Manual for MoNAn in R

Per Block, Nico Keiser

University of Zurich, Department of Sociology

April 2024
Manual version 2

Abstract

MoNAn (Mobility Network Analysis) is a package for the statistical environment R that implements the model for the analysis of mobility networks outlined in Block et al. (2022). The purpose of the package and model is to analyse the structure of mobility, incorporating exogenous predictors pertaining to individuals and locations known from classical mobility analyses, as well as modelling emergent mobility patterns akin to structural patterns known from the statistical analysis of social networks.

Since functionality and recommendations are consistently improved, the manual and the package are frequently updated; please make sure you stay up-to-date with the latest developments.

Version 2 of the manual is aligned with version 1.0.0 of MoNAn. Note that there are some important changes in the use of MoNAn compared to version 0.1.3.

Contents

1 General information

The R package MoNAn is an implementation of the statistical model for the analysis of mobility networks described in Block et al. (2022). It enables researchers to apply the described model to their own research. When using MoNAn, please remember it is a software written and maintained by academics. While we perform extensive checks, we cannot guarantee the absence of mistakes and cannot offer full-time support to users.

User support for the package is consistently built and improved. Currently, the main resources for using MoNAn are this manual, the help-files in the R package itself, and the github page, which provides an example script and other resources. The package homepage, currently hosted here, provides further information about materials to help using the package, as well as teaching workshops. In case none of the resources provide the help needed, please contact the package maintainer (Per Block) at his institutional email address (per.block at uzh.ch).

1.1 Accessing MoNAn

MoNAn is an extension to the statistical environment R (R Core Team, 2023); thus, it only works within R. RStudio (RStudio Team, 2020) has some useful features that facilitate its use. MoNAn is hosted on github and freely available at github.com/stocnet/MoNAn. To install MoNAn from the github repository, type the following in your R/RStudio console:

```
# install.packages("remotes")
remotes::install_github("stocnet/MoNAn")
```

As of 30 Aug 2023 MoNAn is also available on CRAN. To get the CRAN version, type the following in your R/RStudio console:

```
install.packages("MoNAn")
```

We recommend installing the github version. The advantage of installing MoNAn from github is that you will always stay up-to-date with the latest developments, in particular, new functionality and effects.

1.2 Acknowledgements

The initial R functions that form the basis of the package were programmed by Christoph Stadtfeld and Per Block. Nico Keiser supported transforming the code into a proper R package. Per Block maintains the package and is responsible for its development from infancy. Further researchers that contributed to the model and package development include Marion Hoffman and Garry Robins. Helpful feedback came from the Duisterbelt group, the Groningen network group, and the Zurich network group. We are grateful for the support by an ETH fellowship of the ETH Zurich, and an Eccellenza fellowship of the Swiss National Science Foundation.

1.3 Referencing MoNAn

When using MoNAn, please reference this manual and underlying model as appropriate. The reference to this manual is:

Block, P., Keiser, N. (2023). *Manual for MoNAn in R.* Zurich: University of Zurich, Department of Sociology. doi:10.31235/osf.io/8q2xu

The model is introduced in:

Block, P., Stadtfeld, C., and Robins, G. (2022). A statistical model for the analysis of mobility tables as weighted networks with an application to faculty hiring networks. *Social Networks*, 68, 264–278.

2 Model overview

2.1 Intuition

This section gives a cursory overview of the statistical model underlying MoNAn. For a more comprehensive treatment, please refer to Block et al. (2022).

The purpose of the model to analyse mobility networks is to understand what regularities are present in the mobility of individuals between locations. Found regularities tend to be interpreted in terms of individual actions, representing preferences and constraints that determine mobility.

Regularities that can be modelled include exogenous and endogenous predictors. Exogenous predictors relate to the match of individual and location characteristics. Two of the most relevant examples for exogenous predictors are that (i) individuals with certain characteristics move to certain types of locations, and (ii) mobility tends to be present between locations that share, or are similar on some characteristic.

Endogenous predictors describe (network-like) structures of mobility. Such structures are composed of multiple observations and, as such, represent the *inter*-dependence of mobility of different individuals. A prominent example of such endogenous structures is reciprocation, that is, some measure of how much the extent of mobility from location a_1 to location a_2 correlates with the extent of mobility from location a_2 to location a_1.

The distinguishing feature of this model for mobility networks is its ability to represent endogenous predictors, which is impossible in many standard statistical models, as it systemically violates the assumption of independent observations. Nevertheless, the model is in the tradition of classical log-linear models for mobility tables. In fact, when no endogenous predictors are included, it reduces to such a standard mobility model. At the same time, the model draws heavily from insights of statistical network models, in particular the Exponential Random Graph Model (ERGM) (Lusher et al., 2013). In case no characteristics describing individuals are used, the mobility network model can be seen as an ERGM for weighted networks of count data with exogenously given row marginals.

2.2 Mathematical formulation

2.2.1 Notation

The following notation is used in this manual, adopted from Block et al. (2022).

- A: Origins and destinations in a mobility network are defined by n locations $A = (a_1, a_2, ..., a_n)$.
- B: Between the locations, r individuals $B = (b_1, b_2, ..., b_r)$ are mobile.
- We refer to A and B as `Nodesets` in the implementation of the model.

Locations and individuals can have exogenous attributes.

- v: Monadic covariates of locations are denoted v.
- u: Dyadic covariates describing exogenous relations between locations are denoted u.
- w: Monodic covariates of individuals are denoted w.

The state of the mobility network is defined by two vectors

- o: An origin vector $o \in \{1, ..., n\}^r$, in which $o_i = j$ if node a_j is the origin of individual i.
- d: A destination vector $d \in \{1, ..., n\}^r$, in which $d_i = h$ if node a_h is the destination of individual i.

The vectors o and d jointly form an edgelist that determines the mobility of the resources.

- $\mathbf{x}$: The variables A, B, v, w, o and d fully define a mobility network, for which we use the shorthand notation $\mathbf{x}$.
- This shorthand notation when used with indices $\mathbf{x}_{jh}$ refers to the number of individuals moving from location j to location h.

2.2.2 The model

We describe any observed or potentially observable mobility network $\mathbf{x}$ by so-called statistics $s(\mathbf{x})$. These statistics are often counts of occurrences of specific types of configurations, e.g., how many individuals move between locations identical on some covariate (exogenous), or how how many transitions from location a_2 to location a_1 co-occur for a transition from a_1 to a_2 (endogenous). These statistics have the function of independent variables in the model[1]. In the implementation in MoNAn, these statistics are referred to as *effects* (see Section 5).

Since o, v, w, A and B are exogenously given in the model, variation in the values of the statistics for different realisations of $\mathbf{x}$ come from different realisations of d. Thus, we can define statistics as follows:

$$s(\mathbf{x}) = f(d|o, v, w, A, B). \tag{1}$$

A list of diverse statistics is presented in Section 5.2. Jointly with a statistical parameter θ, a set of statistics $s(\mathbf{x})$ defines the quality function $Q(\mathbf{x})$ as a linear predictor:

$$Q(\mathbf{x}) = \sum_k \theta_k s_k(\mathbf{x}), \tag{2}$$

[1] In case of exogenous statistics, they are identical to independent variables of standard regression models.

In case of a positive (negative) θ_k, the value of the quality function increases (decreases) with a higher count of the statistic $s_k(\mathbf{x})$.

For a given θ, we define the probability to observe any possible realisation of a a mobility network as follows:

$$Pr(\mathrm{X} = \mathbf{x}) = \frac{\exp(Q(\mathbf{x}))}{\kappa}, \tag{3}$$

where κ is a normalising constant that depends on $\mathbb{C}$, which is the set of all possible configurations that a mobility network $\mathbf{x}$ can take (i.e., any possible realisation of d):

$$\kappa = \sum_{\mathbf{x}' \in \mathbb{C}} \exp(Q(\mathbf{x}')). \tag{4}$$

The intuition behind the model is that mobility networks with higher counts of statistics that are associated with a positive θ are more likely to be observed. In practise, however, we do not start from a set of θ's and are interested in the probability to observe different realisations of $\mathbf{x}$. Instead, we tend to start from an observed mobility network and are interested in *estimating* θ's based on the data. The estimated values give an indication whether the associated statistic is present in the data more than expected 'by chance'.[2] Such over- or under-representation is the basis for interpretation that can shed light on the processes that brought about the observed mobility network.

[2]The meaning of 'by chance' is left purposefully vague, as it requires a deeper treatment than this introduction wants to provide.

3 Using the manual

This manual provides comprehensive guidance to using MoNAn beyond what is available from the help-files within the R library.

A high level overview of how the different functions in MoNAn are connected is provided in Figure 1. This flow-chart shows the steps a user must take to get from raw data (yellow rounded rectangles at the top of the page) to a completed estimation (blue rectangle called 'myResDN') and possibly beyond. All functions of the package and how they fit in the analysis work-flow are included in Figure 1.

The use of each function is documented in this manual. Especially more complicated functions (like `monanAlgorithmCreate` with many optional parameters), and functions that are involved in their use (like `addEffects` that makes use of further effect functions in its specification) are explained in detail.

Each function covered in the manual is – as far as necessary – explained theoretically, and subsequently linked to the example illustrating its use. The examples are identical to the ones specified in the `MoNAnExampleScript.R` and the `README`, both available on the MoNAn github page.

3.1 Example data

To illustrate the package use in the MoNAn documentation, exemplary input data in standard R format (`mobilityData`) and outcome objects (`myOutcomeObjects`) are provided in the package.

The raw example data is synthetic (i.e., made up). This fictitious example contains 17 organisations representing a labour market that are located in two regions (north and south). 742 workers are employed in these organisations at two timepoints. Some are mobile while others work in the same organisation at both timepoints.

