Multilevel analysis of network dynamics using **sienaBayes**

Tom A.B. Snijders

University of Groningen
University of Oxford

June, 2024

Model Specification

Prior distributions

The function **sienaBayes**

Data exploration

Literature

Multilevel analysis of network dynamics using `sienaBayes`

This is a second set of slides, continuing on

'Introduction to Multilevel Analysis of Network Dynamics using `sienaBayes`'.

Model Specification

Group-level characteristics

Varying group sizes

Randomly varying effects

Group-level characteristics

Group-level characteristics can be used in the multi-group analysis, both as main effects and in interaction with other effects: '*cross-level interactions*'.

Group-level covariates can be included in the data set for each group as actor covariates.
It is helpful for convergence to center the group-level covariates.
This can best be done 'by hand':
subtract a value that is equal, or close to, the average value across all groups and include this covariate in each group data set.
E.g., using a variable n defined as group size, one might use

```
nn  <- coCovar(rep((log(n)-3.3),n), center=FALSE, warn=FALSE)
```

if 3.3. is close to the mean of $\log(n)$.

Group-level characteristics (2)

In the model specification,
group-level covariates will have an `egoX` effect,
and/or be included in interactions with their `egoX` effect.

It is also possible in **`sienaBayes`**
to use group-level averages of endogenous variables
as 'real-time' effects following the endogenous dynamics:
`avDeg, avDegIntn, avGroupEgoX, avGroup`.

See their definitions in the RSiena manual.

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 not 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; or check with `sienaGOF` that $\log(n)$ does not matter.

Varying group sizes (2)

The theoretically expected effect of log group size (natural log) is –0.5 for the empty model; or –1 for the `creation` effect in the empty model (the problem occurs only for creation of new ties).

Experience is that in many non-empty models for data sets with a moderate variation in group sizes, log group size does not have important effects and can be omitted.

If there is an effect, it is expected to be negative, and then there also my be interaction effects with reciprocity and the effect representing transitivity; it is unknown whether this extends also to other effects.

Randomly varying effects

Some effects are randomly varying across groups,
having a multivariate normal distribution $\mathcal{N}(\mu, \Sigma)$;
the others are constant across groups, with parameter value η.

We shall use the terms
random parameters for those having different values across groups;
fixed parameters for those having the same value across groups.

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.

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.

How many random parameters could be included in the model
depends also on the number of groups.
E.g., for 20 groups, 5 random parameter is a lot;
for 100 groups, 12 random parameters is a lot.

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 assumption that effect k is fixed across groups 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.

Prior distributions

`priorMu`

`priorSigma` and `priorKappa`

`priorDf`

`priorMeanEta` and `priorSigEta`

Prior distributions

For a smallish number of groups, the prior is consequential.

It is given to **sienaBayes** as:
for the random effects:
`priorMu`, `priorSigma`, `priorDf`, `priorKappa`,
for the fixed effects:
`priorMeanEta`, `priorSigEta`.

For rate parameters, **sienaBayes** uses a data-dependent prior (depending on the option chosen; the default assumption is OK).

The six prior parameters are explained in the following pages.

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, `priorMu(sameX(V))` for important variables V with homophily effects a value like 0.2 or 0.4, and its other coordinates 0.

priorSigma and priorKappa

1. priorSigma is the prior guess for
   the between-groups covariance matrix Σ of the $\theta_j^{(1)}$.
   At the same time, $\text{priorKappa}^{-1} \times$ priorSigma
   is the uncertainty of your guess priorMu.
2. (That both are proportional is a mathematical issue.)
   This implies that priorKappa should be quite small,
   because the uncertainty about priorMu is much larger
   than the variability between the groupwise parameters.
   priorKappa can be put at 0.01 or even 0
   (note that variances are on a quadratic scale, so
   the value 0.01 means the variability between the groupwise parameters
   might be 10 times smaller than your uncertainty about the global mean).

priorSigma (2)

1. 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 (density)
   effect; reciprocity; linear and avSim for behavior; 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.)
2. Prior correlations ... why not choose the value 0.

