Multilevel longitudinal network estimation using sienaBayes

Bayesian SAOM

Johan Koskinen and T.A.B. Snijders

Department of Statistics
Stockholm University

June 24, 2025

Stockholm
University

All Material:
https://github.com/johankoskinen/SunbeltXVLHSAOM.git

Traditionally

SNA case studies of single networks

To generalise, preferably

a population of networks

AIM: population of networks

multilevel network analysis in the sense of analyzing
multiple similar networks, mutually independent.

This was proposed by Snijders & Baerveldt (*J. Math. Soc. 2003*)[1]

---

[1]Also, see Entwisle, Faust, Rindfuss, & Kaneda (*AJS*, 2007) for review of empirical work involving multiple networks

Sample from Population of Networks

We assume:

- Suppose we have a samples $j = 1, \ldots, N$
- from a population of N networks on **disjoint** node sets
- generated from same model
- but having different parameters $\theta_1, \ldots, \theta_j, \ldots, \theta_N$

We can call each network j a group.
We assume we have network panel data for each group,
and wish to analyze these by fitting
Stochastic Actor-Oriented Models ('*SAOMs*', RSiena).

Random effects: hierarchical multilevel structure

1. On the global level there is a population of networks
   with either a multivariate normal distribution $\mathcal{N}(\mu, \Sigma)$
   for the randomly varying parameters,
   and a common parameter η for the rest;
   or without distribution assumption.

2. On the network level there is a Stochastic Actor-Oriented Model
   (SAOM) with parameter vector $(\theta_j^{(1)}, \eta)'$ for group j.
   Here $\theta_j^{(1)} \sim \mathcal{N}(\mu, \Sigma)$.

3. On the tie level there is a dynamic process
   governed by the SAOM.

Hierarchical multilevel analysis

Assumption: population of networks;

$\theta_j \sim$ multivariate normal distribution,
perhaps conditionally on network-level covariates.

1. mixed effects:
some parameters varying, others constant across groups:
    restricted integrated hierarchical approach.

2. random effects:
all parameters varying across groups:
    integrated hierarchical approach.

multiSiena: *multi-level option*, `sienaBayes()`
available from Siena website.

Integrated procedure:
Estimate the *distribution* of θ_j and consider the
'posterior' (*predictive*) distribution of θ_j given the data.

Advantage:
The analysis of the separate networks draws strength from
the total sample of networks by regression to the mean.

Useful especially for many small networks.

Crash-course in Bayesian inference?

Separate slides on introduction to Bayesian inference

Hierarchical model: restricted variation

Terminology here:

Fixed parameters η

Fixed parameters η have the same value across groups

Random parameters $\theta_j^{(1)}$

Random parameters have different values

$$\theta_1^{(1)}, \theta_2^{(1)}, \ldots, \theta_N^{(1)}$$

across groups,
with a priori a normal distribution.

From regular hierarchical linear models (*HLMs*) we know
that 'random slopes' require a lot from the data.

In practice, HLMs have only a few random slopes.

For `SienaBayes`, however, a large number of random parameters seems[2] not to be a problem.

But note the important distinction:

For fixed parameters (η),
there is much more information than
for the means and variances of random parameters (μ, Σ).

Therefore, the posterior distribution will be more concentrated (smaller variance) for the η parameters than for μ.

For the purpose of *testing*, effects that are
the focus of investigation therefore may be
treated as fixed (which they are under the null).

---

[2]Tom's words

Fixed v random effect

Fixed effect k

Testing $\eta_k = 0$ means testing the null hypothesis

$$H_0 : \theta_{jk} = 0$$

in *every* group j, under the auxiliary assumption (AA) that

$$\theta_{jk} = \eta_k, \text{ for all } j = 1, \dots, N$$

Random effect k

Testing $\mu_k = 0$ means testing the null hypothesis

$$\mathbb{E}(\theta_{jk}) = 0, \text{ for all } j = 1, \dots, N$$

that the population average of the is 0.