Organisational membership of where each worker is working at time 1 and at time 2 is stored in the object `mobilityEdgelist`. The workers' sex represents an individual characteristic stored in `indSex`. Each organisation's size (`orgSize`) and location (`orgRegion`) are included as continuous and categorical characteristics of locations, respectively. The former represents a composite measure of the organisations' assets and revenue.

For reference (and to save time), example outcome objects (`myOutcomeObjects`) of the package's main functions are stored. The data can be accessed as follows:

```
?mobilityData
# load raw data objects
data('mobilityEdgelist'')
# overview of the outcome objects
?myOutcomeObjects
```

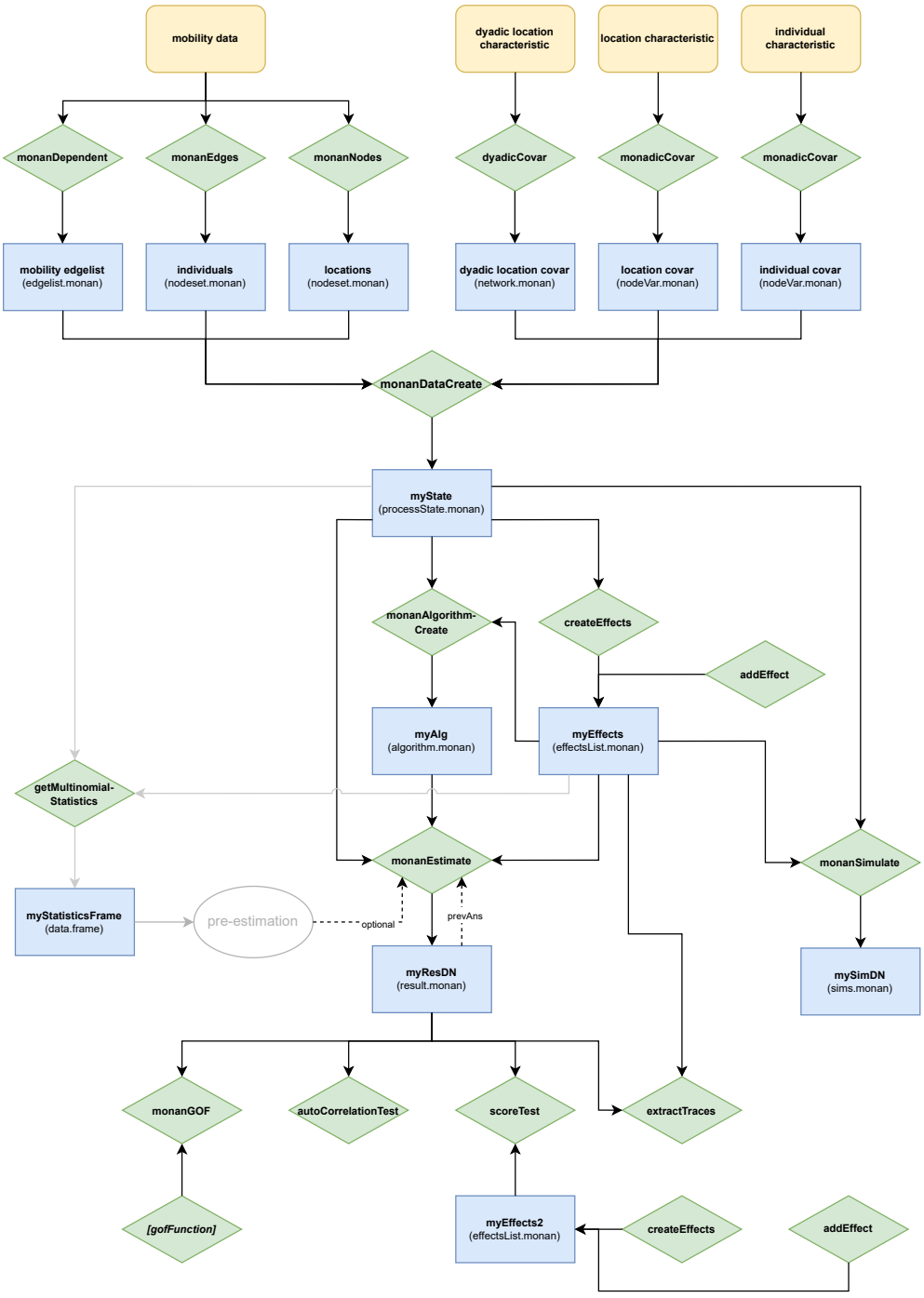

Figure 1: Schematic overview of MoNAn functions and objects
Yellow rounded rectangles indicate raw data imported to MoNAn, green diamonds represent MoNAn functions, blue rectangles are MoNAn data objects. Arrows indicate the input needed for a function and the outcome objects created by functions; e.g. to obtain the results of an estimation (called myResDN in the flowchart), the function `monanEstimate` needs to be executed, which in turn requires a state (myState), effects (myEffects), and an algorithm (myAlg) – optionally, the results of a pre-estimation or a previous estimation can be used as input.

4 Data

This section outlines how to transform raw data into a format that MoNAn uses internally, and how to combine this into a *process state*. It is assumed that the user already imported all data into R and it is present in base R format (i.e., as a series of data.frames, matrices, and/or vectors).

4.1 Input data

For any MoNAn analysis, it is necessary to specify *nodesets*, i.e., who is mobile between what locations, and a *mobility edgelist*, representing the mobility network under analysis. In Figure 1, this is referenced on the top left in the arrows originating from the yellow rounded rectangle 'mobility data'. Additionally, *monadic* or *dyadic covariates* of individuals and locations can be specified. Their inclusion is optional. In Figure 1, they are depicted on the top right.

4.1.1 Defining the units of analysis: Nodesets

The locations and individuals that make up the mobility network need to be specified and named. Typical nodesets denoting locations are 'organisations' or 'occupations'; these are specified using the function `monanNodes`. Typical nodesets of individuals are 'workers' or simply 'people'; these are specified using the function `monanEdges`. The use of both functions is identical.

Function names: `monanEdges` and `monanNodes`

Arguments:

- `x`. *necessary*. The size of the nodeset, i.e., the number or organisations or individuals in the data.
- `considerWhenSampling`. *optional*. Only for use in special cases. A boolean/ logical vector of the length of the nodeset. If the nodeset indicates a location, `considerWhenSampling` indicates whether the location is a possible destination, or is only an origin (e.g., a training facility). Entries in the vector of locations that cannot be a destination are FALSE. If the nodeset indicates mobile individuals, `considerWhenSampling` indicates whether their mobility should be modelled or whether it is structurally determined, that is, their mobility is exogenously defined and does not follow the same logic as the mobility of everybody else.

Usage:

```
# extract number of ind. and org. from the mobility data
N_ind <- nrow(mobilityEdgelist)
N_org <- length(unique(as.numeric(mobilityEdgelist)))
```

```
# create monan nodesets
people <- monanEdges(x = N_ind)
organisations <- monanNodes(x = N_org)
```

4.1.2 The dependent variable: Edgelist

The mobility network itself is specified by an edgelist, that is, a list of the origin and destination of each individual in the data. The names of the nodesets that make up the mobility network are used in the specification of the edgelist. The name of the edgelist as determined by the user will be used later to reference the dependent variable.

Function name: `monanDependent`

Arguments:

- `el`. *necessary.* The mobility data in the format of an edgelist. An edgelist is a matrix (or data.frame) containing 2 columns and as many rows as individuals. The first column indicates the origin of a person/resource, the second row the destination.
- `nodes`. *necessary.* This references the names of the `nodes`, i.e., locations.
- `edges`. *necessary.* This references the names of the `edges`, i.e., individuals.

Usage:

```
transfers <- monanDependent(mobilityEdgelist,
                            nodes = "organisations",
                            edges = "people")
```

4.1.3 Predictor variables: Monadic covariates

Monadic covariates describe attributes of locations and/or individuals that are meant to be used in the estimation of the model. Examples for attributes of organisations as examples of locations are their size (continuous) or region (categorical). Examples of individual attributes include sex (categorical) or age (continuous).

In the creation of covariates in the process of a MoNAn analysis, in some cases it is necessary to have the later model specification in mind. This is the case if there are more than 500 locations. In case the covariates describe locations and are later meant to be used to represent homophily, the additional parameters addSame for categorical or addSim for continuous covariates need to be set to TRUE. However, if there are less than 500 locations, addSim and addSame are included automatically. The reasons for this are the way MoNAn calculates effects internally. Note that it is not recommended to include addSame or addSim by default, since it slows the estimation down.

Function name: `monadicCovar`

Arguments:

- `values`. *necessary*. A vector that contains the covariate values of each element of the nodeset. The ordering of the covariate values for individuals must be identical to the ordering in the *edgelist* that specifies the mobility network.
- `nodes`. *necessary unless edges specified*. The nodes to which the covariate applies for location characteristics.
- `edges`. *necessary unless nodes specified*. The edges to which the covariate applies for individual characteristics.
- `addSame`. *optional*. Are there more than 500 locations and will the variable be used to model categorical homophily (e.g., with the `same_covariate` effect)? In this case, addSame needs to be set to TRUE.
- `addSim`. *optional*. Are there more than 500 locations and will the variable be used to model continuous homophily (e.g., with the sim_covariate effect)? In this case, addSim needs to be set to TRUE.

Usage:

```
region <- monadicCovar(values = orgRegion,
                       nodes = "organisations")
size <- monadicCovar(values = orgSize,
                     nodes = "organisations")
sex <- monadicCovar(values = indSex,
                    edges = "people")
```

4.1.4 Predictor variables: Dyadic covariates

Dyadic covariates specify exogenously determined relations between the locations of the data. Examples of such dyadic covariates for the case of organisations can be their geographical distance, co-ownership, or other formal contractual ties between organisations.