`priorSigma` (3)

This leads to, e.g., if 2 and 6 are the positions of the `density` and `linear` effects:

```
Sig <- matrix(0,p,p)
diag(Sig) <- 0.01
Sig[2,2] <- 0.1
Sig[6,6] <- 0.1
.....
ans.multi <- sienaBayes(...,
                        priorSigma=Sig, ...)
```


`priorKappa` (3)

It is possible to set `priorKappa` equal to 0.

This means that you do not assume anything about the mean μ of $\theta_j^{(1)}$.

This simplifies a lot of things.

The value of `priorMu` now has little consequences;
only for the initialisation of the estimation.

This may be preferable, unless the data gives so little information
(e.g., few or very small groups)
that convergence will be enhanced by prior knowledge about μ.

`priorDf`

`priorDf` represents your certainty about Σ.

It is expressed as the hypothetical sample size
on which your knowledge about Σ is based.
The prior assumptions will have a greater weight
accordingly as `priorDf` is higher.

This means that normally, you want to choose it as small as possible.
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.

`priorMeanEta` and `priorSigEta`

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 may be group-level effects.

For such effects, **sienaBayes** has the option
to specify a prior normal distribution with
mean given by `priorMeanEta`, and variance by `priorSigEta`.

The elements of `priorMeanEta` and `priorSigEta` which are `NA`
will specify the constant prior.
The default is that all are `NA`, and this will often be OK.

The function sienaBayes

Bayesian MCMC procedure

The estimation is done by a MCMC procedure having three levels.
It works by iteratively updating provisional estimates
for $(\theta_1, \ldots, \theta_G)$ and μ, η, Σ.

1. At the lowest level, the sequence of ministeps to connect the data for the consecutive waves is simulated;
2. at the intermediate level, groupwise parameters θ_j are updated to correspond to this sequence of ministeps;
3. at the highest level, global parameters μ, η, Σ are updated to correspond to $(\theta_1, \ldots, \theta_G)$.

Simulations at the lowest level

At the lowest level, the sequence of ministeps connecting the data for the consecutive waves is simulated; let us call this the *bridge*.

At every moment of the MCMC estimation process, the process has a *state* consisting of
$\mathcal{B}$ = the bridge (sequence of ministeps) and
Φ = the parameters $\Phi = \left(\theta_j^{(1)}, \mu, \eta, \Sigma\right)$.

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.

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[1] $s(\mathcal{B})$.

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.

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

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, and you can use `nsub=2` or `3` to limit computation time.

How much of the initialization then is skipped depends on the `prevOnly` parameter; see the help page.

The total 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 iterations per group (levels 1 and 2) 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`.

`nwarm`, `nmain`, and `nrunMHBatches`; 'thinning'

The total number of iterations per group is
$\texttt{nrunMH} \times \texttt{nrunMHBatches} \times (\texttt{nwarm} + \texttt{nmain})$.

This means that (e.g.,) multiplying `nwarm` and `nmain` by 2 and simultaneously dividing `nrunMHBatches` by 2 gives the same end result; however, the number of draws of (μ, Σ, η) recorded is half as much.

This is the idea of *thinning*. You can just select a value for `nrunMHBatches` (e.g., the default of 20) and go with it; perhaps change later on.

The value of `nwarm` can be set so that the MCMC will be stable after this number of simulations,
and `nmain` so that it suffices to get information about the posterior distribution.

Of course this will not be clear from the start.
Usual values are `nwarm=500` or `100` and `nmain=1000` or `2000`.

Making trace plots of the results

After having executed the estimation,
you can look at the results first by making *trace plots*,
representing successive draws from the posterior distribution.

`https://www.stats.ox.ac.uk/~snijders/siena/BayesPlots.r`
contains a script with functions for plotting results of **sienaBayes**.

