CODE MANUAL

Simulation and estimation of Exponential Random Partition Models

ARTICLE HISTORY
Compiled December 11, 2021

1. File structure

All relevant R files are available in the repository https://github.com/marion-hoffman/ERPM. An example script displaying all possible uses of the code is found in the Rmarkdown file Example_script.Rmd. The functions necessary to estimation, simulation, and diverse calculations are found in the following files:

- **functions_burninthining.R**: contains functions to help determining the right value of burnin and thining for a given dataset and model,
- **functions_change_statistics.R**: contains the function *computeStatistics* that calculates for a given sufficient statistic the value of this statistic for a given partition,
- **functions_estimate.R**: contains the parent function for the estimation *estimate_ERPM*,
- **functions_exactcalculations.R**: contains functions to calculate average sizes and likelihood functions whenever it is possible,
- **functions_exchangealgorithm.R**: contains functions to run the bayesian estimation (not ready yet),
- **functions_loglikelihood.R**: contains the functions used for estimation of the log-likelihood and AIC, including the calculation of the log likelihood for a simple Dirichlet model (only one effect being the number of groups),
- **functions_Metropolis.R**: contains the function *draw_Metropolis* used for the Metropolis sampler, and auxiliary functions,
- **functions_output.R**: contains the functions to print results of the estimation,
- **functions_permutation.R**: contains functions for permutation tests on partition data,
- **functions_phaseX.R**: contains the functions *run_phaseX* for the first, second, or third phase of the algorithm,
- **functions_utility.R**: contains diverse functions to calculate specific counts of partitions, intra-class correlations, diverse partition statistics etc.

  The code covers the estimation of either single partition observations (cross-sectional estimation) and multiple partitions (longitudinal estimation). The functions adapted to the second type of data are indicated with "_multiple" or "_multipleERPM" at the end of their names.

2. Model specification

2.1. Objects definition

- Node attributes should be contained in a data.frame, with each column being an attribute (integer or character).
- Networks should be simple full matrices.
- Effects should be a list with first element called "names" including a list of effect names, and a second "objects" element containing the name of the object needed to calculate the effects (for example, column name in the nodeset if it's an attribute effect, or name of a matrix).
- We add objects (matrices) into another list, with "name"and "object" attribute (to recover which matrix to use in a network effect for example).

2.2. Parametrization

For now the following structural effects are implemented:

- **isolates**: number of isolates
- **num_groups**: number of groups
- **num_ties**: number of intra-group ties
- **num_groups_3**: number of groups of size 3
- **num_groups_4**: number of groups of size 4
- **num_groups_5**: number of groups of size 5
- **num_groups_6**: number of groups of size 6
- **num_groups_x_num_nodes**: number of groups multiplied by number of nodes (only useful for longitudinal estimation)
- **num_groups_x_log_num_nodes**: number of groups multiplied by the log of the number of nodes (only useful for longitudinal estimation)
- **num_triangles**: number of intra-group triangle configurations
- **num_fours**: number of intra-group 4-clique configurations
- **num_fives**: number of intra-group 5-clique configurations
- **num_alt_cliques**: sum for x=1 to n of x-clique configurations, with alternated sign (similarly to GWESP type of effects in network models)
- **sizes_squared**: sum of squared sizes
- **sizes_squared_norm**: sum of squared sizes, normalized by the number of groups
- **sizes_degree2**: sum of squared degrees (intra-group ties) for all actors
- **sizes_av_degree**: average degree (intra-group ties) for all actors
- **sizes_av_degree2**: squared average degree (intra-group ties) for all actors
- **product_sizes**: product of all group sizes
- **sum_log_factorials**: sum of logarithms of group sizes

And the (dyadic or individual) attribute effects are:

- **tie**: sum of group intra-ties for a given network
- **tie_X_diff**: sum of differences in attributes for pairs of individuals in the same group AND having a tie in a given network
- **bipartite_tie**: sum of group intra-ties for a given bipartite network
- **bipartite_group**: sum of groups with individuals all linked in a given bipartite network
- **attisolation**: sum of the attributes of isolated people
- **attgroups**: sum of the attributes of in-groups people