Function name: `dyadicCovar`

Arguments:

- `m`. *necessary*. A square matrix containing the network data.
- `nodes`. *necessary*. Which nodeset are the nodes of the dyadic covariate. Usually this will be the locations in the data.

Usage:

```
sameRegion <- outer(orgRegion, orgRegion, "==") * 1
sameRegion <- dyadicCovar(m = sameRegion,
              nodes = "organisations")
```

4.2 The process state

Once all data that will be used in an analysis (the mobility edgelist, the nodesets, and all covariates) are specified according to the functions above, this data is combined into the *process state*. The process state is a data object that is used by MoNAn in the estimation, simulation, etc. In other words, it is the internal format MoNAn uses to deal with data. From here on, MoNAn will not access any of the raw data stored in an R environment. This means that any changes in the raw data necessitates re-running the data create functions from Section 4.1 and re-creating the process state.

Creating a data object – the process state – by the user means a MoNAn analysis requires more steps than, e.g., a linear regression, which directly accesses, e.g., a data frame. The reason to create a dedicated MoNAn data object is that it breaks running a MoNAn analysis into manageable steps that can be performed and debugged one-by-one.

Function name: `monanDataCreate`

Arguments:

- `....` *necessary.* The outcome variable (edgelist), the nodesets, and all covariates that contain the information about the data that will be used in the estimation.

Usage:

```
myState <- monanDataCreate(transfers,
                           people,
                           organisations,
                           sameRegion,
                           region,
                           size,
                           sex)

myState
```

The process state has a dedicated print function to inspect the data that is included in a current process state.

5 Model specification

Model specification in the mobility network model and, accordingly, MoNAn is not trivial. While the choice of exogenous statistics/effects is straight-forward and mirrors that of standard log-linear models for mobility, the choice of endogenous statistics is less researched. Users are recommended to take substantive considerations and previous network literature into account. Some more thoughts on model specifications are elaborated in Section 5.3.

5.1 The effects object

The model is specified in an *effects object*. This unusual name is chosen, because the predictors of the models are called 'effects'. The core of an effects object are effect functions. These effect functions are in fact proper R functions that are called internally when the effects object is created. Thus, the 'tab' button can be used to complete effect names in an R script.

A model is specified in two steps, first an effects object is created and, second, effects are added one by one to this object. The use of pipes (|>) comes in handy at this stage.

Function name: `createEffects`

Arguments:

- `state`. *necessary*. The state that will be modelled with this effects object.

Usage:

```
myEffects <- createEffects(myState)
```

Each effect to be included is added one by one using the function `addEffect` including its name and and potentially the location/individual attributes to which it refers. Attributes are called 'node.attribute' or 'edge.attribute' in model specifications of MoNAn.

Function name: `addEffect`

Arguments:

- `effectsObject`. *necessary*. The effects object to which new effects are added.
- `effectName`. *necessary*. The name of the additional effect.
- `....` *optional*. As needed the node.attribute, edge.attribute, or parameter.

Usage:

```
myEffects <- addEffect(myEffects, myEffects, reciprocity_min)
myEffects <- addEffect(myEffects, dyadic_covariate,
                       node.attribute = "sameRegion")
myEffects <- addEffect(myEffects, loops_resource_covar,
                       edge.attribute = "sex")

# together with createEffects and pipes, this can be simplified as

myEffects <- createEffects(myState) |>
  addEffect(reciprocity_min) |>
  addEffect(dyadic_covariate, node.attribute = "sameRegion") |>
  addEffect(loops_resource_covar, edge.attribute = "sex")

myEffects
```

Effects objects have dedicated print functions to inspect the specification of a given effects object.

5.2 Implemented effects

MoNAn comes with a set of implemented effects that represent the most common predictors of mobility network evolution. The set of implemented effects is consistently expanded; experienced R users might want to program additional effects. Such effects can be directly used if the effects functions are in the global R environment.

The list in the following subsections introduces all effects by its name, a brief description, its mathematical formula and its internal MoNAn name and usage. The mathematical formulation uses the notation introduced in Section 2.2.1. The usage is shown taking the attribute names from the example data.

5.2.1 Exogenous effects

1.1 *Name:* **alter covariate**

Description: Are locations higher on some attribute v more popular targets of mobility? E.g., do workers have a tendency to move to larger organisations?

Formula:

$$s(\mathbf{x}) = \sum_{i \in B} v_{d_i}. \tag{5}$$

Usage:

```
myEffects <- addEffect(myEffects, alter_covariate,
                       node.attribute = "size")
```

1.2 *Name:* **same covariate**

Description: Is mobility more likely between locations that are identical on some attribute v? E.g., is mobility more likely between organisations that are located in the same region?

Formula:

$$s(\mathbf{x}) = \sum_{i \in B} I(v_{o_i} = v_{d_i}), \tag{6}$$

where I is an indicator function that takes the value 1 in case the condition in parentheses is true and 0 otherwise.

Usage:

```
myEffects <- addEffect(myEffects, same_covariate,
                       node.attribute = "region")
```

1.3 *Name:* **covariate similarity**

Description: Is mobility more likely between locations that are similar on some attribute v? E.g., is mobility more likely between organisations that have a similar size?

Formula:

$$s(\mathbf{x}) = \sum_{i \in B} 1 - \frac{|v_{o_i} - v_{d_i}|}{\Delta}, \tag{7}$$

where Δ is the maximum difference between covariates v of any pair of locations. This normalises the similarity between two locations to a value between 0 and 1.

Usage:

```
myEffects <- addEffect(myEffects, sim_covariate,
                       node.attribute = "size")
```

1.4 *Name:* **dyadic covariate**

Description: Is mobility between locations predicted by the dyadic covariate u? E.g., is mobility likely between organisations that are in the same region? Note that in many cases dyadic covariates can convey the same information as the 'same covariate' or the 'covariate similarity' effects.

Formula:

$$s(\mathbf{x}) = \sum_{i \in B} u_{o_i d_i}, \tag{8}$$

Usage:

```
myEffects <- addEffect(myEffects, dyadic_covariate,
                       node.attribute = "sameRegion")
```

1.5 *Name:* **dyadic covariate * individual attribute**

Description: Is mobility between locations predicted by the dyadic covariate u weighted by the individual covariate w? E.g., is mobility of women more likely between organisations that are in the same region?

Note that this effect can be used to also model the interaction between the 'same covariate'/'covariate similarity' effect and individual attributes, since sameness and similarity between locations can be translated into dyadic covariates.

Formula:

$$s(\mathbf{x}) = \sum_{i \in B} u_{o_i d_i} w_i. \tag{9}$$

Usage:

```
myEffects <- addEffect(myEffects,
                       dyadic_covariate_resource_attribute,
                       node.attribute = "sameRegion",
                       edge.attribute = "sex")
```

1.6 *Name:* **alter covariate * individual attribute**

Description: Do individuals with some individual attribute w tend to move to locations with some location characteristic v? E.g., do women move to larger organisations than men?

Formula:

$$s(\mathbf{x}) = \sum_{i \in B} v_{d_i} w_i. \tag{10}$$

Usage:

```
myEffects <- addEffect(myEffects,
                       resource_covar_to_node_covar,
                       node.attribute = "size",
                       edge.attribute = "sex")
```

5.2.2 Endogenous effects

2.1 *Name:* **loops**

Description: Do individuals stay in their location of origin, compared to going to a different location?

Formula:

$$\begin{aligned} s(\mathbf{x}) &= \sum_{i \in B} I(o_i = d_i) \\ &= \sum_{j \in A} \mathbf{x}_{jj}, \end{aligned} \tag{11}$$

where I is an indicator function that takes the value 1 in case the condition in parentheses is true and 0 otherwise. Note that here the matrix representation $\mathbf{x}_{jj}$ is used for the first time.

Usage:

```
myEffects <- addEffect(myEffects, loops)
```

2.2 *Name:* **geometrically weighted loops**

Description: Do individuals stay in their current location, in case many other from their current location also stay? This effect tests whether the 'benefit' of staying in the origin location depends on the number of others also staying. Note that this effect should be modelled alongside the **loops** effect. For a deeper introduction of geometrically weighted effects, see Snijders et al. (2006).

Formula:

$$s(\mathbf{x}) = \sum_{j \in A} g_{cum}(\mathbf{x}_{jj}), \tag{12}$$

where

$$g_{cum}(y) = \sum_{k=1}^{y} (y - k) e^{-log(\alpha)k}. \tag{13}$$

The user-defined parameter α guides the decay in the additional contribution of a further count of y. For $\alpha = 1$, $g_{cum}(y) = y(y-1)/2$, for larger values the increase in $g_{cum}(y)$ approaches linearity for higher values of y. For very large values of α, $g_{cum}(y) \sim y$ for $y \geq 1$. The default value for α is 2, meaning that each additional count on y contributes as much as the previous count, plus half the difference between the previous and pre-previous count (for $y > 2$). In other words, the growth in the contribution to the statistic by each count decreases by factor 2.

Usage:

```
myEffects <- addEffect(myEffects, loops_GW)
```

2.3 *Name:* **loops by additional origin**

Description: In cases when individuals have more than one origin (for example in intergenerational occupational mobility, when people have two parents), this implements the loops effect for a second origin coded as an individual covariate.

Formula:

$$s(\mathbf{x}) = \sum_{i \in B} I(w_i = d_i) \tag{14}$$

where I is an indicator function that takes the value 1 in case the condition in parentheses is true and 0 otherwise.

Usage:

```
myEffects <- addEffect(myEffects,
                       loops_additional_origin,
                       edge.attribute = "second_or")
```

2.4 *Name:* **loops x loops by additional origin**

Description: In cases when individuals have more than one origin (for example in intergenerational occupational mobility, when people have two parents), this implements the loops effect for a second origin coded as an individual covariate, interacted with the loops effects, i.e., the effect of both origins being identical on staying in the same location.

