Multilevel analysis of network dynamics using `sienaBayes`

Tom A.B. Snijders

University of Groningen
University of Oxford

June, 2023

Traditionally, network analysis tended to consist of cases studies of single networks.

However, it is preferable to generalize to a **population** of networks.

This can be achieved, in principle, by multilevel network analysis in the sense of analyzing multiple similar networks, mutually independent.

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

Also see Entwisle, Faust, Rindfuss, & Kaneda (*AJS*, 2007) who gave on overview of empirical work until then involving multiple networks.

Sample from Population of Networks

Suppose we have a sample indexed by $j = 1, \ldots, N$
from a population of networks on disjoint node sets,
where the networks are 'replications' of each other
in the following sense:

they all are regarded as realizations of
processes obeying the same model,
but having different parameters $\theta_1, \ldots, \theta_j, \ldots, \theta_N$.

Each disjoint network is called 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).

Several approaches are possible for combining such data:

1. Multi-group analysis:
   assume all parameters are identical.
2. Integrated hierarchical approach:
   Assumption: population of networks, normal distribution,
   A. mixed effects,
   some parameters varying, others constant across groups;
   B. random effects, all parameters varying.
3. Meta analysis:
   Assumption: population of networks,
   no distributional assumptions:
   two-stage meta analysis.
4. Meta analysis: no population assumption:
   Fisher combination of independent tests.

Such data sets have multilevel structure;
all caveats and considerations from usual multilevel analysis apply!

See Snijders & Bosker (textbook 2012);
T.A.B. Snijders, 'The Multiple Flavours of Multilevel Issues for Networks',
in Lazega & Snijders (2016).

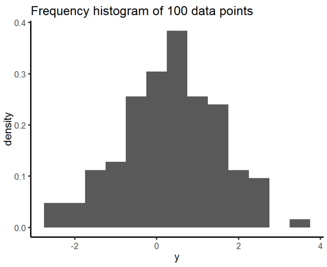
Multilevel designs, generalization, and the risks of having few groups

For some basic intuitive understanding of the consequences of multilevel designs, consider the simplest case:
estimation of the population mean μ of some variable.

Left one-sided testing.

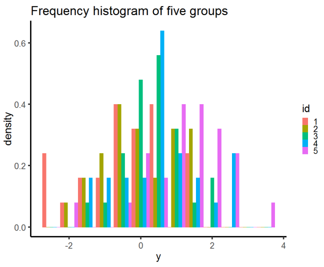
We have $N = 5$ randomly drawn groups each of size $n = 20$.
Data range from –2.5 to +3.5, as in the histogram (next page).
Is there evidence that the population mean is positive?

Test $H_0 : \mu = 0$ against $H_1 : \mu > 0$.



mean = 0.36, s.d. = 1.18, t_{99}=3.40, p = 0.0003

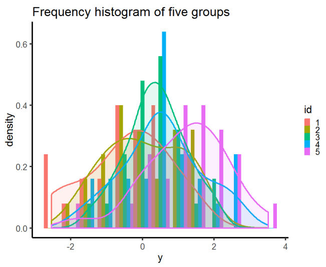
linear model (OLS): yes



multilevel t_4 = 1.49, p = 0.12

multilevel model: no

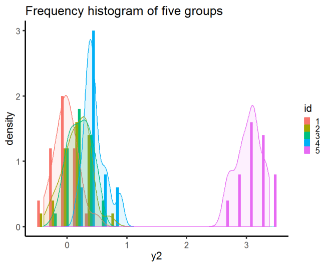
but by group p = 0.11, 0.84, 0.034, 0.021, 0.00002



multilevel $t_4 = 1.49$, $p = 0.12$

multilevel model: no

but by group $p = 0.11, 0.84, 0.034, 0.021, 0.00002$



An even more extreme example:
OLS $t_{99} = 7.9$, $p \approx 0$; multilevel $t_4 = 1.50$, $p = 0.12$

by group $p = 0.64, 0.002, 0.000012, < 0.0001, < 0.0001$

The conclusion for both data sets is not only, that
the multilevel model tells us μ is not significantly positive.

There is also significant heterogeneity between the groups:
μ is not zero in all groups
(this can be concluded already from an OLS analysis).

And one of the groups has such a small *p*,
that the significance even survives multiple testing with Bonferroni.

If we wish to generalize to the population of all groups,
there is no evidence of a positive mean;
but there is evidence that
at least one of these groups has a positive mean,
and that the mean of these five groups is positive.

Message

The message of the preceding is:

When you have a small number of groups,
especially if they are heterogeneous,
assuming a random effects model is risky.

It is important to look at results for individual groups.

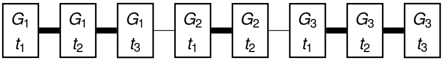
If groups are large enough to be analysed individually, do so.