- **alter**: sum of individual attributes multiplied by their group size
- **attisolation**: sum of the attributes of isolated people
- **same**: sum of the number of dyads in groups having the same attribute
- **same_norm**: sum of the number of dyads in groups having the same attribute (the attribute was previously normalized)
- **diff**: sum of the absolute differences in attributes for people in the same groups
- **diff_norm**: sum of the absolute differences in attributes for people in the same groups (the attribute was previously normalized)
- **diff_ind**: sum of the lowest differences in attributes per individual compared to other people in the same group
- **diff_ind_norm**: sum of the lowest differences in attributes per individual compared to other people in the same group (the attribute was previously normalized)
- **number_attributes**: sum of different attributes in a group, summed over all groups
- **range**: sum of ranges of an attribute in the groups
- **proportion**: sum of proportion of a binary attribute in the groups
- **number_attributes**: sum of counts of different attributes present in each group

3. Notations

In the following sections, flow charts describe the different steps in the algorithm. The following notations are used:

- P_{obs}: observed partition,
- P_{start}: partition used to initiate the Markov chains,
- p: number of effects (and parameters) of the model,
- α_1: starting values for parameters,
- α_2: values for parameters after the first phase,
- α_3: values for parameters after the second phase,
- N_{phase1}: number of sampled partitions in the first phase,
- S_{phase2}: number of subphases in the second phase,
- m_{phase2}: number of steps in the first subphase of the second phase, the next subphases k contain $(2.52)^k * m_{phase2}$,
- N_{phase3}: number of sampled partitions in the third phase,
- b: number of sampled partitions to realize the burn-in of the Markov chain,
- t: number of sampled partitions to realize the thinning of the Markov chain,
- $r_{truncation1}$: truncation factor to reduce the magnitude of the step from α_1 to the α_2 in Phase 1,
- $r_{truncation2}$: truncation factor to reduce the magnitude of the steps between parameters values in each iteration of Phase 2,
- $a_scaling$: multiplication factor used for the non-diagonal elements of the scaling matrix (reducing the magnitude of these values helps the stability of the algorithm),
- g: gain factor that is used to define the magnitude of steps between parameters values in Phase 2 in the first subphase, we then use $g/2^(k-1)$ for the subphases k to define the gain sequence,
- $p_{transitions}$: probabilities to use the different available relations to move from one partition to the next in the Markov chain,
- DA: binary variable defining whether double-averaging is used in Phase 2 (i.e., at each iteration between parameters, we no longer use the previous value of the parameters but the average of previous values),
- C: covariance matrix of the sampled statistics,
- D: scaling matrix, i.e., the covariance matrix with non-diagonal elements multiplied by $a_scaling$.

4. Simulation

- The code implements a Metropolis-Hastings algorithm to sample from the partition distirbution. Because the proposals in the algorithm are not symmetric (as they often are in this type of algorithm), we must include in the Hastings ratio the ratio of possible neighbor partitions between the old and the newly sampled partition, depending on the proposal used.
- When the options for allowed sizes and simulated sizes are given, the Metropolis sampler goes through the partitions allowed by the simulated sizes option, but samples only the ones that have sizes in the allowed sizes. This has to be done when the space of allowed partitions is disconnected or ill-connected to cover better the space. It can be done by adding the sizes just smaller or bigger than the limits, and it often helps to allow singletons.
- Warning: Right now only size restrictions with a minimum size and a maximum size work¿
- The neighborhood argument is a probability vector, indicating how often each of the three following proposals should be used. The first type of proposal is to suggest a partition created from the current partition by swapping two actors in different groups (possibly isolates). The second proposal consists in moving from one partition to the next by merging two groups, or by dividing one group into two. The third proposal consists in moving one actor from its current group to another group (possibly an isolate). The first proposal shouldn't be used alone as it does not allow a full sampling of the space, but the second and third can. A combination of them is possible.
- In terms of computational time, it is more efficient to use only one proposal at a time (usually the second is better and also faster). But often, this will not be good enough for the mixing of the chain, so it should at least be combined with the first neighborhood. A combination of the three might be needed, but it is computationally expensive.
- When combining different proposal, the Hasting ratio no longer includes the mere ratio of number of possible neighbor partitions for the current and proposed partitions for one proposal, but a ratio that takes into account the probabilities of each proposal and the neighbor sizes for all proposals that can link the two partitions (see code for detailed ratio).
- Sometimes, it will be difficult to sample if the model favours partitions having sized within the simulated sizes option but not within the allowed sizes option. In this case it can useful to change the option of simulated sizes.