Formula:

$$s(\mathbf{x}) = \sum_{i \in B} I(w_i = o_i = d_i) \tag{15}$$

where I is an indicator function that takes the value 1 in case the condition in parentheses is true and 0 otherwise.

Usage:

```
myEffects <- addEffect(myEffects,
                       loops_x_loops_additional_origin,
                       edge.attribute = "second_or")
```

2.5 *Name:* **concentration**

Description: Is there a bandwagon effect in mobility, i.e., do mobile individuals move to locations that are the destination of many others from their origin?

Formula:

$$s(\mathbf{x}) = \sum_{j \neq h \in A} \mathbf{x}_{jh}^2, \tag{16}$$

Note: This effect models a quadratically increasing benefit to more individuals on the same path – this is prone to degeneracy! In many cases, the **geometrically weighted concentration** effect might be a better modelling choice.

Usage:

```
myEffects <- addEffect(myEffects, concentration_basic)
```

2.6 *Name:* **concentration squared**

Description: Is there a bandwagon effect in mobility, i.e., do mobile individuals move to locations that are the destination of many others from their origin, and is there a non-linear relation between more individuals on the same path and attractiveness for further individuals?

Formula:

$$s(\mathbf{x}) = \sum_{j \neq h \in A} \mathbf{x}_{jh}^2 \mathbf{x}_{jh}^2, \tag{17}$$

Note: This effect models the square of a quadratically increasing benefit to more individuals on the same path – this is likely to be negative and can only be interpreted properly together with the basic concentration effect. Its usage is to avoid degeneracy in the main concentration effect.

Usage:

```
myEffects <- addEffect(myEffects, concentration_basic_squared)
```

2.7 *Name:* **geometrically weighted concentration**

Description: Is there a bandwagon effect in mobility, i.e., do mobile individuals move to locations that are the destination of many others from their origin? The functional form of this statistic assumes that there are decreasing additional returns to more others on the same mobility path. For example, the probability to choose a mobility path that already contains 20 other individuals is hardly different from a path with 25 other individuals; however, there is a substantial difference in the comparison of paths with 2 or 7 other individuals. For a deeper introduction of geometrically weighted effects, see Snijders et al. (2006).

Formula:

$$s(\mathbf{x}) = \sum_{j \neq h \in A} g_{cum}(\mathbf{x}_{jh}), \tag{18}$$

with $g_{cum}(y)$ defined as in equation 13.

Usage:

```
myEffects <- addEffect(myEffects, concentration_GW)
```

2.8 *Name:* **reciprocity**

Description: Do individuals move to destinations dependent on the number of individuals that move to ego's origin from that destination?

Formula:

$$s(\mathbf{x}) = \sum_{j < h \in A} \mathbf{x}_{jh} \mathbf{x}_{hj}, \tag{19}$$

Note that this effect is prone to near-degeneracy. Using the **geometrically weighted reciprocity** effect might be a better idea in many cases.

Usage:

```
myEffects <- addEffect(myEffects, reciprocity_basic)
```

2.9 *Name:* **geometrically weighted reciprocity**

Description: Do individuals move to destinations that send many individuals to ego's origin? The number of incoming individuals has decreasing returns, that is, every additionally incoming individual influences ego's choice less. For a deeper introduction of geometrically weighted effects, see Snijders et al. (2006).

Formula:

$$s(\mathbf{x}) = \sum_{j \neq h \in A} \mathbf{x}_{jh} \; g_{mar}(\mathbf{x}_{hj}), \tag{20}$$

where

$$g_{mar}(y) = -1 + \sum_{k=0}^{y} e^{-log(\alpha)k}. \tag{21}$$

The user-defined parameter α has the same function as defined in equation 13. The default value for α is 2, meaning that each additional count on y contributes half as much as the previous count (for $y > 0$). Thus with increasing y, $s(\mathbf{x})$ approaches $\mathbf{x}_{jh}$

Usage:

```
myEffects <- addEffect(myEffects, reciprocity_GW)
```

2.10 *Name:* **minimum reciprocity**

Description: Do individuals move to destinations that send more individuals to ego's origin? This version of the effect is the minimum of the moves in either direction, thereby guarding against degeneracy and guaranteeing sample size consistency. It counts the 'raw' number of reciprocated transitions in the mobility network.

Formula:

$$s(\mathbf{x}) = \sum_{j < h \in A} \min(\mathbf{x}_{jh}, \mathbf{x}_{hj}). \tag{22}$$

Usage:

```
myEffects <- addEffect(myEffects, reciprocity_min)
```

2.11 *Name:* **basic transitivity**

Description: Is mobility clustered in groups? Among the three nodes j, h, l, this is represented by the product of the ties $j \to h$, $j \to l$, and h $\to l$.

Formula:

$$s(\mathbf{x}) = \sum_{j<h<l \in A} \mathbf{x}_{jh}\mathbf{x}_{jl}\mathbf{x}_{hl} \tag{23}$$

Note that this will almost always be degenerate. Using the geometrically weighted or minimum version of the effect will often be better.

Usage:

```
myEffects <- addEffect(myEffects, transitivity_basic)
```

2.12 *Name:* **geometrically weighted transitivity**

Description: Is mobility clustered in groups? This is represented by mobility closing two-paths that exist between j and l, where the total number of two-paths is geometrically weighted.

Formula:

$$s(\mathbf{x}) = \sum_{j<l \in A} \mathbf{x}_{jl}\ g_{mar}(\sum_{h \in A} \mathbf{x}_{jh}\mathbf{x}_{hl}) \tag{24}$$

where g_{mar} is defined as in equation 21.

Usage:

```
myEffects <- addEffect(myEffects, transitivity_GW, alpha = 1.1)
```

2.13 *Name:* **minimum transitivity**

Description: Is mobility clustered in groups? This is represented by the minimum of reciprocated mobility being present among three nodes. Using the minimum ensures that the effect is not degenerate and it is sample size consistent.

Formula:

$$s(\mathbf{x}) = \sum_{j<h<l \in A} \min(\mathbf{x}_{jh}, \mathbf{x}_{hj}, \mathbf{x}_{jl}, \mathbf{x}_{lj}, \mathbf{x}_{lh}, \mathbf{x}_{hl}). \tag{25}$$

Usage:

```
myEffects <- addEffect(myEffects, transitivity_min)
```

2.14 *Name:* **netflow transitivity**

Description: Do individuals move in one direction in locally ordered triads? E.g., is there a local hierarchy that individuals follow when moving between locations? The effect is sample size consistent.

Formula:

$$s(\mathbf{x}) = \sum_{j \neq h \neq l \in A} \min(\mathbf{x}_{j \Rightarrow h}, \mathbf{x}_{j \Rightarrow l}, \mathbf{x}_{h \Rightarrow l}). \tag{26}$$

with $\mathbf{x}_{i \Rightarrow j}$ indicating the netflow from i to j defined as

$$\mathbf{x}_{j \Rightarrow h} = \max(0, \mathbf{x}_{jh} - \mathbf{x}_{hj}). \tag{27}$$

Usage:

```
myEffects <- addEffect(myEffects, transitivity_netflow)
```

2.15 *Name:* **present relations**

Description: Do individuals move along many or few paths out of their origin? This models whether individuals have a tendency against being the only one moving to a particular destination from their origin.

Formula:

$$s(\mathbf{x}) = \sum_{j \neq h \in A} I(\mathbf{x}_{jh} > 0). \tag{28}$$

Usage:

```
myEffects <- addEffect(myEffects, present_relations)
```

2.16 *Name:* **loops by indegree**

Description: Are individuals that are in locations with a large inflow more likely to stay in their current location?

Formula:

$$\begin{aligned} s(\mathbf{x}) &= \sum_{j \neq h \in A} \mathbf{x}_{hj} \mathbf{x}_{jj} \\ &= \sum_{j \in A} \mathbf{x}_{+j} \mathbf{x}_{jj}, \end{aligned} \tag{29}$$

Usage:

```
myEffects <- addEffect(myEffects, in_ties_loops)
```

2.17 *Name:* **indegree concentration exponent**

Description: Is there a preferential attachment in the mobility network, i.e., do individuals move particularly to popular destinations?

Formula:

$$s(\mathbf{x}) = \sum_{j \in A} (\mathbf{x}_{+j})^{\alpha}, \tag{30}$$

For a value of $\alpha = 2$, the effect is akin to the in-two-star statistic in ERGMs.
Usage:

```
myEffects <- addEffect(myEffects, in_weights_exponent)
```

5.2.3 Mixed effects

3.1 *Name:* **loops by location covariate**
Description: Are locations with specific attributes 'stickier' than others, i.e., do individuals have a higher propensity to stay in some locations? E.g., are individuals working in organisations in one region less likely to change their employer?
Formula:

$$s(\mathbf{x}) = \sum_{j \in A} v_j \mathbf{x}_{jj} \tag{31}$$

Usage:

```
myEffects <- addEffect(myEffects, loops_node_covar,
                       node.attribute = "region")
```

3.2 *Name:* **loops by individual covariate**
Description: Are individuals with certain characteristics more likely to remain in their current location? For example, are men more likely to remain in their current organisation, while women are more likely to move employer?
Formula:

$$s(\mathbf{x}) = \sum_{i \in B} w_i I(o_i = d_i) \tag{32}$$

Usage:

```
myEffects <- addEffect(myEffects, loops_resource_covar,
                       edge.attribute = "sex")
```

3.3 *Name:* **loops by location and individual covariate**
Description: This is an interaction of the previous two effects: Do individuals with certain characteristics have a tendency to stay in locations of certain types? Note that this effect should be included alongside the main effects of 'loops by individual covariate' and 'loops by location covariate'.
Formula:

$$s(\mathbf{x}) = \sum_{i \in B} v_{o_i} w_i I(o_i = d_i) \tag{33}$$

Usage:

```
myEffects <- addEffect(myEffects,
                       loops_resource_covar_node_covar,
                       node.attribute = "region",
                       edge.attribute = "sex")
```

3.4 *Name:* **geometrically weighted concentration by dyadic covariate**

Description: Are bandwagon effects (concentration) particularly prevalent between locations that share characteristics as encoded in a binary dyadic covariate? E.g., do workers follow the moves of other workers mainly in case they go to organisations in the same region?