For an object `ans.multi` created by **sienaBayes**,
you can make trace plots by commands such as

```
GlobalRateParameterPlots(ans.multi)
GlobalNonRateParameterPlots(ans.multi, setOfEffects = ...)
```

where ... is a set of effect numbers
(where 1=outdegree (density) effect).

If the plots suggest that the process stabilized after more than `nwarm` runs, e.g., after 800 runs, you can give a value for a better value `nfirst` from which to start calculating results in functions such as, e.g.,
`print(multi.ans, nfirst=800)`,
`sienaBayes.table(multi.ans, nfirst=800)`,
`plotPostMeansMDS(multi.ans, nfirst=800)`,
`print(extract.sienaBayes, nfirst=800)`, or
`print(extract.posteriorMeans, nfirst=800)`.

Prolonging **sienaBayes**

If the main phase was not long enough (and chances are that it was not), you can prolong the estimation.

The same prior distribution and effects object should be used.
The initialization and warming phase then are skipped, and the new runs start at the end of the earlier runs.

If the first object created was `ans.multi`, this can be done by

```
ans.multi1 <- sienaBayes(..., nmain=1000,
                      prevBayes=ans.multi, ...)
```

Whether an `improveMH` step is applied, depends on the `newProposalFromPrev` parameter; see the help page for **sienaBayes**.

Results then can be combined by using

```
ans.combi <- glueBayes(ans.multi, ans.multi1)
```

Further parameters of function `sienaBayes`

We have treated some of the parameters of function `sienaBayes`.

There are many more parameters; many of these are experimental.

The following pages treat the most important further parameters.

Further parameters of function `sienaBayes`(1)

- `algo`: a `sienaAlgorithm` object created by `sienaAlgorithmCreate`; currently, mainly the multiplication factor `mult` and the `seed` parameters are relevant; `mult` is extensively treated 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.

Further parameters of function **sienaBayes** (2): 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**.

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

The multiplication factor

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 $\times$ period combination.
Then it should be a vector with length `nGroups` $\times$ `nPeriods`
(the number of basic rate parameters for one dependent variable).

The function **sienaBayes** Multiplication factor

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` $\times$ `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.

Diagnostic exploration of the data

It is important when starting
to have a good descriptive knowledge of the data;
this is cumbersome because there are many groups...

Some diagnostics/descriptives are the following.

Change in outdegrees

The maximum change (between waves) in outdegrees is an important diagnostic; outliers in this respect may cause problems.

Code that may be used (for 2 waves):

```
maxdif <- function(x,k){
 net <- x$depvars[[k]]
 net[net==10] <- 0
 max(abs(rowSums(net[,,1], na.rm=TRUE) -
          rowSums(net[,,2], na.rm=TRUE)))
}
maxGroupsDif <- sapply(groups_data, maxdif, k=1)
```

This can then be plotted against other relevant group characteristics.
Adapt the code for 3 or more waves.

Maximum sums of absolute degree differences, and estimated rate parameters
for the MoM estimation used as `prevAns`,
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`).

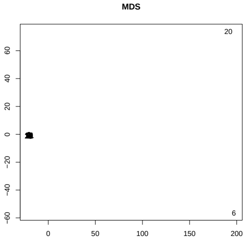
Plotting the posterior means

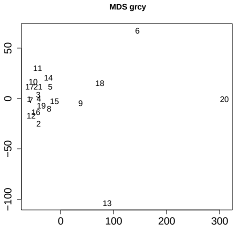
Posterior means can be plotted by the function `plotPostMeansMDS`.

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.

Posterior means can also be used for
further interpretation of the groupwise results.

A helpful function here is `extract.posteriorMeans`.

Literature

Johan H. 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 several
examples of the use of **`sienaBayes`**;
see `http://www.stats.ox.ac.uk/~snijders/siena/`

RSiena manual: Chapter 11.