesTest this must be a number (applied to all coordinates), for multipleBayesTest it can be a number (applied to all coordinates) or a vector.
probs	A vector of two numbers between 0 and 1, the credibility limits for the credibility intervals.
ndigits	Number of digits to be given when representing the limits of the credibility intervals.
testedPar	A vector with the numbers of the parameters to be tested (numbered as in the output of simpleBayesTest), or a k * p matrix, for testing k linear combinations given by the rows, where p is the number of effects in the model except the basic rate parameters.
x	A multipleBayesTest object, resulting from a call to multipleBayesTest.
...	Extra arguments for print.multipleBayesTest and plot.multipleBayesTest.
descriptives	Boolean: whether to print the posterior mean, s.d., and covariance matrix of the parameters included in the test.
xlim	Parameter xlim in plot; if NULL, a sensible default will be used.
ylim	Parameter ylim in plot; if NULL, a sensible default will be used.
main	Parameter main in plot; if NULL, 'posterior distances'.

Details

Note that for the default values of probs and tested0, simpleBayesTest gives information also contained in print.sienaBayesFit.
To use multipleBayesTest, it is advisable first to execute simpleBayesTest to see the numbers that index the testedPar parameter; or its columns, if it is a matrix.
multipleBayesTest tests the hypothesis that the multivariate parameter for the effects defined by testedPar is equal to tested0. If testedPar is a set or vector of numbers then the parameters with these numbers are tested; if testedPar is a k * p matrix, the tested hypothesis is for the k linear combinations defined by its rows, and the number of columns p should be equal to the number of non-rate effects, which is the same as the number of effects listed by simpleBayesTest.
For the test, distances (for one parameter) or Mahalanobis distances (for more than one parameter) of the elements of the posterior sample from the posterior mean are calculated, and the p-value is the relative frequency that these are greater than the distance between the tested value and the posterior mean.
The p-values reported by simpleBayesTest are one-sided, and those reported by multipleBayesTest are two-sided for k=1, and 'all-sided' if k is larger than 1.
The plot presents a density plot of the posterior Mahalanobis distances, and the observed distance is indicated by a vertical line (if within the plot window).

Value

simpleBayesTest produces a data frame containing for the non-rate effects their names, an indication of whether or not they are varying parameters, and test results. For each parameter separately, (for the varying parameters) the population mean or (for the non-varying parameters) the value is tested. The test results are the posterior credibility intervals for the probability limits defined by probs, and the posterior probabilities of values larger than tested0.
multipleBayesTest produces an object of type multipleBayesTest which is a list with values:

prob	posterior p-value
chisquared	test statistic
postDistances	sample from posterior of distances from tested0
nullValue	input value of tested0
effectNames	names of effects tested as given in ...
posteriorSample	sample from posterior of tested effects

print.multipleBayesTest prints the posterior p-value with an explanation; and the proportions of all sign patterns (of deviations from 0) in the sample from the posterior.
plot.multipleBayesTest plots the density function of x$postDistances.

Author(s)

Tom Snijders

References

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

See Also

sienaBayes, print.sienaBayesFit

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))
  FourEffects <- getEffects(FourGroups)
  FourEffects <- includeEffects(FourEffects, transTrip)
  FourEffects <- setEffect(FourEffects, density, random=TRUE)
  FourEffects <- setEffect(FourEffects, recip, random=TRUE)
  print(FourEffects, includeRandoms=TRUE)
  # Note this also shows the "randomEffects" column.
  FourAlgo <- sienaAlgorithmCreate(projname = "FourGroups", maxlike=TRUE)
## Not run:
  bayes.model <- sienaBayes(FourAlgo, data = FourGroups,
        effects = FourEffects, nwarm=10, nmain=25, nrunMHBatches=10, saveFreq=0)
  bayes.model
  simpleBayesTest(bayes.model, probs = c(0.05,0.95))
  (mbt <- multipleBayesTest(bayes.model, c(2,3)))
  (mbt2 <- multipleBayesTest(bayes.model, c(2,3), tested0 = c(2,0.6)))
  plot(mbt)
  plot(mbt2)
  plot(mbt$posteriorSample)
  # Possible because the number of tested parameters is 2.
  A <- matrix(c(1,1,0),1,3)
  multipleBayesTest(bayes.model, A)

## End(Not run)
```

extract.sienaBayes *Extraction of posterior samples or posterior means from sienaBayes results*

Description

The first function extracts posterior samples from a list of `sienaBayesFit` object to be used, e.g., for assessing convergence.
The second function extracts posterior means and standard deviations per group from a `sienaBayesFit` object.

Usage