Formula:

$$s(\mathbf{x}) = \sum_{j \neq h \in A} u_{jh} \; g_{cum}(\mathbf{x}_{jh}), \tag{34}$$

with $g_{cum}(y)$ defined as in equation 13. Note that this effect only makes sense for binary relations u.

Usage:

```
myEffects <- addEffect(myEffects,
                       concentration_GW_dyad_covar_bin,
                       node.attribute = "sameRegion")
```

3.5 *Name:* **geometrically weighted concentration by individual-level binary covariate**

Description: Are bandwagon effects (concentration) particularly prevalent between people that are of the same type? E.g., do male workers follow the moves of other male workers additional to following the moves of all workers?

Formula:

$$s(\mathbf{x}) = \sum_{j \neq h \in A} g_{cum}(\mathbf{x}\text{-}\mathbf{w}_{jh}), \tag{35}$$

with $g_{cum}(y)$ defined as in equation 13 and $\mathbf{x}\text{-}\mathbf{w}_{jh}$ refers to the weight of the tie from j to h of individuals of type w. Note that this effect only makes sense for binary relations w.

Usage:

```
myEffects <- addEffect(myEffects,
                       concentration_GW_dyad_covar_bin,
                       node.attribute = "sameRegion")
```

3.6 *Name:* **geometrically weighted reciprocity by binary dyadic covariate**

Description: Is reciprocity in mobility particularly prevalent between locations that share characteristics as encoded in a binary dyadic covariate? E.g., do workers move to organisations in the same region that send more workers to ego's origin?

Formula:

$$s(\mathbf{x}) = \sum_{j \neq h \in A} u_{jh}\ \mathbf{x}_{jh}\ g_{mar}(\mathbf{x}_{hj}), \tag{36}$$

where g_{mar} is defined in equation 21. Note that this effect only makes sense for binary relations u.

Usage:

```
myEffects <- addEffect(myEffects, reciprocity_GW_dyad_covar_bin,
                       node.attribute = "sameRegion")
```

3.7 *Name:* **geometrically weighted reciprocity by dyadic covariate**

Description: Is reciprocity in mobility particularly prevalent between locations in which the dyad is high on a dyadic covariate? E.g., do workers move to organisations of the same size that send more workers to ego's origin?

Formula:

$$s(\mathbf{x}) = \sum_{j \neq h \in A} u_{jh}\ \mathbf{x}_{jh}\ g_{mar}(\mathbf{x}_{hj}), \tag{37}$$

where g_{mar} is defined in equation 21. Note that this effect generalises the above effect to continuous dyadic covariates.

Usage:

```
myEffects <- addEffect(myEffects, reciprocity_GW_dyad_covar,
                       node.attribute = "simSize")
```

3.8 *Name:* **minimum reciprocity by individual covariate**

Description: Do people reciprocate moves to other locations specifically if they and others have a higher value on some covariate? For example, do women move to organisations that send women to their origin organisation?

Formula:

$$s(\mathbf{x}) = \sum_{j < h \in A} \min(\mathbf{z}^w_{jh}, \mathbf{z}^w_{hj}), \tag{38}$$

where z^w represents a weighted matrix representation of the mobility network with each individual contributing to the weight of a tie by the value of covariate w. This makes most sense in case of a binary covariate w.

Usage:

```
myEffects <- addEffect(myEffects,
                       reciprocity_min_resource_covar,
                       edge.attribute = "sex")
```

3.9 *Name:* **loops by inflow of individuals of category w**

Description: Is the tendency to stay in vs. move out of a location dependent on the proportion of individuals of type w that enter the location? This is especially geared towards modelling how some locations become more or less attractive dependent on the change in composition.

Formula:

$$s(\mathbf{x}) = \sum_{i \in B} \frac{\sum_{i' \in B} I(d_{i'} = o_i)\ w_{i'}}{\sum_{i' \in B} I(d_{i'} = o_i)} I(o_i = d_i). \tag{39}$$

Note that this effect is developed for binary covariates w.

Usage:

```
myEffects <- addEffect(myEffects, staying_by_prop_bin_inflow,
                       edge.attribute = "sex")
```

3.10 *Name:* **loops of non-w by inflow of individuals of category w**

Description: Is the tendency to stay in vs. move out of a location of individuals of type *non-w* dependent on the proportion of individuals of type w moving into the location? This is especially geared towards modelling how some locations become more or less attractive dependent on the change in composition for particular groups. This models segregation dynamics.

Formula:

$$s(\mathbf{x}) = \sum_{i \in B} \frac{\sum_{i' \in B} I(d_{i'} = o_i)\ w_{i'}}{\sum_{i' \in B} I(d_{i'} = o_i)} I(o_i = d_i)\ (1 - w_i). \tag{40}$$

Note that this effect is developed for binary covariates w.

Usage:

```
myEffects <- addEffect(myEffects, crowding_out_prop_covar_bin,
                       edge.attribute = "sex")
```

5.2.4 Undocumented effects

There is a series of effects that are implemented but not documented. This is because they are currently under development and not sufficiently understood to recommend their use. For the adventurous researcher, here is a list of the names of these effects that are sometimes degenerate and generally require great care before use.

```
  # effect functions: concentration
concentration_prop
concentration_prop_orig_cov
concentration_rankGW
concentration_norm
concentration_norm_squared

  # effect functions: reciprocity
reciprocity_min_dyad_covar

  # effect functions: transitivity
triad120D
triad120U
triad120C

  # effect functions: endogeneous covariate based
joining_similar_avoiding_dissimilar_covar_bin
joining_similar_avoiding_dissimilar_covar_cont
avoiding_dissimilar_covar_bin
avoiding_dissimilar_covar_cont

  # effect functions: other
in_weights_GW
```

5.3 Modelling considerations

Model specification for mobility network models is not (yet) a very well researched area. Thus, practical experience currently cannot guide model specification for a large number of empirical applications. Nevertheless, there are some considerations that a researcher can take into account.

First, theoretical knowledge about the case to which the model is applied should be the most important determinant in specifying (especially the endogenous part of) the model. Clear theoretical ideas about the mechanisms shaping the mobility network under analysis should give strong indication what effects to include in a model.

Second, it might be desirable to include statistics that are counts of cliques of multiple observations (following the Hammersley-Clifford theorem). This ensures that particular configurations are equally 'preferred' or 'avoided', independent of the rest of the mobility network. Effects that satisfy the Hammersley-Clifford condition include all that end in '_basic' and '_GW'; furthermore the effects 'loops', 'present relations', 'loops by indegree', and 'ìndegree concentration exponent' for $\alpha = 2$ fulfil the Hammersley-Clifford condition. Readers interested in the underlying reasoning

are referred to Besag (1974).

Third, in case direct interpretation of parameters on an individual level is desired, effects that offer clear change statistics might be preferable. These are basic effects, as well as effects that work with 'min'. To a lesser extent, 'GW' effects offer a clear individual-level interpretation.

Fourth, if the modelled data is a sample of a population, effects that are sample size consistent might be preferred, to ensure results extend to the population under analysis.

5.3.1 Modelling column marginalals

In some cases, column marginals of the data are exogenously determined, for example, when organisations have a certain demand for personnel and hire until each position is filled. In these cases, it might not be meaningful trying to model column marginals by 'normal' predictors. In these cases, Block et al. (2022) suggest two options. One is including the log of the empirical column marginals as a 'covariate alter' effect. The second is modelling the size of each destination by a fixed effect.

6 The estimation algorithm

The estimation algorithm for MoNAn makes use of Markov Chain Monte Carlo (MCMC) methods. These are outlined in Block et al. (2022), and are directly adopted from the procedure for ERGMs introduced in Snijders (2002). Readers interested in the method are referred to Snijders (2002), especially Section 7 and the appendix, and the literature cited therein.

The algorithm proceeds in three phases. In each phase likely outcomes of a mobility network under a specified parameter vector θ are simulated. These simulations are used for different purposes in each phase. The simulations use MCMC procedures, in which networks are simulated by changing individual observations one at a time – this is why it is a 'chain', as each simulated outcome differs at most by one observation from the previous simulated outcome. To get independent draws of simulated networks, many simulation steps are necessary to ensure that subsequent draws are not highly correlated.

Details of the estimation algorithm are outlined below. For default values on all options, see the help-files and Section 6.5.

6.1 The 3 phases

In the *first phase* of the estimation, networks are simulated using the initial values of θ. The initial values are specified in the estimation function (Section 7). These simulated networks are used to assess the sensitivity of the statistics $s(\mathbf{x})$ to the parameters θ, which guides the size of the individual updating steps in phase 2.

In the *second phase*, $\hat{\theta}$ is estimated by iterative updating. To that end, one realisation of the mobility network under the current $\hat{\theta}^n$ is simulated and compared to the empirically observed network. The deviation of the different statistics between simulated and observed network, together with the sensitivity calculated in phase 1 and a user-defined step-size, determine the updated next value $\hat{\theta}^{n+1}$. These updating steps are repeated with decreasing step-size, to approximate the maximum likelihood estimate (MLE).