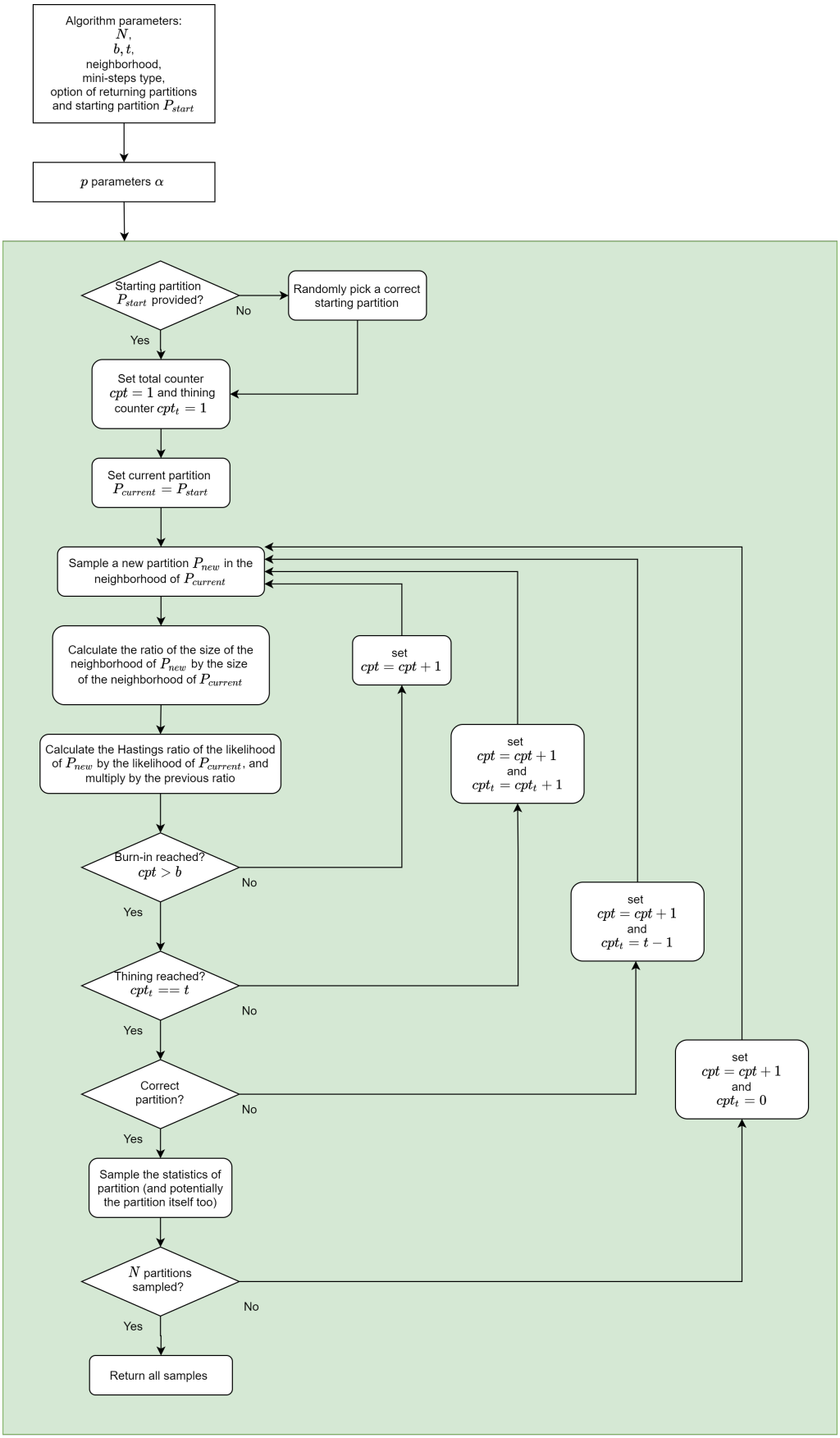

Figure 1. Flowchart for sampling partitions with the Metropolis-Hastings algorithm.