```
extract.sienaBayes(zlist, nfirst=zlist[[1]]$nwarm+1, extracted,
                     sdLog=TRUE)
extract.posteriorMeans(z, nfirst=z$nwarm+1, pmonly=1, excludeRates=FALSE,
                     verbose=TRUE)
```

Arguments

`zlist`	A list of `sienaBayesFit` objects, further called 'chains', resulting from calls to `sienaBayes` with a common data set and model specification.
`z`	A `sienaBayesFit` object.
`nfirst`	Integer: the first element for the first MCMC chain used for calculating properties of the chain.
`extracted`	The parameters for which posterior samples are to be extracted: `"all"`: all parameters; `"rates"`: all groupwise rate parameters; `"varying"`: all varying non-rate parameters: global means and standard deviations; `"non-varying"`: all estimated non-varying (and therefore, non-rate) parameters; `"objective"`: all non-rate parameters.
`sdLog`	Logical: should the logarithms be taken of the posterior standard deviations (applies only to varying parameters).
`pmonly`	if `pmonly=0` the posterior means and standard deviations are extracted, if `pmonly=1` only the posterior means, if `pmonly=2` only the posterior standard deviations.
`excludeRates`	Logical: whether to exclude the rate parameters.
`verbose`	Logical: gives some console output to show ongoing activity (may be reassuring for large number of groups).

Details

`extract.sienaBayes` produces a 3-dimensional array, iterations by chains by parameters, suitable for use, e.g., in function `monitor()` of package `rstan`. The 'iterations' are draws from the posterior distribution (provided that convergence was achieved), after the thinning in `sienaBayes` implied by its parameters `nrunMHBatches`, `nSampVarying`, `nSampConst` and `nSampRate`.
The number of iterations should be the same for all chains.
It is not checked that the specifications of the chains are identical. For the parameter names, the shortNames in the first chain are used.

`extract.posteriorMeans` produces the posterior means of the groupwise varying parameters. The resulting matrix has the groups in the rows. The effects are in the columns: for each effect comes first the posterior mean, then the posterior standard deviation.

Value

For `extract.sienaBayes` a 3-dimensional array, iterations by chains by parameters, the third dimension having the names of the parameters.
For `extract.posteriorMeans` a matrix with the groups in the rows and all effects in the columns, with for each effect two columns: first the posterior mean (effect name preceded by "p.m.") and then the posterior standard deviation (effect name preceded by "psd.").
If some names are duplicated (e.g., because of the presence of evaluation and creation effects), these can be changed by the user after creation of the array.

Author(s)

Tom Snijders

References

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

See Also

sienaBayes, plotPostMeansMDS

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))
  FourEffects <- getEffects(FourGroups)
  FourEffects <- includeEffects(FourEffects, transTrip)
  FourEffects <- setEffect(FourEffects, density, random=TRUE)
  FourEffects <- setEffect(FourEffects, recip, random=TRUE)
  print(FourEffects, includeRandoms=TRUE)
  algo0 <- sienaAlgorithmCreate(projname = NULL, seed=321,
        nsub=2, cond=FALSE, lessMem=TRUE)
  mom.model <- siena07(algo0, data = FourGroups, effects = FourEffects,
        batch=TRUE)
  # Note this also shows the "randomEffects" column.
  # Get three parallel estimates (the numbers of runs are way too low):
## Not run:
  bayes.models <- lapply(1:3, function(k){
    algok <- sienaAlgorithmCreate(projname = NULL, seed=321+(k*37))
    sienaBayes(algok, data = FourGroups,
        effects = FourEffects, prevAns=mom.model,
        nprewarm=5, nwarm=10, nmain=25, nrunMHBatches=2, nImproveMH=10)})
  bayes.extracted <- extract.sienaBayes(bayes.models, extracted="all")
  dim(bayes.extracted)
  dimnames(bayes.extracted)
  extract.posteriorMeans(bayes.models[[1]])
  extract.posteriorMeans(bayes.models[[2]])
  extract.posteriorMeans(bayes.models[[3]])

## End(Not run)
```

`print.sienaBayesFit` *Methods and functions for processing sienaBayes objects*

Description

print and summary methods for sienaBayesFit objects, and further functions for interpretation of results.

Usage

```
## S3 method for class 'sienaBayesFit'
print(x, nfirst=NULL, ...)

## S3 method for class 'sienaBayesFit'
summary(object, nfirst=NULL, allGroups=FALSE, ...)

## S3 method for class 'summary.sienaBayesFit'
print(x, nfirst=NULL, allGroups=FALSE, ...)

sienaBayes.table(x, nfirst=NULL, d=3,
    filename=paste(deparse(substitute(x)),'.tex',sep=""), align=TRUE)

shortBayesResults(x, nfirst=NULL)

plotPostMeansMDS(x=NULL, pm=NULL, pmonly=1, dim=2,  method=1, excludeRates=TRUE,
   nfirst=NULL, main=NULL, ...)
```