AA $\Rightarrow$ posterior sd η_k smaller than for μ_k, unless small prior variance for μ_k

Group-level variables

To use group-level variables:
Defined as constant actor covariates (coCovar),
with the same value for all actors.
It is helpful for convergence to center the group-level variables.

Tom recommends that you center such variables "by hand"[3], by
subtracting their mean (or approximate mean) before executing coCovar.

---

[3]He's not sure default centering in coCovar survives the operation of
sienaGroupCreate

Varying group sizes

Network models are not comparable across different network sizes (here group sizes)
Think of how *density does not scale*!
To account for different group sizes:

- use the group-level covariate with the value $\log(n_j)$,
- where $\log(\cdot)$ is the natural logarithm and n_j group size;
- or check with sienaGOF that $\log(n_j)$ does not matter.

Varying group sizes (2)

If group size is available as a group-level variable n,
with mean across groups given by avlogn,
this can be done with a script line such as

```
logn <- coCovar(rep(log(n) - avlogn, n),
                centered=FALSE, warn=FALSE)
```

This can be used as a main effect (egoX), and in interaction with
reciprocity and the effect representing transitivity; perhaps also with other
effects?
Since the incomparability is for creation of new ties, the type of these
effects can be creation. If these interactions are 'not significant', they may
be dropped.

The Bayesian MCMC procedure produces,
when/if there is convergence (i.e., hopefully, after a burn-in period),
a sample from the posterior distribution of all the parameters,
both the θ_j referring to the individual sampled networks,
and η, μ and Σ referring to the population of networks.

The inference is based on these
sampled posterior distributions.

Two kinds of plot are important:

1. *MDS plots*, of the group-wise posterior means, indicating possible outliers,

2. *trace plots*, representing successive draws from the posterior distribution (after *thinning*),

3. *density plots*, representing the plausible values of the parameters, given the observed data.

Prior distributions

For a smallish number of groups, the prior is consequential.
It is given to SienaBayes as priorMu, priorSigma, priorDf,
priorKappa (and priorMeanEta, priorSigEta)

For rate parameters, SienaBayes uses a data-dependent prior
(depending on the option chosen).

The four (six) prior parameters are explained in the following pages.

Model and prior specification

Model: for all groups j independently, $\theta_j^{(1)} \sim \mathcal{N}(\mu, \Sigma)$.

Prior: the prior distribution for (μ, Σ) is the
inverse Wishart distribution for Σ;
and, conditional on Σ, for μ a multivariate normal distribution with mean
μ_0 and covariance matrix Σ/κ_0. In formula:

$$\Sigma \sim \mathrm{InvWishart}_{p_1}(\Lambda_0^{-1}, \mathrm{df}), \text{ and conditionally on } \Sigma$$
$$\mu \mid \Sigma \sim \mathcal{N}_{p_1}(\mu_0, \Sigma/\kappa_0).$$

Here $\mu_0 = \texttt{priorMu}$, $\Lambda_0 = \texttt{priorDf} \times \texttt{priorSigma}$,
$\mathrm{df} = \texttt{priorDf}$, $\kappa_0 = \texttt{priorKappa}$.

The 'central tendency' of this inverse Wishart distribution is approximately
$\texttt{priorSigma}$.

priorMu

1. priorMu is your prior guess for the mean of $\theta_j^{(1)}$, priorSigma is your prior guess for their between-groups variance.

2. For priorMu you normally do not want to specify much; perhaps[4] priorMu(outdegree) a value like -1 or -2, priorMu(reciprocity) a value like $+1$ or $+2$, and its other coordinates 0.

---

[4]Tom's words - BE CAREFUL!!!

priorSigma

3. priorSigma is he prior guess for the between-groups covariance matrix of the $\theta_j^{(1)}$
   At the same time, priorKappa$^{-1}\times$ priorSigma is the uncertainty of your guess for priorMu.