5. Estimation

5.1. General algorithm

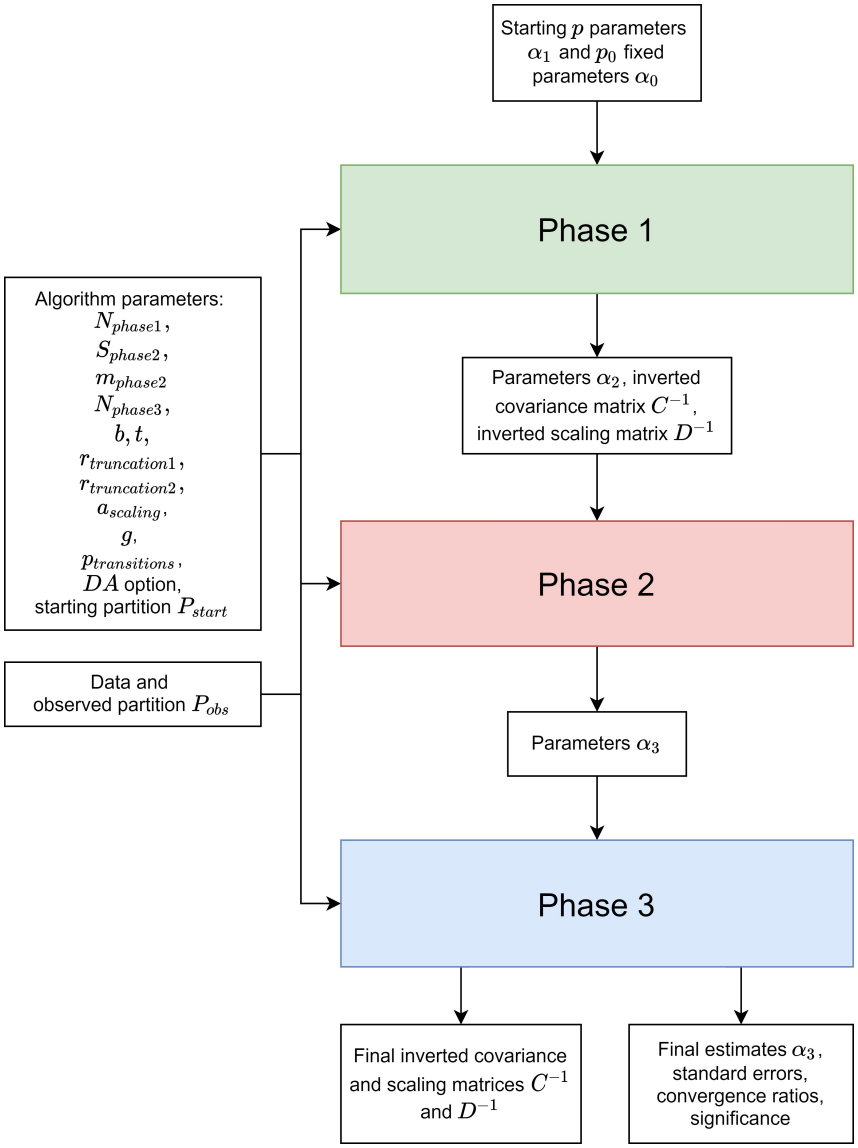

Figure 2. Flowchart for the general estimation algorithm.