Arguments

`object`	An object of class `sienaBayesFit`.
`x`	An object of class `sienaBayesFit` or `summary.sienaBayesFit` as appropriate.
`nfirst`	The first element of the MCMC chain used for calculating properties of the chain; i.e., the first element for which it is assumed that convergence has occurred. If `NULL`, the value `x$nwarm+1` will be used.
`allGroups`	Logical: whether to print results for each group.
`pm`	In `plotPostMeansMDS`, at least one of `x` or `pm` should be given. If `pm` is not `NULL`, it should be an earlier obtained result of `extract.posteriorMeans`. If this is given, it overrides the values of `pmonly` and `excludeRates`.
`pmonly`	if `pmonly=0` a plot is made for posterior means and standard deviations: if `pmonly=1` only for the posterior means, if `pmonly=2` only for the posterior standard deviations.
`dim`	Dimension of the MDS representation. For `dim=1` the single dimension is plotted vertically, and the horizontal dimension is the sequence number of the group. For `dim` >= 3 only the first two dimensions are plotted.
`method`	Method for determining the similarities between the vectors of posterior means. See "Details".
`excludeRates`	Logical: whether to exclude the rate parameters for determining the coordinates.
`...`	For extra arguments. At present used only for further specification of the plot made by `plotPostMeansMDS`, allowing further arguments for `plot` and `text`.
`main`	Parameter `main` in `plot`; if `NULL`, "MDS" with name of `x`.
`filename`	filename for output; if "", printed to the console.
`d`	Number of decimals to be used in table.
`align`	Whether to align numbers at the decimal point.

Details

The function `print.sienaBayesFit` prints results (estimated posterior means, posterior standard deviations, and percentiles) of a Bayesian analysis, as resulting from function `sienaBayes`.
The function `summary.sienaBayesFit` prints more extensive results, including groupwise results. The columns "cred. from" and "cred. to" give the left and right end points of the estimated 95 percent credibility interval for the parameters, and the column "p" gives the estimated posterior probability that the parameter is greater than 0.

The function `plotPostMeansMDS` uses an MDS plot of the posterior means; if `pm` is not `NULL`, function `extract.posteriorMeans` will be used to compute it.
For `method=1` the dissimilarities are the sum of squared differences of distances between the posterior means normalized and orthogonalized with respect to a robustly estimated covariance matrix;
for `method=2` the dissimilarities are the sum of squared differences of distances between the posterior means divided by their standard deviations;
for `method=3` the dissimilarities are the sum of squared differences of distances between the posterior means.

Function `plotPostMeansMDS` uses random starting points, and running it will give a different result each time it is done. On convergence, it reports the minimized stress function. Since this usually will be a local minimum only, it is advisable to do it repeatedly and then interpret the plot with the lowest stress value.

Function `plotPostMeansMDS` can be useful for detection of outliers, or can signal that too many randomly varying effects are included in the model.

It is advisable to employ the `first` parameter; looking at trace plots will be helpful to estimate the number of runs from which convergence has occurred; this may well be different from the default value `x$nwarm+1`.

Function `siena.table` can also be used to make html or LaTeX tables of posterior means and standard deviations of a `sienaBayesFit` object.

Value

The function `shortBayesResults` produces a data frame containing posterior means and between-group standard deviations, and 95 percent crdibility intervals. This is meant to be a component for use in other functions.
Of this data frame, variables "name" to "effectName" are as in a `sienaEffects` object; variables "postMeanGlobal", "postSdGlobal", "cFrom", "cTo" are the posterior mean, posterior standard deviation, and lower and upper boundaries of the 95 percent posterior interval for the global parameters (`mu` and `eta`); variables "postSdBetween", "cSdFrom", "cSdTo" are the posterior mean, and similar percentiles, for the global between-group standard deviations (`sigma`) of the parameters that are randomly varying between groups.

Function `sienaBayes.table` is based on `shortBayesResults` and constructs a LaTeX tables of posterior means and posterior between-groups standard deviations.

The return value of `plotPostMeansMDS` is a list with three components:
`pm`, the matrix of posterior means;
`dist`, the dissimilarities between the groups, used for the MDS;
`points`, the matrix of `dim`-dimensional MDS coordinates.

Author(s)

Tom Snijders

References

See also http://www.stats.ox.ac.uk/~snijders/siena/
For plotPostMeansMDS: for the MDS method see p. 308 and for the robust covariance matrix see p. 336 of
W.N. Venables and B.D. Ripley (2002), *Modern Applied Statistics with S.* Fourth edition. Springer.

See Also

siena.table, sienaBayes, extract.sienaBayes, extract.posteriorMeans.