4. By priorSigma you want to express the prior idea that the groups have rather similar parameters. For most parameters this corresponds to differences of the order of magnitude of about 0.3, so prior variances of about 0.1. For some parameters the likely values of similar groups may have a larger range; this could be the case e.g., for the outdegree effect; reciprocity; avSim; and egoX, altX, egoXaltX effects of covariates with a *small* variance; this would lead to larger prior variances.

5. Prior correlations ... why not choose the value 0.

priorKappa

Your ignorance about the mean of $\theta_j^{(1)}$
is expressed by $\text{priorKappa}^{-1} \times \text{priorSigma}$.

Note that this entails a double use of priorSigma:
it is the prior between-groups covariance matrix,
and also the ignorance about the prior mean times
a constant (viz., $1/\text{priorKappa}$).
This is perhaps unfortunate but a mathematical convenience.

The consequence is that you should choose priorKappa
in view of the diagonal elements of priorSigma,
so that priorSigma/priorKappa represents
a reasonable degree of uncertainty.

E.g., diag(priorSigma) between 0.05 and 0.6 and priorKappa=0.1.

priorKappa (2)

It is technically possible to set priorKappa=0.

This 'means' that you do not 'assume anything' about the mean μ of θ_j^1.

- This simplifies a lot of things.
- The value of priorMu now has consequences only for the initialisation of the estimation.

priorDf

priorDf represents your certainty about Σ.

The smallest mathematically possible value
is the dimension of $\theta^{(1)}$ plus 2.

That will be almost always a reasonable choice (it is the default).

The determination of these prior parameters is so full of uncertainties, that it is advisable to do a sensitivity analysis:
try out several plausible values and
investigate the importance of the differences in results.

Unfortunately, this takes a lot of time...

priorMeanEta and priorSigEta

Model: η ('eta') is the vector of non-varying parameters.

Mostly, there will be a lot of information about them,
and it is not necessary to specify a prior distribution;
technically, they can get a constant improper prior.

However, some effects could be group-level effects, e.g., effects of group
variables such as the proportion of males or log group size (represented by
its egoX effect). For such variables, sienaBayes has the option to specify
a normal distribution with mean given by priorMeanEta; the variance is
given by priorSigEta.

The elements of priorMeanEta and priorSigEta which are NA will
specify the constant prior.

Practical advice for prior specification

In most cases: use defaults for priorRatesFromData, priorDf, priorSigEta; for priorMu, e.g. use -2 for the density and $+2$ for the reciprocity parameter;
priorKappa = 0;
if p is the number of randomly varying effects (including rate parameters), as reported from
print(myeff, includeRandoms=TRUE, dropRates=TRUE) if there are two waves and one dependent variable:

```
Sig <- matrix(0,p,p)
diag(Sig) <- 0.1
Sig[2,2] <- 0.6 # density parameter
Sig[3,3] <- 0.6 # reciprocity parameter
..........
ans <- sienaBayes(..., priorMu=Mu, priorSigma=Sig, priorKappa=
```

sienaBayes {multiSiena}

A function for fitting Bayesian models

Description

A function to fit hierarchical Bayesian models random effects to sienaGroup data objects. Uses the function maxlike the Bayesian part is performed in R.

Usage

```
sienaBayes(data, effects, algo,  saveFreq=100,
    initgainGlobal=0.1, initgainGroupwise = 0.02, initfgain=0.2, gamma=0.05,
    initML=FALSE, priorMeanEta=NULL, priorSigEta=NULL,
    priorMu=NULL, priorSigma=NULL, priorDf=NULL, priorKappa=NULL,
    priorRatesFromData=2,
    frequentist=FALSE, incidentalBasicRates=FALSE,
    reductionFactor=0.5, delta=1e-04,
    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)
```

Data for sienaBayes

- For sienaBayes, a sienaGroup data set is needed. This means you first create the *N* separate data sets, which need to have the same variables, with the same names, and the same number of waves; then you apply sienaGroupCreate to combine them.