5.2. Algorithm for phase 1

- The starting partition has to respect the allowed sizes given to the estimation.
- The length of phase 1 does not need to be very long, the calculation of the co-variance is not very sensitive.
- The variable $a_{scaling}$ is used to reduce the influence of non diagonal elements of the scaling matrix (before inverting it!). If it's zero, the estimation in phase 2 might fluctuates less around the parameter space, so it helps keeping things under control, but it might not converge exactly to what we want.
- The truncating factor $r_{truncation1}$ helps avoiding making too large steps when estimating the parameter in case some elements in the co-variance matrix are too large; the convergence will be slower but safer.
- Co-variance and scaling matrices are inverted once for all here.
- It's better to skip this phase if we've already ran the estimation once and have better estimates and co-variance matrix from a previous phase 3 anyway.

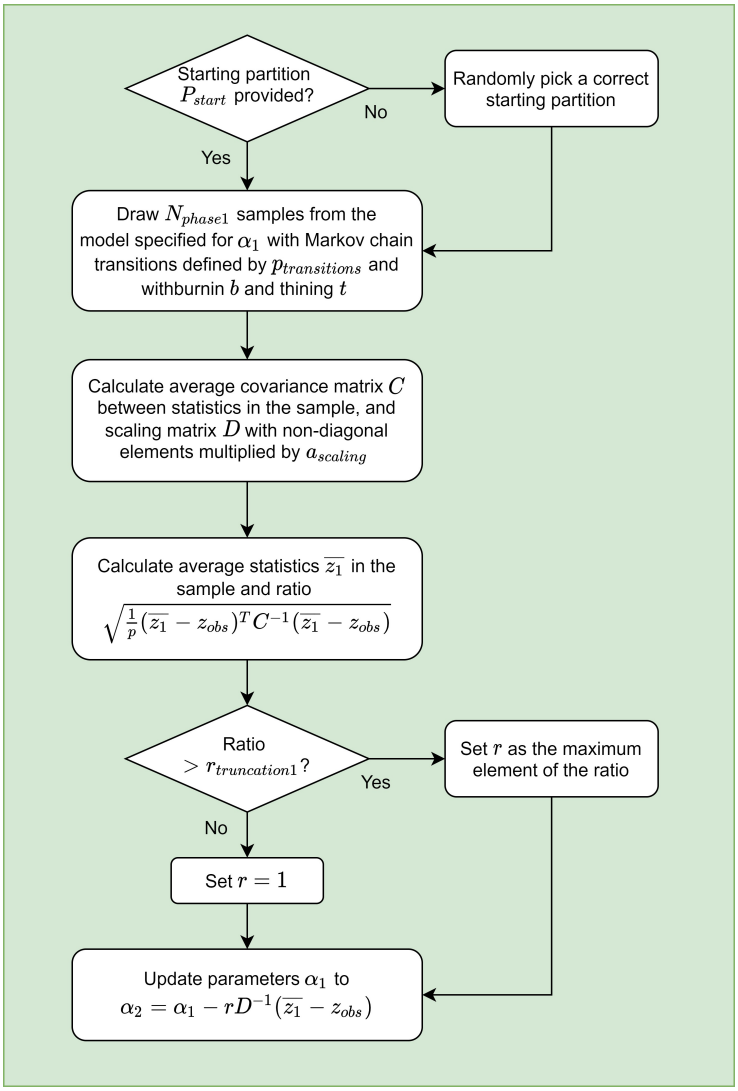

Figure 3. Flowchart for Phase 1 of the main algorithm.