Examples

```
# Construct a list of Siena data sets of smallNetworks
# together with group-level variable "num", the rank number in 1:50
mData <- list()
for (i in 1:20){
    net <- sienaDependent(array(c(smallNetworks[[1]],smallNetworks[[i+1]]),
                           dim=c(5,5,2)), allowOnly=FALSE)
    num <- coCovar(rep(i,5), warn=FALSE)
    mData[[i]] <- sienaDataCreate(net,num)
}
# Construct the multigroup dataset:
(multiData <- sienaGroupCreate(mData))
# specify model
eff.p <- getEffects(multiData)
eff.p <- setEffect(eff.p, density, random=TRUE)
eff.p <- setEffect(eff.p, recip, random=TRUE)
eff.p <- setEffect(eff.p, transTrip, random=TRUE)
eff.p <- setEffect(eff.p, egoX, interaction1='num')
eff.p <- includeInteraction(eff.p, egoX, recip, interaction1=c('num',''))
eff.p <- includeInteraction(eff.p, egoX, transTrip, interaction1=c('num',''))
print(eff.p, includeRandoms=TRUE, dropRates=TRUE)
alg.ml <- sienaAlgorithmCreate(projname=NULL, seed=1234)
# Prior distribution; for rho it does not matter
# (because priorRatesFromData=2)
Mu <- c(1,-1,0.5,0.2)
Sig <- matrix(0,4,4)
diag(Sig) <- 0.09
# Estimate (the numbers of runs indicated are way too low):
bayes.model <- sienaBayes(alg.ml, data=multiData,
       effects=eff.p, nprewarm=10, nwarm=25, nmain=50, nrunMHBatches=2,
       initgainGroupwise=0, reductionFactor=0.1,
       priorMu=Mu, priorSigma=Sig, priorKappa=0.09,
       silentstart=TRUE, saveFreq=0)
# If one would assume that convergence was reached by iteration 30 out of 75:
print(bayes.model, nfirst=30)
summary(bayes.model)
siena.table(bayes.model, type="html", d=3)
shortBayesResults(bayes.model)
pm <- extract.posteriorMeans(bayes.model)
ppm <- plotPostMeansMDS(bayes.model, pm=pm)
```

`sienaBayes` *A function for fitting Bayesian models*

Description

A function to fit hierarchical Bayesian models random effects to sienaGroup data objects. Uses the function `maxlikec` for the SAOM part, the Bayesian part is performed in R.

Usage

```
sienaBayes(data, effects, algo,  saveFreq=100,
    initgainGlobal=0.1, initgainGroupwise = 0.001, initfgain=0.2, gamma=0.05,
    initML=FALSE, priorMeanEta=NULL, priorSigEta=NULL,
    priorMu=NULL, priorSigma=NULL, priorDf=NULL, priorKappa=0,
    priorRatesFromData=2,
    frequentist=FALSE, incidentalBasicRates=FALSE,
    reductionFactor=0.5, delta=1e-10,
    nprewarm=50, nwarm=50, nmain=250, nrunMHBatches=20,
    nSampVarying=1, nSampConst=1, nSampRates=0,
    nImproveMH=100, targetMHProb=0.25,
    lengthPhase1=round(nmain/5), lengthPhase3=round(nmain/5),
    storeScores = FALSE,
    storeAll=FALSE, prevAns=NULL, usePrevOnly=TRUE,
    prevBayes=NULL, newProposalFromPrev=(prevBayes$nwarm >= 1),
    silentstart = TRUE,
    nbrNodes=1, clusterType=c("PSOCK", "FORK"),
    getDocumentation=FALSE)

glueBayes(z1, z2, nwarm2=0)
```

Arguments

`data`	A sienaGroup object as returned by `sienaGroupCreate`. It is planned to also allow a `siena` data object as returned by `sienaDataCreate`.
`effects`	sienaEffects object as returned by `getEffects(data)`. The effects indicated by its column `randomEffects` get effects that are varying across the groups.
`algo`	Algorithm object, as created by `sienaAlgorithmCreate`. Should contain all options required for the MCMC scheme, and a random seed if required.
`saveFreq`	Integer. If this is larger than 1, the provisional results are saved after each multiple of saveFreq iterations in the main phase, in a file with name PartialBayesResult.RData (if a file with this name exists, it will be overwritten). This file contains an object `z` of class `sienaBayesFit`, with the provisional results. This is to guard against crashes or power failure. It can be used as value for `prevBayes` as indicated below.
`initgainGlobal`	Step sizes in initial searches for good parameter values across the groups.
`initgainGroupwise`	Step sizes in initial searches for good parameter values by group; can be up to 0.1 for larger networks, 0 for very small networks.