- It is still useful to read the descriptions given by print01Report, even though this is rather repetitive.

- If there any groups with 'forbidden changes', e.g., structural zero turning into observed 1, which would run into logical errors, the data set needs to be changed; e.g., by splitting waves, or changing some values into NA.

https://www.stats.ox.ac.uk/~snijders/siena/
changeForbiddenChanges.R

The function sienaBayes: parts

- Data input and checks
- Initialization: MoM estimation for the whole data set (all parameters equal across groups except for basic rates)
  to give initial values; and then also for each group.
- improveMH: tuning of MH steps for the SAOM parameters.
- warming phase of nwarm iterations.
- second improveMH: tuning of MH steps again.
- main phase of nmain iterations.

The function sienaBayes: initialization

The initialization consists, first, of a brief MoM estimation for the whole data set as a multi-group model by siena07.

Assumption:
all parameters equal across groups except for basic rates.

After this, a brief MoM estimation for each group,
taking the earlier estimation as the starting point.

These estimations do not need to converge (they use nsub=2), they are allowed to be quite rough.

The function sienaBayes: initialization (2)

You can circumvent the multi-group MoM estimation by doing it outside of sienaBayes, and then giving the result to sienaBayes as the prevAns parameter.

This is advisable. It can be illuminating to take a look at the results. It is not necessary to have this estimation converged well.

How much of the multi-group MoM estimation then is skipped depends on the prevOnly parameter; see the help page.

Outline of estimation process

Consider a data set with just two observation waves.

The estimation uses a sequence of ministeps that brings the data of the first wave to the data of the second wave; let us call this the *bridge*.

At every moment of the estimation process, the process has a *state* consisting of

$\mathcal{B} = $ the bridge (sequence of ministeps) and

$\Phi = $ the parameters, $\Phi = (\theta_j^{(1)}, \mu, \eta, \Sigma)$.

The purpose of the estimation is that the process converges : for Φ to a sample from the posterior distribution of the parameters,

for $\mathcal{B}$ to a sample from the post. distr. of the bridge of ministeps.

Outline of estimation process (2)

The estimation process consists of alternations between updates of bridge $\mathcal{B}$ and of parameters Φ. (Many changes in $\mathcal{B}$ for each change in Φ.) These are guided by the estimation statistics[5] $s(\mathcal{B})$.

The statistics $s(\mathcal{B})$ are strongly auto-correlated; for the changes in Φ, these auto-correlations should not be too high.

To obtain convergence, a high number of iterations of Φ is required; it is not necessary to record them all, therefore *thinning* is applied when recording the results.

---

[5]For likelihood estimation, used in sienaBayes, these are the score functions.

Outline of estimation process (3)

The numbers of iteration steps in the MCMC process are as follows

1. nwarm + nmain iterations recorded at the highest level.
2. Each of these is composed of nrunMHBatches steps, of which only the last one is recorded ('thinning') in (1).
3. Each of the steps in (2) contains nrunMH Metropolis-Hastings steps to simulate the next $\mathcal{B}$, using the method of Snijders, Koskinen & Schweinberger (2010) used also for ML estimation by siena07.

So the total number of lowest-level iterations, per group, is
nrunMH×nrunMHBatches×(nwarm + nmain). Values of nrunMH depend on data and multiplication factor mult; nwarm, nmain, and nrunMHBatches are given in the call of sienaBayes. For an estimation object ans constructed by sienaBayes or by siena07-ML, the value of nrunMH can be obtained as ans$nrunMH.

The multiplication factor: intro

The multiplication factor is called `mult`. It is set in the algorithm in `sienaAlgorithmCreate`.

Computation time is roughly proportional to `mult`; autocorrelations are lower when `mult` is higher.