5.3. Algorithm for phase 2

- For each step we start from a random matrix (unless we provided a starting partition), but from one sub-step to the next we keep a partition generated by the previous chain.
- Again the truncating factor $r_{truncation2}$ helps avoiding making too large steps when estimating the parameter in case some elements in the co-variance matrix are too large; the convergence will be slower but safer. This is always calculated from the co-variance and note the scaling matrix!
- Double-averaging can be used to average the estimates over a step AND over all sub-steps of one step. It can sometimes stabilize the estimation procedure. Double-averaging is only when there are no fixed parameters for now.
- A step ends when along all sub-steps we sampled statistics going above AND below the observed statistics (that's what we call statistics "crossing"). Additionally, the parameters m_{phase2} help controlling that each step lasts a bit but not too long even if statistics do not cross. The minimum length is $m_{phase2}(2.52)^k$ and $m_{phase2}(2.52)^k+200$ (so the subphases get longer and longer, without threatening the convergence of the Robbins-Monro procedure).
- When we have a bad starting estimates for the parameters, it can help to have many steps, but when we are close we can reduce to a couple of them, potentially with long sub-steps.

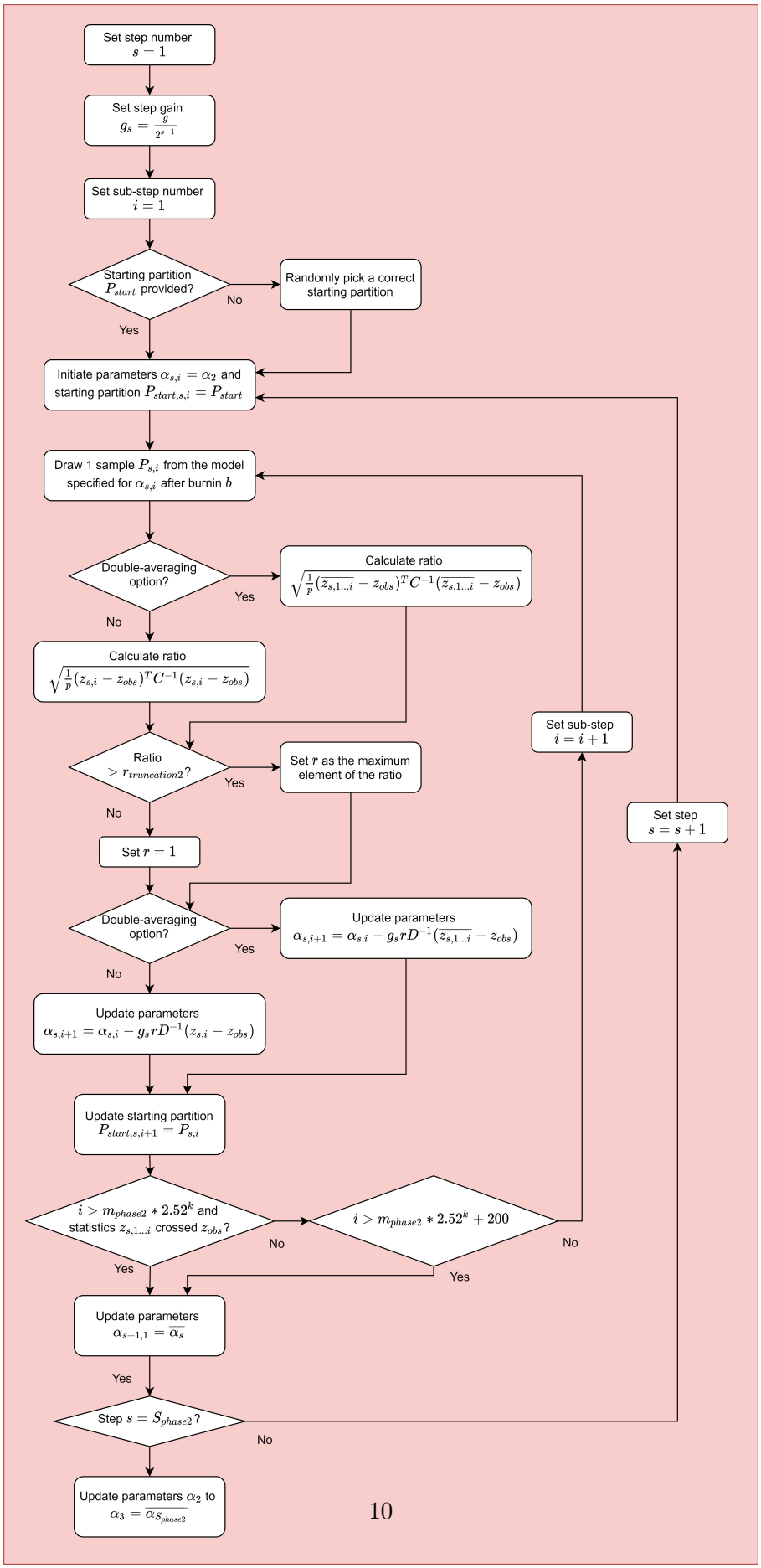

Figure 4. Flowchart for Phase 2 of the main algorithm.