`initfgain`	Positive number, used only for frequentist estimation and incidentalBasicRates: the gain factor in the Robbins Monro algorithm is `initfgain` during warming and Phase 1, and `initfgain * ((iteration number after Phase 1)^(-gamma))` after that.
`gamma`	Positive number, used only for frequentist estimation: see `initfgain`.
`initML`	Boolean, whether to use maximum likelihood estimation for the groupwise initial estimation.
`priorMeanEta`	Vector of length equal to the number of fixed parameters, or `NULL`. Prior mean of `eta` (fixed effects); default: 0.
`priorSigEta`	Vector of length equal to the number of fixed parameters, or `NULL`. If not `NULL`, values that are not `NA` are prior variances of `eta` (fixed effects); these must all be positive.
`priorMu`	Vector of length equal to the number of randomly varying parameters, or `NULL`. Prior mean of `mu` (global population mean for varying parameters); default: 0. For the basic rate parameters the defaults are data-dependent, but if ((`priorRatesFromData = 0`) and `priorMu=NULL`), for these parameters the prior mean will be 2.
`priorSigma`	Square matrix of dimension equal to length of `priorMu`, or `NULL`. Prior between-groups population covariance matrix `Sigma` for the varying parameters; default: identity matrix.
`priorDf`	Prior degrees of freedom for `Sigma` (global population covariance matrix); default: number of randomly varying parameters + 2.
`priorKappa`	Proportionality constant between prior covariance matrix and covariance matrix of prior distribution for `mu`; default: 0 (then the prior for mu is improper, and `priorMu` has no effect, except for the rate parameters if `priorRatesFromData=0`).
`priorRatesFromData`	-1, 0, 1, or 2. Determines the prior distribution for the rate parameters. -1: all basic rate parameters are fixed; this must correspond to `effects`; 0: prior is taken from `priorMu` and `priorSigma`; 1: prior is defined by mean and covariance matrix of estimated rate parameters from initialization phase; 2: prior is defined by robust estimates for location and multivariate scale of estimated rate parameters from initialization phase.
`frequentist`	Currently only `frequentist=FALSE` works. Boolean: chooses between frequentist or Bayesian estimation of the global parameters. Frequentist estimation is possible only for at least 2 groups.
`incidentalBasicRates`	Boolean. If this is `TRUE`, the basic rate parameters are defined specifically for each group, and estimated using a Robbins Monro algorithm; if `FALSE`, they have a common prior for all groups.
`reductionFactor`	Positive number. If `priorRatesFromData` = 1 or 2, the prior distribution for the rate parameters is the estimated covariance matrix of the estimated rate parameters in the initialization phase multiplied by `reductionFactor`, plus a contribution of 0.01 for the variances. For many data sets the default will be OK. If the rate parameters diverge, as evidenced by the traceplots of the posterior values, then a smaller value is advisable, such as 0.01. Has no effect if `priorRatesFromData` <= 0.
`delta`	When the global population covariance matrix becomes non-positive definite (i.e., has one or more negative correlations) during iterations, it is changed so that all eigenvalues are at least delta.

`nprewarm`	Number of iterations in the pre-warm-up phase, before `improveMH`.
`nwarm`	Number of iterations in the warm-up phase. Used only if `prevBayes` is `NULL`. Then it should be at least 5.
`nmain`	Number of iterations in the main phase. Should be at least 10.
`nrunMHBatches`	Integer: thinning ratio in MCMC process; but thinning at the lowest level is further determined by parameter `mult` in `algo`, see `sienaAlgorithmCreate`. Should be even.
`nSampVarying`	Number of samples of varying parameters for each chain sample.
`nSampConst`	Number of samples of constant parameters (`"eta"`) for each chain sample.
`nSampRates`	Number of extra samples of basic rate parameters for each chain sample.
`nImproveMH`	Number of iterations per `improveMH` step. If `nImproveMH=0`, no `improveMH` steps at all. It is recommended to use the default value.
`targetMHProb`	Desired proportion of acceptances in MH steps. This can be one number, or a vector of two numbers. In the latter case, the first number applies to the MH steps per group (for each group), the second to the MH steps for the constant parameters (`"eta"`).
`lengthPhase1`	Only used for frequentist estimation or `incidentalBasicRates`: length of the first phase of the Robbins Monro algorithm. `lengthPhase1` + `lengthPhase3` should be strictly less than `nmain`.
`lengthPhase3`	Only used for frequentist estimation or `incidentalBasicRates`: length of the third phase of the Robbins Monro algorithm. `lengthPhase1` + `lengthPhase3` should be strictly less than `nmain`.
`storeScores`	Boolean: whether to store thinned scores.
`storeAll`	Boolean: whether to store parameters for all MCMC iterations, i.e., before thinning. `storeAll=TRUE` may lead to producing very large objects and is not recommended for usual operation.
`prevAns`	An object of class "sienaFit" as returned by `siena07` for the same data set and effects object. This then is the result of a multi-group estimation for these data, from which scaling information (derivative matrix and standard deviation of the deviations) can be extracted along with the parameters estimates which will be used as the initial values, unless `algo` requests the use of standard initial values. If `prevAns=NULL`, then a multi-group estimation using `siena07` will be performed as part of the initialization. If `prevAns` is an object as described, estimated with unconditional estimation (`cond=FALSE`), with 500 or more runs in Phase 3, then if `usePrevOnly=TRUE` this multi-group estimation will be skipped; if `usePrevOnly=FALSE` a multi-group estimation still will be performed, but starting at the results provided by `prevAns`; if `usePrevOnly=TRUE` but with less than 500 runs in Phase 3, then the multi-group estimation will be done only for Phase 3 (skipping Phases 1 and 2).
`usePrevOnly`	Boolean: see immediately above.
`prevBayes`	An object of class `sienaBayes` as returned by function `sienaBayes`, on which the current function will continue. For example, the object `z` contained in the intermediate saved result `PartialBayesResult.RData` as mentioned above. Initialization and warming phases are skipped. If these are given, the values of `nrunMHBatches`, `nSampVarying`, `nSampConst`, and `nSampRates` supersede those in `prevBayes`. If these are given, the values of `prevAns`, `nwarm`, `incidentalBasicRates`, `priorRatesFromData`, and the prior distributions are disregarded.