The *third phase* checks whether simulations using the finally obtained value $\hat{\theta}$ from phase 2 reproduce the statistics of the observed network w.r.t. the included effects. This is called a convergence check.

6.2 Burn-in, thinning, and number of iterations

In each phase, the user has the option to specify the burn-in, the thinning, and the number of iterations. In phase 2, the number of iterations varies by sub-phase, as explained in section 6.4.2.

In each phase, the burn-in specifies the number of simulation steps the algorithm makes before taking the first draw of a simulated network. The thinning determines the number of simulation steps between consecutive draws. As a general rule, the

burn-in and thinning should be a multitude of the number of observations, with the burn-in being larger than the thinning.

The number of iterations determines how many draws are taken to execute the respective phase.

In specifying each of the parameters burn-in, thinning, and number of iterations, choosing a value excessively large will mean the estimation takes longer than necessary and consumes more energy. Choosing a value too small leads to different problems depending on the phase.

The smallest damage can be done in *phase 1*. In case burn-in, thinning, or number of iterations is too small, the sensitivity of the statistics to the parameters will be imprecisely estimated. This might result in more steps in phase 2 being necessary. However, note that the sensitivity will be evaluated at the initial values of θ, therefore, these values might not be ideal at later values of θ during phase 2 anyway. Nevertheless, values way too small can result in an R error 'matrix not invertible' that will lead to the estimation being aborted.

Too small values for the number of iterations in *phase 2* will result in the updating of parameters being terminated before the final (converged) value is reached. However, it is not unusual that more than one run of estimations is necessary. Too small values in burn-in and thinning will result in imprecise values in the comparison between simulated and observed networks (especially the thinning is important here). This will happen because the simulation draw will represent a network guided by a mix of the previous and current values of θ, since too few steps mean that the current state does not yet reflect a draw under the current values of θ. In the worst case this can lead to the updating steps 'overshooting' the target of the parameter it is meant to reach, which can lead to the algorithm never converging. If the same values for burn-in and thinning are chosen as in phase 3, the functions `extractTraces` and `autoCorrelationTest` can be used to check whether the thinning and burn-in is adequate (see Sections 8 and 10.1).

Too small values for the number of iterations in *phase 3* might lead to erroneous conclusions regarding the convergence of the model. Too small values for the burn-in can lead to the early simulation draws being influenced by the initial state of the simulations. Too small values for the thinning can lead to a high correlation between subsequent draws that mean they will not be independent anymore. The functions `extractTraces` and `autoCorrelationTest` can be used to check whether the thinning and burn-in is adequate (see Sections 8 and 10.1).

6.3 Choosing the proposal function

In the simulations determining the updating steps, two ways can be chosen. The multinomial proposal function considers more options for the next step and, thus, leads to better 'mixing' of the chain. However, each step requires more calculations and is accordingly slower. The alternative proposal only suggests one option for the next simulated state, which means it is quicker to execute but more simulation steps

are necessary until the correlation between subsequent draws is low enough.

There is currently insufficient experience to make a clear recommendation when to use which option.

6.4 Other options

6.4.1 Allow loops

Is staying in the current location an option for individuals in the data?

6.4.2 Number of sub-phases in phase 2

Phase 2 is executed over a number of sub-phases, where each sub-phase has a smaller gain and more steps. The idea is that towards the end of the algorithm updating steps get more and more precise. The standard value for the number of sub-phases is four. However, if this is not the first run, but only one to improve convergence, only one or two sub-phases might be used, since only small updating steps are desired.

6.4.3 Number of initial iterations in phase 2

This parameter sets the number of iteration in the first sub-phase of phase 2, i.e., the number of updating steps. The number of iterations increases by a factor of 1.75 from sub-phase to sub-phase. Thus, choosing a large value and many sub-phases will result in very long computation times.

6.4.4 Initial gain

In the comparison of simulated to observed networks, the deviations suggest the size of an updating step. This is down weighted by the gain parameter to guard against overshooting in the updates. In each sub-phase, the value of gain is halved to ensure smaller and smaller updating steps when coming closer to the target.

6.4.5 Gain after phase 1

After phase 1, it is possible to make an updating step based on the initial simulations, which should accelerate convergence (slightly). The size of the updating step can be determined here, with values closer to zero being conservative, while values close to one are courageous.

6.5 The R function `monanAlgorithmCreate`

In the creation of the algorithm, all previously mentioned factors can be determined by the user. Most have default values; however, they might not be useful for all applications.

Function name: `monanAlgorithmCreate`

Arguments:

- `state`. *necessary*. A monan state object that contains all relevant information about the outcome in the form of an edgelist, the nodesets, and covariates.
- `effects`. *necessary*. An effect object that specifies the model.
- `multinomialProposal`. *optional*. How should the next possible outcome in the simulation chains be sampled? If TRUE, fewer simulation steps are needed, but each simulation step takes considerably longer. Defaults to FALSE.
- `burnInN1`. *optional*. The number of simulation steps before the first draw in Phase 1. A recommended value is at least n_Individuals * n_locations if multinomialProposal = F, and at least n_Individuals if multinomialProposal = TRUE which is set as default.
- `thinningN1`. *optional*. The number of simulation steps between two draws in Phase 1. A recommended value is at least 0.5 * n_Individuals * n_locations if multinomialProposal = F, and at least n_Individuals if multinomialProposal = TRUE which is set as default.
- `iterationsN1`. *optional*. The number of draws taken in Phase 1. A recommended value is at least 4 * n_effects which is set as default. If the value is too low, there will be an error in Phase 1.
- `gainN1`. *optional*. The size of the updating step after Phase 1. A conservative value is 0, values higher than 0.25 are courageous. Defaults to 0.1.
- `burnInN2`. *optional*. The number of simulation steps before the first draw in Phase 1. A recommended value is at least n_Individuals * n_locations if multinomialProposal = F, and at least n_Individuals if multinomialProposal = TRUE which is set as default.
- `thinningN2`. *optional*. The number of simulation steps between two draws in Phase 2. A recommended value is at least 0.5 * n_Individuals * n_locations if multinomialProposal = F, and at least n_Individuals if multinomialProposal = TRUE which is set as default.
- `initialIterationsN2`. *optional*. The number of draws taken in subphase 1 of Phase 2. For first estimations, a recommended value is around 50 (default to 50). Note that in later sub-phases, the number of iterations increases. If this is a further estimation to improve convergence, higher values (100+) are recommended.
- `nsubN2`. *optional*. Number of sub-phases in Phase 2. In case this is the first estimation, 4 sub-phases are recommended and is set as default. If convergence in a previous estimation was close, then 1-2 sub-phases should be enough.

- `initGain`. *optional*. The magnitude of parameter updates in the first subphase of Phase 2. Values of around 0.6 (default) are recommended.
- `burnInN3`. *optional*. The number of simulation steps before the first draw in Phase 3. A recommended value is at least 3 * n_Individuals * n_locations if multinomialProposal = F, and at least 3 * n_Individuals if multinomialProposal = TRUE which is set as default.
- `thinningN3`. *optional*. The number of simulation steps between two draws in Phase 3. A recommended value is at least n_Individuals * n_locations if multinomialProposal = F, and at least 2 * n_Individuals if multinomialProposal = TRUE which is set as default. In case this value is too low, the outcome might erroneously indicate a lack of convergence.
- `iterationsN3`. *optional*. Number of draws in Phase 3. Recommended are at the very least 500 (default). In case this value is too low, the outcome might erroneously indicate a lack of convergence.
- `allowLoops`. *optional*. Logical: can individuals/resources stay in their origin?

Usage:

```
myAlg <- monanAlgorithmCreate(state = myState,
                              effects = myEffects,
                              multinomialProposal = FALSE)
```

7 Estimating the model

The estimation of MoNAn combines and uses the R objects defined in the previous sections. This section discusses additional options that can be used in the estimation, in particular using non-default starting values.

7.1 Pre-estimation

MoNAn uses MCMC methods for parameter estimation. Standard statistical estimation methods cannot be used, because assumptions of independent observations are systematically violated (unless no endogenous effects are included). However, estimation of the model under the (wrong) assumption of independent observations usually provides parameters that are closer to the MLE than naïve guesses. Therefore, estimating a multinomial logit model with MoNAn effects as model parameters can give good starting values for a proper MoNAn estimation, which considerably improves chances of model convergence after the first run. We refer to parameters estimated under the assumption of independent observations as the pseudo-MLE estimates (Frank and Strauss, 1986).

To obtain such estimates, statistics for a multinomial logit model needs to be calculated from the data using the following function. One example of an estimation routine is provided below.

Function name: `getMultinomialStatistics`

Arguments:

- `state`. *necessary.* A processState.monan object that stores all information to be used in the model.
- `effects`. *necessary.* An effects object for which the statistics of a multinomial model should be calculated.

Usage:

```
myStatisticsFrame <- getMultinomialStatistics(
     state = myState,
     effects = myEffects)
```

Additional script to get pseudo-likelihood estimates, requiring the dfidx and mlogit package:

```
library(dfidx)
library(mlogit)
my.mlogit.dataframe <- dfidx(myStatisticsFrame,
                             shape = "long",
                             choice = "choice")
```

```
colnames(my.mlogit.dataframe) <-
     gsub(" ", "_", colnames(my.mlogit.dataframe))

IVs <- (colnames(my.mlogit.dataframe)[2:(ncol(myStatisticsFrame)-2)])

f <- as.formula(paste("choice ~ 1 + ",
                paste(IVs, collapse = " + "), "| 0"))

my.mlogit.results <- mlogit(formula = eval(f),
                            data = my.mlogit.dataframe,
                            heterosc = F)

summary(my.mlogit.results)

initEst <- my.mlogit.results$coefficients[1:length(IVs)]
```

7.2 The estimation function

The central step in a MoNAn analysis is the estimation. Here, the information from the data (state), the model specification (effects), and the algorithm is used to estimate parameters that represent the observed mobility network. Since most important modelling decisions have been taken in the previous steps, running an estimation is rather straightforward. Estimation options (parallel computing, returning simulated networks, and starting values) are discussed in Section 7.4.