After this sensitization, on with the four approaches:

1. Multi-group analysis:
   assume all parameters are identical.
2. Integrated hierarchical approach:
   Assumption: population of networks, normal distribution,
   A. mixed effects,
   some parameters varying, others constant across groups;
   B. random effects, all parameters varying.
3. Meta analysis:
   Assumption: population of networks,
   no distributional assumptions:
   two-stage meta analysis.
4. Meta analysis: no population assumption:
   Fisher combination of independent tests.

Multi-group analysis

Multi-group analysis (`sienaGroupCreate`, `siena07`) assumes all parameters are the same. The groups are treated just like subsequent waves, strung in a sequence, where
(of course) transitions between groups are not analyzed.



Example for 3 groups, with 3 / 2 / 3 waves;
the analyzed 2+1+2 periods are indicated as bold lines.

Groups can have different sizes, must have same variables and a common model specification.

The multi-group analysis assumes
all parameters are identical.

possible but risky
Gives a global impression,
risk of underestimated standard-errors.

Try to find relatively homogeneous sets of groups for doing this.

`sienaTimeTest()` can be used to test homogeneity assumption.

Random effects: hierarchical multilevel structure

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

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 <u>distribution</u> of θ_j and consider the
'posterior' 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 rather small networks.

(For large networks, where the model of interest can be estimated
without any problems, a two-stage procedure may be better:
(1) estimation by group, (2) meta-analysis.)

Bayesian approach

The approach taken here is that of *Bayesian statistics*,
as distinct from the more usual *frequentist statistics*.

Bayesian statistics postulates a probability distribution
not only for the data, but also for the parameters.

The summary is that the data Y have a conditional distribution
given the parameters θ, which is the statistical model;
and the parameters θ have a **prior distribution**,
reflecting what the researcher knows about them in advance,
i.e., before having looked at the data;
the statistical analysis then leads to the **posterior distribution**,
which is the conditional distribution of the parameters given the data.

Bayesian approach (2)

In symbols:

Before the data we have the model $p(x \mid \theta)$
and the prior $\pi(\theta)$,
then given the data we have the posterior $p(\theta \mid x)$
which is defined, according to Bayes' rule, as

$$p(\theta \mid x) \,=\, \frac{\text{joint p.d. of } (x,\,\theta)}{\text{p.d. of } x} \,=\, \frac{p(x \mid \theta) \,\times\, \pi(\theta)}{\int \big(p(x \mid \theta') \,\times\, \pi(\theta')\big)\, d\theta'} \;.$$

Bayesian approach (3)

The parameter can then be estimated by its posterior mean

$$\mathrm{E}\{\theta \mid x\}$$

and its uncertainty is expressed
by the posterior covariance matrix

$$\mathrm{cov}\{\theta \mid x\} \ ;$$

the posterior standard deviations are the square roots
of its diagonal elements, and have the role of standard errors.

This is all very nice;
the catch is that the researcher has to define a prior.

Hierarchical model: restricted variation

Terminology here:
fixed parameters have the same value across groups;
random parameters have different values 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 to be less of 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.

Fixed effect k:

Testing $\eta_k = 0$ means testing the null hypothesis
that the effect $\theta_{jk} = 0$ in every group *j*,
under the auxiliary assumption that
θ_{jk} is the same for all *j*, i.e., $\theta_{jk} = \eta_k$.

random effect k:

Testing $\mu_k = 0$ means testing the null hypothesis
that the population average of the effects θ_{jk} is 0.

The auxiliary assumption leads to a smaller
posterior standard deviation for η_k than for μ_k,
unless the prior for μ_k has a very small variance.

Group-level variables

It is possible to use group-level variables.
These should be defined as constant actor covariates (`coCovar`), with the same value for all actors.

It is helpful for convergence to center the group-level variables.
I'm not sure that the centering in the definition of `coCovar` survives the operation of `sienaGroupCreate` in a good way.

Therefore the recommendation is to center such variables "by hand", by subtracting their mean (or approximate mean) before executing `coCovar`.

Varying group sizes

Parameters of the ERGM and SAOM are not well comparable across group sizes (number of actors in the networks), because if the number of actors is larger while further the network evolution is similar, there are more ties that should <u>not</u> be created.

For ERGMs this was shown in research by Krivitsky, Kolaczyk, and Butts.

The main culprit for the SAOM is the `density` parameter.

To take account of the different group sizes, it is advisable to use the group-level covariate with the value $\log(n)$, where $\log(\cdot)$ is the natural logarithm and n group size.

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 posteriors.

Three kinds of plot are important:

1. *multidimensional scaling 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`, `priorMeanEta`, `priorSigEta`.