	A new set of `nmain` iterations in the main phase are done. All further relevant parameters of the function call should be identical. This is not completely checked, so errors may occur if this is not the case.
`newProposalFromPrev`	Boolean, has consequences only when `prevBayes` is given: whether to use a robust estimate of the covariance matrix of parameters in `prevBayes` to define a new proposal distribution. Else the proposal distribution is the same as in the `prevBayes` object.
`silentstart`	Boolean: whether to suppress most information to the console during the calculation of initial values.
`nbrNodes`	Number of processes to be used. Cannot be more than the number of periods summed over number of groups.
`clusterType`	If using multiple processes, whether to use forking processes or not. (Only "PSOCK" can be used on Windows.)
`getDocumentation`	Flag to allow documentation of internal functions, not for use by users.
`z1`	`sienaBayes` object.
`z2`	`sienaBayes` object with the same data and model specification as `z1`.
`nwarm2`	Number of warming iterations in `z2`; the first `nwarm2` iterations of `z2` will be left out of the combined object.

Details

The function `sienaBayes` is for Bayesian estimation of one group or of multiple groups all having the same number of waves and the same model specification. It wraps Bayesian sampling of parameters around calls to `maxlikec`. The RSiena manual has a lot of information.
Effects can be randomly varying between groups, or global (i.e., constant across groups). This is indicated by the keyword `random` in the call of `setEffect`, and reported when using the keyword `includeRandoms` in `print.sienaEffects`.
For the groupwise parameters normal distributions are assumed with conjugate priors. The conjugate prior is an inverse Wishart distribution for the covariance matrix `Sigma`, with parameters `Lambda^{-1}` and `priorDf`, where `Lambda` = `priorDf*priorSigma`; and, if `priorKappa` > `0`, conditional on `Sigma`, for the expected value a multivariate normal with mean `priorMu` and covariance matrix `Sigma/priorKappa`; if `priorKappa` = `0`, the expected value has an improper prior.
For the fixed parameters, if `priorSigEta=NULL`, the improper constant prior is used. Else, for the components of `eta` (fixed effects parameter) for which `priorEta` is `NA`, the improper constant prior is used, and for other components, independent normal priors with mean 0 and variances given by `priorEta`.
Recommendation: use the normal prior only for effects of group-level variables. If a non-zero prior mean is desired, transform the group-level variable.
The required dimensions of the prior parameters `priorSigEta`, `priorMu` and `priorSigma` depend on the number of groupwise varying parameters and are given in the output for `print.sienaEffects` when using the keyword `includeRandoms` .
If `priorRatesFromData` is 1 or 2, the prior distribution for the basic rate parameters is determined in a data-dependent way. For the non-varying parameters, a flat prior is assumed.
The frequentist option currently is not supported (probably broken).
The procedure consists of three parts: initialization, warming, main phase.
In the initialization phase, initial parameter values and the proposal covariance matrix for Metropolis-Hastings steps for groupwise parameters are obtained from, first, Method of Moments estimation of a parameter vector assumed to be the same across the groups (in a multi-group estimation); replacing this estimate by a precision-weighted mean of this estimate and the prior mean; followed by one

subphase of the Robbins-Monro algorithm for Method of Moments estimates for the groups separately, starting from the overall estimate, with step size `initgain`. This is skipped if `initgain=0`.
If the `prevBayes` object is supplied, this initialization phase is skipped, and if `newProposalFromPrev` the proposal covariance matrices are calculated from the simulated chains in this object.
From this basis, `nprewarm` Metropolis Hastings steps are taken; however, if no acceptances occur in two consecutive steps, the prewarming phase is ended. The proposal covariance matrices then are scaled, in the function '`improveMH`', to achieve a proportion of accepted MH proposals approximately equal to `target`.
After initialization and scaling of the proposal covariance matrices, a warming phase is done of `nwarm` Bayesian proposals each with a number of MH steps, followed again by the function '`improveMH`'. If the `prevBayes` object is supplied, the warming phase is skipped.
Finally `nmain` repeats of `nrunMHBatches` are performed of a number of MH steps sampling chains, plus `nSampVarying` MH steps sampling the varying parameters ('`theta_j`') plus `nSampConst` MH steps sampling the non-varying parameters ('`eta`') plus one Gibbs step sampling the global mean and covariance matrix of the varying parameters ('`mu`' and '`Sigma`'). In the warming as well as the final phase, the number of MH steps within each run is determined by parameter `mult` ("multiplication factor") in the algorithm object `algo`.