So the point is to set `mult` high enough, but not too high. The default value `mult=5` is reasonable, and it is not really necessary to change it; however, efficiency may be improved by giving it a better value.

The multiplication factor can be given as one number, or specific for each group×period combination. Then it should be a vector with length `nGroups` × `nPeriods` (the number of basic rate parameters for one dependent variable).

The multiplication factor: intro (2)

Adequate values of the multiplication factor `mult` do not depend strongly on the model specification; the method to update the 'bridge' $\mathcal{B}$ is used also in `siena07-ML`; therefore, good values of `mult` can be determined from results for ML estimation by `siena07`, using an empty model.

The procedure on the following pages can be used for determining a good value for the multiplication factor. This is just one approach; when you understand it, you can use whatever way to get good values.

Setting the multiplication factor (1)

1. Define the sienaGroup data set that will be used for sienaBayes and use getEffects to specify the empty model (with or without the reciprocity effect).

2. Specify algorithm settings by sienaAlgorithmCreate for a short estimation by maximum likelihood; e.g., with mult=5, nsub=2, n3=500, maxlike=TRUE, and a value for seed to make things replicable.

3. Estimate this multi-group model using siena07; the result will be denoted by mlans. It does not matter that it has not yet converged.

Setting the multiplication factor (2)

4. Inspect the autocorrelations mlans$ac.
   Especially important are the autocorrelations for the rate parameters.
   The rate parameters in the effects object are given by
   mlans$effects$basicRate.
   You could look at
   hist(mlans$ac[mlans$effects$basicRate])
   If the maximum is less than 0.4, mult is OK, and you are done (or
   you might even try to reduce mult, if the maximum is much lower).

Setting the multiplication factor (3)

5. If the maximum is greater than 0.4, make `mult` into a vector of length
   nGroups× nPeriods and increase the coordinate of `mult`
   for those group/wave combinations for which ac is greater than 0.4.
   Given that you started with mult=5, an example for doing this,
   in the case of one dependent variable, is

   `mult.r <- round(20*mlans$ac[mlans$effects$basicRate], 1)`

   This will set the multiplication factor to 5 for group/wave
   combinations
   for which ac was exactly 0.4,
   and the others to a proportionally lower or higher value,
   with rounding to get nice values.
   Inspect the values:
   `hist(mult.r)`
   just to have an idea of their sizes.

Setting the multiplication factor (4)

6. Create an algorithm object by sienaAlgorithmCreate,
   still using nsub=2, n3=500, maxlike=TRUE,
   but now with mult=mult.r,
   where mult.r is the new vector multiplication factor.

7. Now estimate the model again, using this algorithm object,
   and with prevAns=mlans. Again, convergence is not necessary.
   Give the result a new name, e.g., mlans2.

8. Again inspect the autocorrelations for mlans2,
   focusing on those for the rate parameters.
   The autocorrelations are random variables (like anything),
   so they will not necessarily have become smaller ...
   If all are less than 0.4, you are done, and
   you can use this mult=mult.r for the estimations using sienaBayes.

Setting the multiplication factor (5)

9. If some of the autocorrelations are higher than 0.4,
   make further modifications to the multiplication factor
   in accordance with the approach above.

10. The multiplication factor found in this way can be used also
    for estimating more complicated models
    for the same data set using sienaBayes.

11. Since autocorrelations are random variables,
    and the threshold of 0.4 is no more than a rule of thumb,
    do not mind small variations.

Prolonging `sienaBayes`

If you decide that your `sienaBayes` run was too short, you can prolong it giving it as the `prevBayes` parameter to a new run of `sienaBayes`.

This should have the same prior distribution. The initialisation then is skipped, and the new runs start at the end of the earlier runs.

If `newProposalFromPrev=TRUE`, the proposal distribution will be determined from the results of the `prevBayes` object; if `newProposalFromPrev=FALSE`, the proposal distribution will be identical to the one of the `prevBayes` object.