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

The six prior parameters are explained in the following pages.

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 μ_0 = `priorMu`, $\Lambda_0 =$ `priorDf` $\times$ `priorSigma`,
df = `priorDf`, κ_0= `priorKappa`.

The 'central tendency' of this inverse Wishart distribution is approximately `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 `priorMu(outdegree)` a value like –1 or –2,
   `priorMu(reciprocity)` a value like +1 or +2,
   and its other coordinates 0.

priorSigma

3. `priorSigma` is the 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 `priorMu`.
   (That both are proportional is a mathematical issue.)
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.
   (And covariates with a large variance would get smaller prior variances.)
5. Prior correlations ... why not choose the value 0.

priorKappa

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

priorDf

priorDf represents your certainty about Σ.

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

That is the default, and will be mostly be a reasonable choice.
But you could use a higher value if you wish.

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;

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=0.1, ... )
```

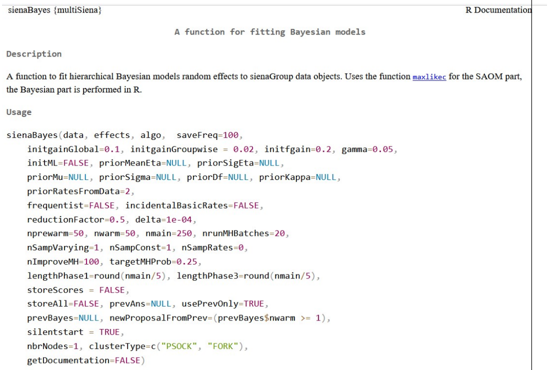
Prior distributions

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...

Perhaps it is more important to try out several specifications of what are the randomly varying parameters.

The function sienaBayes



sienaBayes {multiSiena} R Documentation

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 = 0.02, 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),
    silentstart = TRUE,
    nbrNodes=1, clusterType=c("PSOCK", "FORK"),
    getDocumentation=FALSE)
```

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`.

  `http://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.

Multilevel: iterations in iterations

- `nrunMH` steps to simulate the chains connecting consecutive panel observations for given $\theta_j^{(1)}, \mu, \eta, \Sigma$, according to Snijders, Koskinen & Schweinberger (2010).
  Values of `nrunMH` depend on data and multiplication factor `mult`, and are reported by results of `siena07`-ML; from `sienaBayes` they can be obtained as `ans$nrunMH`.
- `nrunMHBatches` MCMCM steps ('thinning') of updating $\theta_j^{(1)}, \mu, \eta, \Sigma$.
- `nwarm` + `nmain` iterations of these steps.

So the total number of lowest-level iterations, per group, is
`nrunMH` $\times$ `nrunMHBatches` $\times$ (`nwarm` + `nmain`).

Multiplication factor

Subsequent simulations (in the `nrunMH` steps) are correlated.
The autocorrelations of the estimation statistics[1]
should preferably be less than 0.4.
In Section 6.11 of the manual, this is mentioned as a requirement for ML estimation. For `sienaBayes` this is not really a requirement; but it might nevertheless be helpful to satisfy it also here.
This autocorrelation is influenced by the *multiplication factor*.

See the manual for the description of what the multiplication factor does.

To get a lower autocorrelation for a given group/period,
the multiplication factor can be increased,
which then increases the number of MCMC steps (given as `ans$nrunMH`).

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

Setting the multiplication factor

The multiplication factor can be given as one number,
or a vector with length `nGroups` × `nPeriods`
(the number of basic rate parameters for one dependent variable),
referring to the group × period combinations.

Computation time is roughly proportional to the multiplication factor,
so it should not be set unnecessarily high.
It may be best to set it for each group/wave combination specifically.

Adequate values of the multiplication factor
do not depend strongly on the model specification,
and therefore can be determined from results
for ML estimation using an empty model.

Setting the multiplication factor (continued 1)

The following procedure can be used for determining a good value
for the multiplication factor.
This is just one approach; after understanding how it works,
you can use whatever way to get good values.

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 (continued 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$basicRate`.
   E.g., if there is only one dependent variable, you may use
   `hist(mlans$ac[mlans$effects$basicRate])`
   If the maximum is less than 0.4, the multiplication factor is OK,
   and you are done (or you might even try to reduce it, if it is much lower).
5. If the maximum is greater than 0.4, increase the multiplication factor 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 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.

`hist(mult.r)`
just to have an idea of their sizes.

Setting the multiplication factor (continued 3)

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. If all are less than 0.4, you are done, and can use this `mult=mult.r` for the estimations using `sienaBayes`.
   If some are higher than 0.4, make further modifications to the multiplication factor in accordance with the approach above.
9. The multiplication factor found in this way can be used also for more complicated models in `sienaBayes`.