The function is time-consuming. When starting to use it, it is advisable to start with low values of `nmain` to explore computing time.
Initialisation is an important part, and will be shorter if `prevAns` is given. If an earlier `sienaBayes` object is available for this data specification (which may differ in which effects are specified as being random), then if the name of this object is `zz` it will be possible to use `prevAns=zz$initialResults`.

When the procedure seems to diverge, and for very small groups, it is advisable to use smaller values of the parameters `initgainGlobal` and `initgainGroupwise`.

`glueBayes(z1,z2)` combines two existing `sienaBayesFit` objects with the same data and model specifications (this is not entirely checked) into one by putting `z2` after `z1`. This is intended to be used when `z2` was produced by calling `sienaBayes` with `prevBayes=z1`.

For the function `print.sienaEffects`, the options `includeRandoms=TRUE` and `dropRates=TRUE` are meant to be useful especially for effects objects used for `sienaBayes`.

Value

`sienaBayes` as well as `glueBayes` return an object of class `sienaBayesFit`. This is a list containing, among other things:

`priorMu`	prior global population mean (not quite the same as corresponding input parameter)
`priorSigma`	prior global between-groups population covariance matrix (not quite the same as corresponding input parameter)
`priorKappa`	proportionality constant between prior covariance matrix and covariance matrix of prior distribution for the mean
`priorDf`	prior degrees of freedom for covariance matrix
`nmain`	Number of iterations used in the main phase, as given in the call of `sienaBayes`
`nwarm`	Number of iterations used in the warm-up phase, as given in the call of `sienaBayes`; 0 if `prevBayes` was used. The value of `nwarm` is used in other functions such as `print.sienaBayesFit`, and when the object is used for `sienaBayes` as a `prevBayes` parameter with `newProposalFromPrev=TRUE`. If it was deemed that converge begins later (or earlier) than the value of `nwarm`, this value can be changed for further use of the object produced by `sienaBayes`.
`effectName`	array of names of effects included in the model

`f$groupNames`	array of names of groups included in the model
`initialResults`	sienaFit object: result of initial MoM estimation under the assumption of same parameters across groups
`ThinParameters`	array of dimensions (`nwarm+nmain` iterations by groups by parameters): sampled groupwise parameters
`ThinPosteriorMu`	array of dimensions (`nwarm+nmain` iterations by parameters): sampled global means of varying parameters
`ThinPosteriorEta`	array of dimensions (`nwarm+nmain` iterations by fixed parameters): sampled fixed parameters
`ThinPosteriorSigma`	array of dimensions (`nwarm+nmain` iterations by parameters by parameters): sampled global covariance matrix of varying parameters
`ThinnedScores`	if `storeScores=TRUE`: array of dimensions (`nwarm+nmain` iterations by non-rate parameters): sampled scores
`nrunMH`	vector (by groups * waves) of number of MH steps in the innermost loop of the likelihood simulations of the SAOM; this is a data-dependent multiple of the value of `mult` set in `sienaAlgorithmCreate`
`ThinBayesAcceptances`	matrix of number of acceptances in the `nrunMHBatches` MH steps in each iteration, and summed over dependent variables, for each group and (duplicated) for the vector of fixed effects
`acceptances`	if `storeAll=TRUE`: matrix of booleans: whether the corresponding change to the parameters was accepted in each step, by group
`MHacceptances`	if `storeAll=TRUE`: array of acceptances of the MH steps, by step type and group but summed over dependent variables
`MHrejections`	if `storeAll=TRUE`: array of rejections of the MH steps
`MHproportions`	if `storeAll=TRUE`: array of proportions of the MH steps accepted

Author(s)

Ruth Ripley, Johan Koskinen, Tom Snijders

References

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

Koskinen, J.H. and T.A.B. Snijders (2007). Bayesian inference for dynamic social network data. *Journal of Statistical Planning and Inference*, 13, 3930-3938.

Koskinen, J.H. and T.A.B. Snijders (2023). Multilevel Longitudinal Analysis of Social Networks. Journal of the Royal Statistical Society, Series A, 186, 376-400, DOI: `https://doi.org/10.1093/jrsssa/qnac009`

See Also

`extract.sienaBayes`, `plotPostMeansMDS`, `smallNetworks`
There are print and summary methods for sienaBayesFit objects, see `print.sienaBayesFit`.