Advice: use `newProposalFromPrev=TRUE` if the trace plots of the `prevBayes` object are not very stable.

The main parameters of the function (1)

- data: a sienaGroup object,
  created by sienaGroupCreate; at least two groups!
  the same data structure as for a multi-group analysis.

- effects: a regular sienaEffects object;
  but now its random column also is important;
  this is set using the parameter random in functions setEffect and
  includeInteraction.
  printed by
  print(myeff, includeRandoms=TRUE)
  perhaps use also (..., dropRates=TRUE).
  This also gives the dimensions for priorMu, priorSigma.
  Rate effects depending on actor covariates or degrees are not
  implemented.

The main parameters of the function (2)

- algo: a sienaAlgorithm object created by
  sienaAlgorithmCreate; currently, mainly the mult
  and seed parameters are relevant;
  note that mult can be a number or a vector.

- saveFreq: frequency for saving intermediate results. Important to
  protect against losing everything
  in case of a failure of some kind.
  Provisional results are save as object z in a file
  PartialBayesResult.RData (overwriting!).
  This can be used as prevBayes, see below.

The main parameters of the function (3)

- `initGainGlobal`, step size in the initialization MoM estimation for the total data set.
  Can be chosen smaller to improve stability.

- `initGainGroupwise`, step size in the initialization MoM estimation for the separate groups. Can be chosen smaller, or even 0, to obtain stability for small groups.

The main parameters of the function (4): priors

- priorMu, priorSigma, priorDf, priorKappa, priorMeanEta, priorSigEta: see above.
- priorRatesFromData: if 1 or 2 (default), priors for the basic rate parameters are calculated from the data; admissible for these 'nuisance parameters'.
  Then priorMu and priorSigma still need to include values for the rates for the priors (the basic rates are always counted among the randomly varying parameters), but these values are not taken into account.
- reductionFactor: multiplier for the observed covariance matrix of rate parameters to get the prior covariance matrix for the basic rate parameters.
  Can be between 0 and 1 in case of too strong variation between these estimated rates.

The main parameters of the function (5)[6]

- `incidentalBasicRates`: the value TRUE can be tried out if convergence is difficult mainly because of the basic rate parameters. The basic rate parameters then are taken out of the Bayesian estimation and estimated by Maximum Likelihood. Experimental. (Target density of the MCMC process then is profile likelihood, maximized over rate parameters.)

- `nImproveMH`: Number of iterations per improveMH step. Can be chosen smaller if faster computations are required. Small values will lead to less precise tuning.

- `targetMHProb`: the desired proportion of acceptances in MCMC steps of updating $\theta_j^{(1)}, \mu, \eta, \Sigma$.
  Theoretical optimal value 0.25, tried to be reached by improveMH. Experimental.

---

[6]Not of great importance

Main parameters (6): numbers of iterations

- nwarm: number of warming iterations.
  Advice: start with 100; if you get experience with a data set and see that convergence sets in later or earlier, change accordingly.

- nmain: number of main iterations.
  Advice: start with 200 for explorations (computing times, etc.); try 1000 for real work; always later stretches can be added using prevBayes (see below).

- nrunMHBatches: proportion of steps for thinning.
  Each of the nwarm and the nmain iterations is composed of nrunMHBatches steps. What counts are effectively the products (nrunMHBatches $\times$ nwarm) and (nrunMHBatches $\times$ nmain).
  Perhaps nrunMHBatches can be chosen 10 (rather than the current default 20).

Main parameters (7): initialization object

- prevAns: A sienaFit object, estimated by siena07 (usually MoM, ML also possible) for *the same effects object* (but random column does not matter). It does not need to have converged well! This then is used for the initialization, of which the first part will be skipped.
  It is advisable to carry out such a MoM estimation as a first exploration anyway.
  When you estimated earlier with sienaBayes for this effects object, perhaps different choice of randomly varying parameters, and obtained object ans.b, then you can use
  prevAns=ans.b$initialResults.