5.4. Algorithm for phase 3

- It helps to keep the co-variance and scaling calculated from this step to a next estimation.
- When we are close to a good estimate for the parameters, it helps to have a very large sample in this phase, to estimate correctly standard errors and convergence ratios.

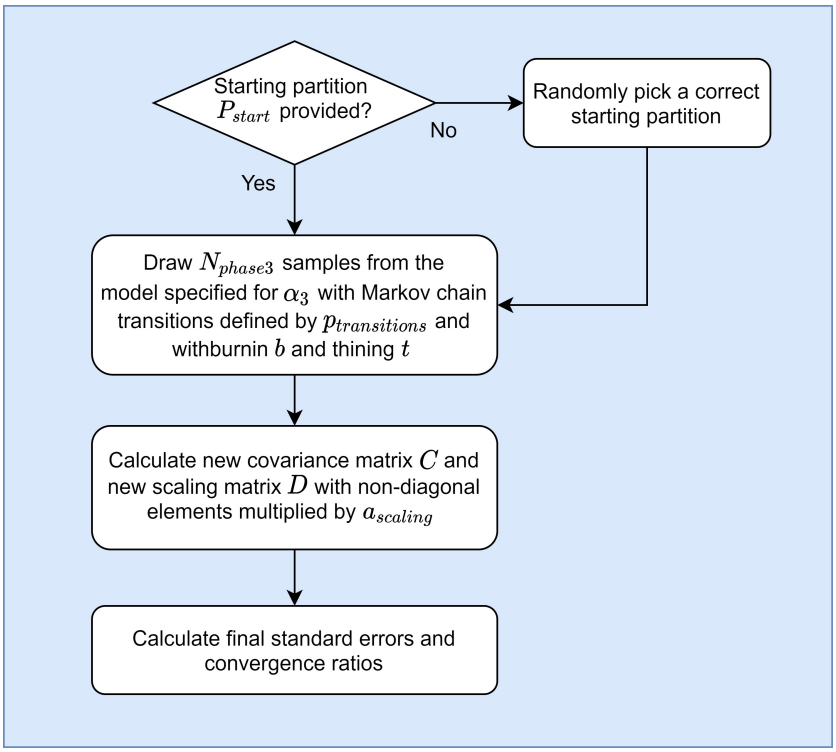

Figure 5. Flowchart for Phase 3 of the main algorithm.

6. Model diagnostics

6.1. Goodness of fit

The example script shows a few examples on how to calculate and visualize the fit of the model regarding certain statistics. It makes use of the simulation engine detailed earlier.

6.2. Estimation of a log-likelihood and AIC

- This procedure can take a while, because each step requires a good sample (just like phase 3), and in the case of restricted sizes sampling is not that easy.
- This can only work if the model include the statistic number of groups, because we can calculate easily the exact likelihood of a model including only this statistic (basically a Dirichlet partition model).
- The α_0 parameters should be a vector of zeros, except for the number of groups effect that should contain the exact estimate (easily found with an optimisation function and using the utility functions to calculate the denominator of Dirichlet partition models).
- It helps to plot the distributions of sampled statistics in the different steps, and follow the "frog-leaping"strategy: the algorithm only works well if the jump from one step to the next is reasonable, meaning if the distributions of statistics have a sufficiently large overlap.