Examples

```
# Construct a list of Siena data sets of smallNetworks
# together with group-level variable "num", the rank number in 1:50
mData <- list()
for (i in 1:50){
    net <- sienaDependent(array(c(smallNetworks[[1]],smallNetworks[[i+1]]),
                            dim=c(5,5,2)), allowOnly=FALSE)
    num <- coCovar(rep(i,5), warn=FALSE)
    mData[[i]] <- sienaDataCreate(net,num)
}
# Construct the multigroup dataset:
(multiData <- sienaGroupCreate(mData))
# specify model
eff.p <- getEffects(multiData)
eff.p <- setEffect(eff.p, density, random=TRUE)
eff.p <- setEffect(eff.p, recip, random=TRUE)
eff.p <- setEffect(eff.p, transTrip, random=TRUE)
eff.p <- setEffect(eff.p, egoX, interaction1='num')
eff.p <- includeInteraction(eff.p, egoX, recip, interaction1=c('num',''))
eff.p <- includeInteraction(eff.p, egoX, transTrip, interaction1=c('num',''))
print(eff.p, includeRandoms=TRUE, dropRates=TRUE)
alg.ml <- sienaAlgorithmCreate(projname=NULL, prML=2, seed=531)
# Prior distribution; for rho it does not matter
# (because priorRatesFromData=2)
Mu <- c(1,-1,0.5,0.2)
Sig <- matrix(0,4,4)
diag(Sig) <- 0.09
## Not run:
# Estimate: (the numbers of runs indicated are way too low)
(bayes.model <- sienaBayes(alg.ml, data=multiData,
       effects=eff.p, nprewarm=10, nwarm=25, nmain=100, nrunMHBatches=10,
       initgainGroupwise=0, reductionFactor=0.1,
       priorMu=Mu, priorSigma=Sig, priorKappa=0.09,
       silentstart=FALSE, saveFreq=0))

## End(Not run)
```

`simulateData` *Forward simulation of data sets from sienaBayes results*

Description

This function performs forward simulations of `sienaBayesFit` results to be used, e.g., for assessing goodness of fit.
Such simulations are intended to be a sample from the posterior predictive distribution.

Usage

```
simulateData(g, x, xd, thin=1, nstart=x$nwarm,
                               seed=NULL, nbrNodes=1)
```

Arguments

g	Integer: group number.
x	A sienaBayesFit object.
xd	The sienaGroup data set corresponding to x.
thin	Integer: a parameter for thinning.
nstart	The first iteration in x, starting from which parameter values will be used.
seed	Integer: random number seed.
nbrNodes	Number of parallel processes to use.

Details

This function takes groupwise simulated parameter values from x for group g, and simulates data for this group for the sequence of these parameters. This is done by running siena07 with returnDeps=TRUE, using as thetaValues the parameters simulated in x, and using an algorithm object with nsub=0, simOnly=TRUE.
Works only if there are no interactions between three effects; and only one dependent network or behavior variable.

Value

sienaFit object, produced using only Phase 3 of siena07, containing a sample from the posterior predictive distribution.
The simulated dependent variables are in the component sims of the object produced by simulateData.
The size of the generated sample is parameter nmain in the call of sienaBayes used to generate x, divided by thin.
See the help page for siena07 for the structure of the simulated dependent variables.

Author(s)

Tom Snijders

See Also

sienaBayes

Examples

```
  mData <- list()
  for (i in 1:10){
    net <- sienaDependent(array(c(smallNetworks[[1]],smallNetworks[[i+1]]),
                             dim=c(5,5,2)), allowOnly=FALSE)
    num <- coCovar(rep(i,5), warn=FALSE)
    mData[[i]] <- sienaDataCreate(net,num)
  }
# Construct the multigroup dataset:
  (multiData <- sienaGroupCreate(mData))
# specify model
  eff.p <- getEffects(multiData)
  eff.p <- setEffect(eff.p, density, random=TRUE)
  eff.p <- setEffect(eff.p, recip, random=TRUE)
  print(eff.p, includeRandoms=TRUE, dropRates=TRUE)
  alg.ml <- sienaAlgorithmCreate(projname=NULL, prML=2, seed=531, lessMem=TRUE)
```

```
    Mu <- c(1,-1,0.5)
    Sig <- matrix(0,3,3)
    diag(Sig) <- 0.25
  # Estimate: (warning: the settings of the algorithm are not realistic)
    (bayes.model <- sienaBayes(alg.ml, data=multiData,
         effects=eff.p, nprewarm=5, nwarm=25, nmain=25, nrunMHBatches=2,
         nImproveMH=0,
         initgainGroupwise=0, reductionFactor=0.1,
         priorMu=Mu, priorSigma=Sig, priorKappa=0.09,
         silentstart=TRUE, saveFreq=0))
    (pred.post <- simulateData(1, bayes.model, multiData, seed=321))
    length(pred.post$sims) # set by nmain in call of sienaBayes
  # pred.post$sims contains the simulated networks as edge lists:
    pred.post$sims[[25]]
```

`smallNetworks` *Network data: list of generated small networks.*

Description

A list of 51 small networks of 5 actors. The first is meant as the initial network; the next 50 were generated with a model including reciprocity and increasing values of the transitive triplets parameter.

Format

A list of 51 adjacency matrices.

Author(s)

Tom Snijders

See Also

sienaBayes

Examples

```
length(smallNetworks)
smallNetworks[[1]]
# some descriptives
totalTies <- sapply(smallNetworks, sum)
distances <- sapply(smallNetworks, function(x){sum(abs(x-smallNetworks[[1]]))})
table(totalTies, distances)
```

Index