Function name: `monanEstimate`

Arguments:

- `ans`. *necessary.* The results object resulting from an estimation with "monanEstimate"
- `state`. *necessary.* A monan state object that contains all relevant information about the outcome in the form of an edgelist, the nodesets, and covariates.
- `effects`. *necessary.* An effect object that specifies the model.
- `algorithm`. *necessary.* An object that specifies the algorithm used in the estimation.
- `initialParameters`. *optional.* Starting values for the parameters. Using starting values, e.g., from a multinomial logit model (see getMultinomialStatistics()) increases the chances of model convergence in the first run of the estimation considerably.

- `prevAns`. *optional*. If a previous estimation did not yield satisfactory convergence, the outcome object of that estimation should be specified here to provide new starting values for the estimation.
- `parallel`. *optional*. Computation on multiple cores?
- `cpus`. *optional*. Number of cores for computation in case parallel = TRUE.
- `verbose`. *optional*. Logical: display information about estimation progress in the console?
- `returnDeps`. *optional*. Logical: should the simulated values of Phase 3 be stored and returned? This is necessary to run GoF tests. Note that this might result in very large objects.

Usage:

```
myResDN <- monanEstimate(
  state = myState,
  effects = myEffects,
  algorithm = myAlg,
  parallel = TRUE, cpus = 4,
  verbose = TRUE,
  returnDeps = TRUE
)

# In case pseudo-likelihood estimates have been
# obtained previously, this can be specified by
myResDN <- monanEstimate(
  state = myState,
  effects = myEffects,
  algorithm = myAlg,
  initialParameters = initEst,
  parallel = TRUE, cpus = 4,
  verbose = TRUE,
  returnDeps = TRUE
)

# If this is not the first estimation run of the model
# previous results can be included by
myResDN <- monanEstimate(
  state = myState,
  effects = myEffects,
  algorithm = myAlg,
  prevAns = myResDN,
  parallel = TRUE, cpus = 4,
```

```
  verbose = TRUE,
  returnDeps = TRUE
)
```

7.3 Checking model convergence

A crucial part after running an estimation is to check whether the model has converged. Model convergence means that the simulations using the finally obtained value $\hat{\theta}$ from phase 2 reproduce the statistics of the observed network w.r.t. the included effects. These checks are carried out in phase 3 of the estimation.

Convergence statistics are calculated by taking the difference between the statistics underlying each effect for the observed network, and for the average of all simulated networks in phase 3. This difference is normalised by the standard deviation of the statistic values over the simulations (effect-by-effect), see Snijders (2002). Small values in the convergence statistics mean that the average simulated network is very close in terms of the effect statistics to the observed network. Thus, small values for all statistics mean that the model has converged and the parameters represent the modelled data.

Ideally, all convergence statistics should be below an absolute value of 0.1 (that is, on average the simulations should be less than 0.1 sd's away from the observation in terms of all statistics). In practise, the final column when printing a results object shows the convergence statistics for the respective effects. Finding the highest (worst) absolute convergence value can be done by:

```
max(abs(myResDN$convergenceStatistics))
```

If this value is above 0.1, the model has not converged and the parameters are not reliable. Further estimation runs are necessary before the parameters can be interpreted in a valid way (see below and Section 10.1)

7.3.1 What if my model did not converge?

There are many reasons why a model can suffer from convergence problems, including the model specification or choice of parameters in the model algorithm. In Section 10.1 the most common problems concerning model convergence are discussed alongside potential solutions.

7.4 Estimation options

7.4.1 Initial parameters and prevAns

Since the estimation algorithm iteratively updates the parameters to get closer and closer to the MLE, the better the initial values are, the quicker the MLE is reached. Choosing good starting values can have a substantial impact on the time it takes and the number of estimation runs needed until the model is converged. Without

specifying any starting values, the estimation algorithm starts at setting all initial parameters to zero.

Initial parameters that are estimated using (much faster) pseudo-MLE procedures often get close to the MLE. How they are obtained is discussed in Section 7.1. In the `monanEstimate` function they are included with the `initialParameters` option.

In case a model was already estimated but did not reach convergence, the results object from the previous estimation run can be used as new starting values. In the `monanEstimate` function results from a previous estimation are included with the `prevAns` option. Another advantage of using the `prevAns` option is that the calculations necessary for phase 1 can be extracted from the results object, further speeding up the estimation.

7.4.2 returnDeps

During phase 3 of the estimation, a number of mobility networks under the final parameters from the estimation are simulated. How many networks are generated is determined by the `iterationsN3` option in `monanAlgorithmCreate`. By default, not the entire networks are stored but only relevant information that is used to calculate convergence. This is to avoid generating overly large results objects.

However, the complete networks resulting from each simulation can be stored in the results object, setting `returnDeps = TRUE`. Storing the simulation outcomes is necessary in case a users wants to do some further checking of the algorithm, score tests, or goodness of fit tests (see Section 8). Each of these tests uses simulated networks under the model. Thus, for more advanced use of MoNAn returning the simulated networks might be the common option.

7.4.3 Parallel computing

All phases can be executed distributed over multiple cores. Especially for phase 3, this reduces the time proportional by the number of cores. In phase 2, multiple sub-phases are run in parallel and resulting parameter values are averaged at the end. There is a limit to how much this speeds up reaching convergence, but there is insufficient experience to give clear guidance on the number of cores. However, for more cores the initial gain (size of the updating steps after each draw) can be chosen higher, which should speed up convergence.

8 Tests and diagnostics

After a completed estimation, a set of diagnostics is available to check the estimated model. These concern checking the algorithm, the model specification regarding the inclusion of additional effects, and overall goodness of fit concerning auxiliary statistics.

8.1 Checking the algorithm

The primary metric whether the parameters from a model estimation are reliable are the convergence statistics (see Section 7.3). However, there are additional tools that give insight into the performance of the estimation algorithm. These insights can be used to understand why model convergence might be problematic (see Section 10.1 for guidance what to do), as well as further assurance of proper behaviour of the algorithm beyond convergence statistics.

Two functions are available to check the modelling algorithm. The first assesses the distance between two subsequent draws of simulations in phase 3, i.e., the autocorrelation. The function simply returns the average proportion of individuals that are in the same location in two consecutive draws of simulated networks. Generally, lower values are better, while values of above 0.4 can be problematic.

Assessing the autocorrelation uses the simulations of the network under the estimated model that are generated in phase 3 of the estimation algorithm. As such, the autocorrelation test function only works if `deps = TRUE` (store networks generated in phase 3) is specified for the estimation.

Function name: `autoCorrelationTest`

Arguments:

- `ans`. *necessary.* The results object resulting from an estimation with "monanEstimate"

Usage:

```
autoCorrelationTest(ans = myResDN)
```

The second function to assess the modelling algorithm looks at the values of the statistics underlying each effect over the course of the simulations in phase 3. To this end, the statistic value for each included effect (e.g., same_covariate_region, or reciprocity_min) is calculated for each simulated network. In the ideal case, the values are randomly scattered around the target value (i.e., the value of the observed network) in a normally looking fashion. If no trends over the course of all simulations are discernible, and there is no obvious relation between the values of consecutive simulation draws, the algorithm 'mixes' well.

The extraction of traces uses the simulations of the network under the estimated model that are generated in phase 3 of the estimation algorithm. As such, the extract traces function only works if `deps = TRUE` (store networks generated in phase 3) is specified for the estimation.

Function name: `extractTraces`

Arguments:

- `ans`. *necessary.* The results object resulting from an estimation with "monanEstimate"
- `effects`. *necessary.* The effects object used in the estimation

Usage:

```
traces <- extractTraces(ans = myResDN,
                        effects = myEffects)
plot(traces)
```

8.2 Checking the specification: score-type tests

Given the large computational burden of estimations in MoNAn, it is not practical to engage in many standard model selection practises. One alternative for forward model selection is using score-type tests.

In score-type tests we define a candidate effect that a user considers to additionally include in the model after an estimation. Score-type tests are used to check whether the network-level statistics of a candidate effect is represented well by the model even without the effect being explicitly modelled. If the specification of an estimated model can reproduce the statistic of a candidate effect, this is a strong indication that including the effect would not be significant. On the other hand, if the simulated networks differ substantially from observation regarding the statistics of the candidate effect, it suggests that including this effect would result in a significant parameter estimate.

Score-type tests use the simulations of the network under the estimated model that are generated in phase 3 of the estimation algorithm. As such, the score-type test function only works if `deps = TRUE` (store networks generated in phase 3) is specified for the estimation.

Function name: `scoreTest`

Arguments:

- `ans`. *necessary.* The results object resulting from an estimation with "monanEstimate"
- `effects`. *necessary.* An effects object in which the non included effects that should be tested are specified

Usage:

```
myEffects2 <- createEffects(myState) |>
  addEffect(transitivity_min)

test_ME.2 <- scoreTest(ans = myResDN,
                       effects = myEffects2)
test_ME.2
```

8.3 Goodness of fit

Goodness of fit (gof) testing in network models in its most usual form checks whether a network model can reproduce un-modelled network characteristics (Hunter et al., 2008), often on a more aggregate (network) level (Snijders and Steglich, 2015). Such characteristics include degree distributions, triad census, community structure, path lengths etc.

This type of gof-testing is equally available in MoNAn. Gof tests use the simulations of the network under the estimated model that are generated in phase 3 of the estimation algorithm. As such, the gof function only works if `deps = TRUE` (store networks generated in phase 3) is specified for the estimation.