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 initialization 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; see below.
- `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. Usually no need to use it.

Main parameters (5): less important

- `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.

Main parameters (6): numbers of iterations

- `nwarm`: number of warming iterations.
  Advice: start with 100 or 200; 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` × `nwarm`) and (`nrunMHBatches` × `nmain`).
  Sometimes higher, sometimes lower values can be used.

Main parameters (7): initialization object

- `prevAns`: A `sienaFit` object, estimated by `siena07` (usually MoM, `conditional=FALSE`, 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 performed entirely, but starting with the `prevAns` object. See help page for details.

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

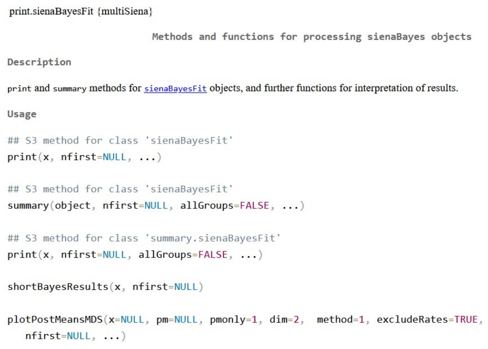
- `nbrNodes`: number of parallel processes.
  See help page for `siena07` for determining the number of processes you can use on your machine.
  For `sienaBayes` as well as `siena07`-ML, parallellization is by period (groups multiply the periods). So for 4 groups with 2 waves you cannot use more than 4 processes.
- `clusterType`: type of cluster for your machine.

The parallel processing implemented in sienaBayes
does not work on all hardware.
If you want to look behind the screens:
all parallellization happens within the function `sienaBayes`.

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}

Methods and functions for processing sienaBayes objects

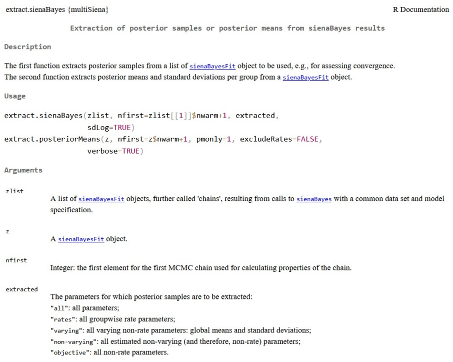
Description

`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, ...)

shortBayesResults(x, nfirst=NULL)

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

The function sienaBayes



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.

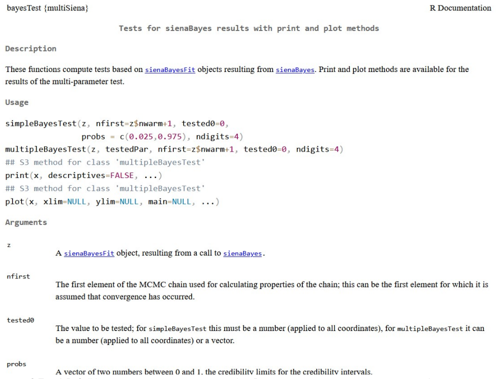
Usage

```
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; `"non-varying"`: all estimated non-varying (and therefore, non-rate) parameters; `"objective"`: all non-rate parameters.

The function sienaBayes



bayesTest {multiSiena} R Documentation

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.

Maximum degree differences, and estimated rate parameters and autocorrelations for rate parameters from the ML estimation can be used as diagnostics.

They may point to groups that can be considered outliers, which then should be scrutinized, perhaps dropped, or modified (replacing unlikely values by `NA`).

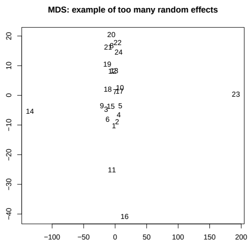
Plotting the posterior means

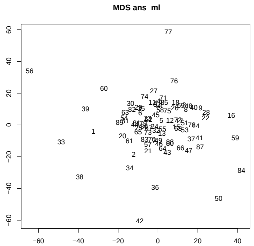
Posterior means can be plotted by the function `plotPostMeansMDS`.

I use this 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.





Too many random effects Good number of random effects

Examples of MDS plots of posterior means

For a reasonable number of random effects,
the MDS plot of the posterior means can be used
as a diagnostic for outlying groups.

Literature

Johan H. Koskinen and Tom A. B. Snijders (2023), Multilevel Longitudinal Analysis of Social Networks. *Journal of the Royal Statistical Society, Series A*, in press.
DOI: `https://doi.org/10.1093/jrsssa/qnac009`

An example of the use of sienaBayes is in the script
`http://www.stats.ox.ac.uk/~snijders/siena/RscriptsienaBayes_4.r`

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 scripts page of the Siena website.

RSiena manual: Chapter 11.