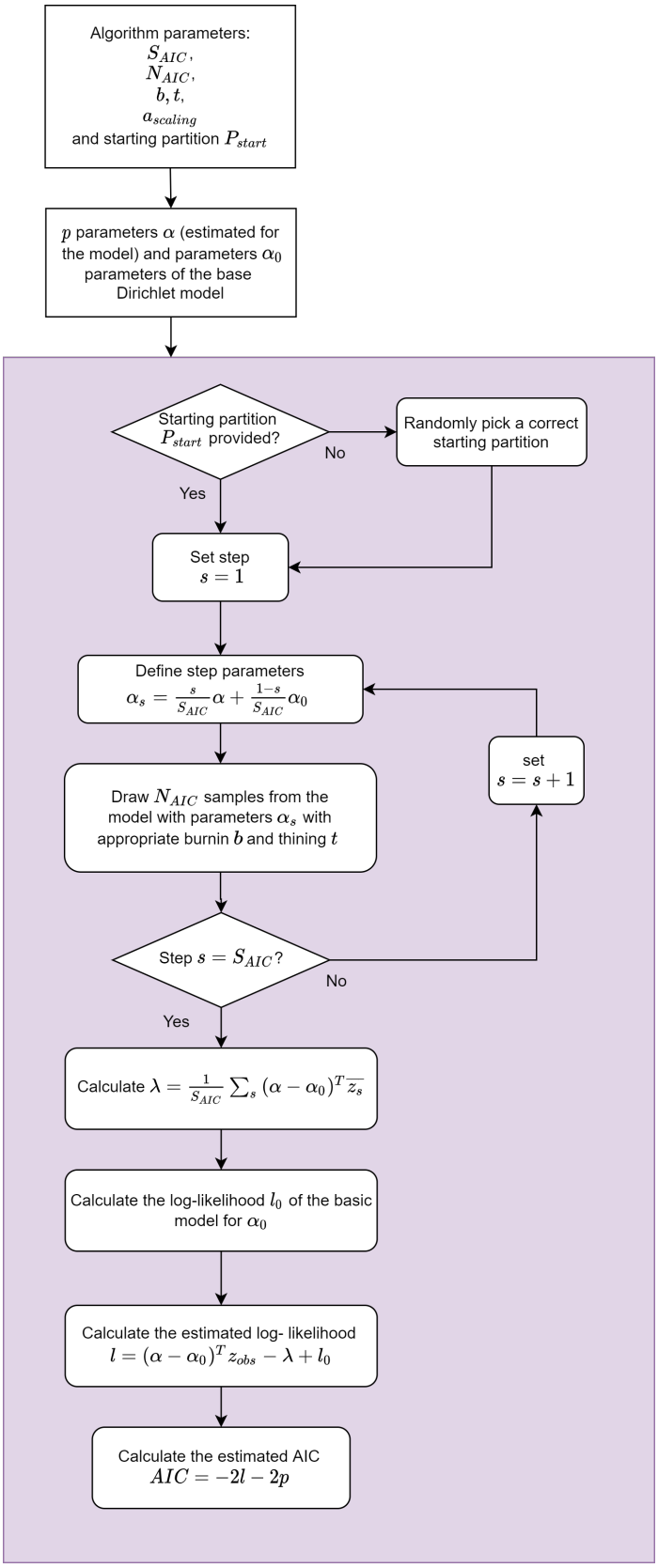

Figure 6. Flowchart for the path-sampling algorithm to estimate the log-likelihood of a given model.

7. Notes

The procedure is heavily inspired from procedures used for Exponential Random Graph Models and the three main references below, and relies on a large amount of great advice from Tom A.B. Snijders.

References

Hunter, D. R., Goodreau, S. M., & Handcock, M. S. (2008). Goodness of fit of social network models. *J. Am. Stat. Assoc.*, *103*(481), 248–258.

Lusher, D., Koskinen, J., & Robins, G. (2013). *Exponential random graph models for social networks: Theory, methods, and applications*. Cambridge University Press.

Snijders, T. A. B. (2002). Markov chain monte carlo estimation of exponential random graph models. *J. Soc. Biol. Struct.*, *3*(2), 1–40.