Function name: `monanGOF`

Arguments:

- `ans`. *necessary.* The results object resulting from an estimation with "monanEstimate"
- `gofFunction`. *necessary.* A gof function that specifies which auxiliary outcome should be, e.g., "getIndegree" or "getTieWeights"
- `lvls`. *necessary.* The values for which the gofFunction should be calculated / plotted

Usage:

```
myGofIndegree <- monanGOF(ans = myResDN,
                                    gofFunction = getIndegree,
                                    lvls = 1:100)

plot(myGofIndegree,  lvls = 20:70)
```

8.3.1 Implemented auxiliary functions

Currently only two of-the-shelf functions for gof testing are implemented in MoNAn. These are:

Function name: `getIndegree`
Description: Calculates the weighted indegree distribution of all locations in the network. The weighted indegree is simply the column sum of the mobility table.

Function name: `getTieWeights`
Description: Extracts the distribution of tie weights in the mobility network.

However, programming of auxiliary functions for MoNAn is fairly straightforward for experienced R users. The major obstacle is understanding where the relevant information is stored in the state (raw data) and internal cache (transformed networks). From that point, standard network functions implemented in R can be used. Gof auxiliary functions implemented in R have the following form:

```
gofFunction <- function(state,
                        dep.var, lvls){
                          # some function of state
                          # e.g. extracting the degree
                          # distribution
                        }
```

9 Simulation

For any given parameter θ, two defined nodesets, and potentially covariates, it is possible to simulate mobility networks. The primary use of simulated data lies in goodness-of-fit or score-type testing; however, this is usually done using simulations from phase 3 of an estimation with parameter values determined by an estimation.

However, there are further uses to simulating data with using values of θ that are set by a researcher. One use is the extrapolation of macro-level consequences depending on micro-level changes in parameters. The simplest application of this idea is setting the parameter related to one effect to zero and examining the changes in macro-level outcomes. This aims to answer how much a network-level characteristic (e.g., segregation) is influenced by micro-level tendencies of individual action (e.g., avoidance of the same location as dissimilar others). One example of this application is Block (2023). Other motivations to simulate data (among many) are to improve the understanding of model behaviour, building toy models for theory development, or developing new effects.

The specification of a simulation depends on the same factors as the simulations in the estimation. Thus, for a deeper introduction on how to specify the simulation parameters `allowLoops`, `burnin`, `thinning`, and `nSimulations`, see Section 6.

Function name: `monanSimulate`

Arguments:

- `state`. *necessary.* A monan state object that contains all relevant information about nodesets, and covariates. Further, an edgelist of the dependent variable needs to be specified with the initial mobility network as starting value for the simulation. For a large enough burnin, any initial mobility network is allowed.
- `effects`. *necessary.* An effects object that specifies the model.
- `parameters`. *necessary.* The parameters associated with the effects that shall be used in the simulations.
- `allowLoops`. *necessary.* Logical: can individuals/resources stay in their origin?
- `burnin`. *necessary.* The number of simulation steps that are taken before the first draw of a network is taken. A number too small will mean the first draw is influenced by the initially specified network. A recommended value for the lower bound is 3 * n_Individuals * n_locations.
- `thinning`. *necessary.* The number of simulation steps that are taken between two draws of a network. A recommended value for the lower bound is n_Individuals * n_locations.
- `nSimulations`. *necessary.* The number of mobility networks to be simulated.

Usage:

```
mySimDN <- monanSimulate(state = myState,
                         effects = myEffects,
                         parameters = c(2, 1, 1.5,
                                        0.1, -1, -0.5),
                         allowLoops = TRUE,
                         burnin = 45000,
                         thinning = 15000,
                         nSimulations = 10
)
```

10 Problems and errors

10.1 Model does not converge

One of the most common problems for this kind of model is that it might be difficult to reach convergence, especially for endogenous parameters. This might have different reasons and requires different solutions.

The easiest case is that there are simply not enough iterations in phase 2 in the first estimation run. The solution is to re-run the estimation with the results of the first estimation as starting values (see Section 7.4.1). Second (or later) estimations can use fewer sub-phases and adjust the gain parameter (see 6). As long as the convergence improves with every run, re-running the model with improved starting values is the best strategy to work towards a converged model. If this does not work, different issues might stop the model from converging.

In case the convergence statistics are very large (> 3) and do not decrease rapidly in subsequent estimations (or even increase), this points to a problem in the model specification. For example, using endogenous statistics that are prone to near-degeneracy will lead to this problem. Candidate effects that are known to lead to near-degeneracy in some cases include 'concentration_basic', 'reciprocity_basic', 'transitivity_basic', and 'in_weights_exponent'. In these cases, adjusting the model specification to use '_GW' effects or effects based on proportions should help.

A second specification problem that can lead to convergence problems of the large kind are highly co-linear effects. When two effects model (almost) the same pattern, this can become apparent in problems reaching convergence. Co-linearity of effects can be checked using the covariance matrix as stored in a monan.results object under `myResDN$covarianceMatrix`.

In case convergence is small (< 1), but not small enough (> 0.1), a few problems might be present. First, phase 3 might not have enough iterations or a too small burn-in or thinning. The functions `extractTraces` and `autoCorrelationTest` can be used to check whether the thinning and burn-in is adequate. The graphs produced by `extractTraces` should show points normally scattered around a constant value on the y-axis. A trend at the beginning of the plots indicates too low burn-in, trends over the entire course of the plot indicate too low thinning. A different way to check for this is `autoCorrelationTest`. The result should be low, preferably below 0.3 (but this is just a rule of thumb). For values above 0.5, the user should consider increasing the thinning parameter. If thinning, burn-in, and the number of iterations in phase 3 are large enough, the problem lies in phase 2.

In case parameters between subsequent runs of the model change very little, increasing the initial gain can help to allow the algorithm to make larger updating steps. Conversely, if parameters change a lot but convergence does not improve, decreasing the gain can help. It might also be that the burn-in and thinning values are too low (see Section 6.2). To check whether this is the case, the user can adjust these values to match the burn-in and thinning parameters in phase 3 and see whether they fulfil the criteria outlined in the previous paragraph.

10.2 Non-invertible matrix

The error message referencing a non-invertible matrix usually happens at the end of phase 1 or phase 3. This can be the case if the number of iterations in phase 1 or phase 3 is too low. This can lead to the inversion of the covariance matrix not being possible and more iterations are needed. The second possible cause can be highly co-linear effects. When two effects model (almost) the same pattern, this can become apparent in problems inverting the covariance matrix. Checking covariance between effects can be done by looking at the covariance matrix at `resultsObject$covarianceMatrix`. This requires updating the model specification to exclude one of the highly co-linear effects.

10.3 Other errors

Many other errors can come from mistakes in specifying the data, or state. They might only become apparent when running the estimation, since the estimation function might try to access data that does not exist. Carefully going through the preparation of the data and making sure no typos are present might be the best option to solve these problems.

For errors that cannot be dealt with this way, contacting the package maintainer is a last option.

A Bibliography

Besag, J. (1974). Spatial interaction and the statistical analysis of lattice systems. *Journal of the Royal Statistical Society: Series B (Methodological)*, 36(2):192–225.

Block, P. (2023). Understanding the self-organization of occupational sex segregation with mobility networks. *Social Networks*, 73.

Block, P., Stadtfeld, C., and Robins, G. (2022). A statistical model for the analysis of mobility tables as weighted networks with an application to faculty hiring networks. *Social Networks*, 68:264–278.

Frank, O. and Strauss, D. (1986). Markov graphs. *Journal of the American Statistical Association*, 81:832–842.

Hunter, D. R., Goodreau, S. M., and Handcock, M. S. (2008). Goodness of fit of social network models. *Journal of the American Statistical Association*, 103:248–258.

Lusher, D., Koskinen, J., and Robins, G. (2013). *Exponential Random Graph Models for Social Networks: Theories, Methods and Applications*. Cambridge University Press.

R Core Team (2023). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

RStudio Team (2020). *RStudio: Integrated Development Environment for R*. RStudio, PBC., Boston, MA.

Snijders, T. A. and Steglich, C. E. (2015). Representing micro–macro linkages by actor-based dynamic network models. *Sociological Methods and Research*, 44:222–271.

Snijders, T. A. B. (2002). Markov chain monte carlo estimation of exponential random graph models. *Journal of Social Structure*, 3:1–40.

Snijders, T. A. B., Pattison, P., Robins, G., and Handcock, M. S. (2006). New specifications for exponential random graph models. *Sociological Methodology*, 36:99–153.

B Index of R functions

C Changes compared to earlier versions

Version 1.0.0, 13 Apr 2024

- The cache is now automatically created and hidden from the user.
- Functions names of all core functions are streamlined and simplified, often in wrapper functions.
- Effects are now created in two successive steps, by createEffects and addEffects.
- Parallelisation of Phase 1.
- If returnDeps = TRUE in the estimation, the returned simulations are much slimmer, only containing the simulated edgelist.
- New print functions for state, and effects.
- Various new effects.
- Release of new package version to CRAN.

Version 0.1.3, 05 Feb 2024

- The "dependentVariable" is now automatically detected and does not need to be specified by the user in all functions.
- Further tests of compatibility and improved error messages in the process of creating the data.
- Various new effects.
- Implemented framework to test new effects.
- Release of new package version to CRAN.

Version 0.1.2, 30 Aug 2023

- Update of the documentation and manual.
- Renaming of `gofDistributionNetwork` to `gofMobilityNetwork`.
- Release of the package to CRAN

Version 0.1.1, 15 Aug 2023

- Update of the documentation.

Version 0.1.0, 02 Aug 2023

- First release.