- usePrevOnly: if FALSE, then even with a prevAns object the initialization still will be entirely performed, but starting with the prevAns object.

Main parameters (8): continuing earlier estimation

- prevBayes: A sienaBayes object, estimated by sienaBayes for the same data, effects, algo objects.
  Initialization then is skipped, and the MCMC process continues from the last point reached by the prevBayes object. nwarm does not matter; nmain is the number of new (additional) iterations; use the same values for the prior parameters unless you want to experiment with changing them.

- proposalFromPrev: whether improveMH again is performed, now using the covariance matrix of the simulated parameters, when using a prevBayes object.

Main parameters (9): parallel processing

- nbrNodes: number of parallel processes.
  See help page for siena07 for determining the number of processes you can use on your machine.
  For sienaBayes as well as siena07-ML, parallellization is by period (groups multiply the periods). So for 4 groups with 1 wave you cannot use more than 4 processes.

- clusterType: type of cluster for your machine.

Combining several `sienaBayes` objects

When you have several `sienaBayes` objects for the same data and model specification (including choice of random effects), e.g. through use of `prevBayes`, they can be combined by

- `glueBayes`: produces one new `sienaBayes` object by concatenating two `sienaBayes` objects;

- `extractSienaBayes`: extracts simulated parameters from a list of `sienaBayes` objects;
  this can be used with a list of one such object, then think of using
  `drop(extractSienaBayes(....`
  to get rid of the superfluous dimension.

print.sienaBayesFit {multiSiena}   R Documentation

Methods and functions for processing sienaBayes objects

Description

`print` and `summary` methods for <u>sienaBayesFit</u> 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 <u>sienaBayesFit</u>.

extract.sienaBayes {multiSiena}   R Documentation

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;

Plotting the posterior means MDS[7]

Posterior means can be plotted by the function `plotPostMeansMDS`.

Tom: mainly as a diagnostic
for whether there are too many random effects.
If there are too many random effects
(in view of what is reasonable given the number of groups)
the 'estimation' of the random effects
will be trapped randomly by a few groups,
and this will show in a pattern with a dense core and a few outliers,
where the outlying groups will be different in different runs
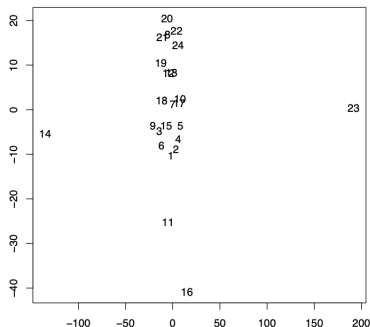of `sienaBayes` (with different random number seeds).
In this case, there probably also will be poor convergence.

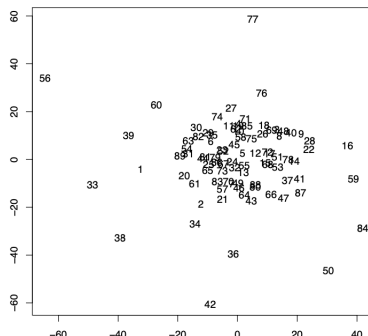[7]Experimental (Tom's experiments

Plotting the posterior means MDS (2)

MDS plots of posterior means

Too many random effects?   Good number of random effects?

Plotting the posterior means MDS (3)

For a reasonable number of random effects,
the MDS plot of the posterior means can be used
as a diagnostic for outlying groups.
(*this is speculative*)

Literature

Johan Koskinen and Tom 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

The Siena scripts page contains some examples of the use of sienaBayes; see http://www.stats.ox.ac.uk/~snijders/siena/

An example of the meta-analysis approach (not the integrated random parameter approach) using Chris Baerveldt's data is in the script

RscriptMultipleGroups_meta.R

on the Siena website.

RSiena manual: Chapter 11.

Further references can be found in the RSiena manual.