Manual for RSiena

Ruth M. Ripley, Tom A.B. Snijders
Zsófia Boda, András Vörös

University of Oxford: Department of Statistics; Nuffield College
University of Groningen: Department of Sociology

October 25, 2026

Abstract

SIENA (for Simulation Investigation for Empirical Network Analysis) is a computer program that carries out the statistical estimation of models for the evolution of social networks according to the dynamic actor-oriented model of which an overview is in Snijders (2017), and also the random coefficient multilevel network model of Koskinen and Snijders (2023). This is the manual for RSiena, a contributed package to the statistical system R. It complements, but does not replace the help pages for the RSiena functions! It also contains contributions written by Mark Huisman, Michael Schweinberger, Christian Steglich, and Viviana Amati.

This manual is frequently updated, mostly only in a minor way. This version was renewed for RSiena version 1.6.14.

In version 1.6, the names of RSiena functions have undergone important changes. This manual is for the new names.

Contents

1 General information **7**
- 1.1 Acknowledgements . . . 8
- 1.2 Giving references to RSiena . . . 9

2 Getting started with SIENA **10**
- 2.1 The logic of Stochastic Actor-Oriented Models . . . 10
  - 2.1.1 Types of Stochastic Actor-Oriented Models . . . 11
  - 2.1.2 Data, variables and effects . . . 12
  - 2.1.3 Outline of estimation procedure . . . 15
  - 2.1.4 Design issues for longitudinal network modeling . . . 16
  - 2.1.5 Identifiability . . . 16
  - 2.1.6 Further useful options in RSiena . . . 17
- 2.2 Installing R and RSiena . . . 17
- 2.3 Using RSiena within R . . . 18
- 2.4 Example R scripts for getting started . . . 19
- 2.5 Steps for looking at results: Executing SIENA. . . . 20
- 2.6 Getting help with problems . . . 21

3 New function names from Version 1.6 **23**
- 3.1 Function transformScript . . . 26

4 Steps of modelling **28**

5 Input data **29**
- 5.1 Data types . . . 29
  - 5.1.1 Network data . . . 29
  - 5.1.2 Transformation between matrix and edge list formats . . . 30
  - 5.1.3 Behavioral data . . . 32
  - 5.1.4 Continuous behavioral data . . . 32
  - 5.1.5 Individual covariates . . . 32
  - 5.1.6 Dyadic covariates . . . 33
- 5.2 Internal data treatment . . . 34
  - 5.2.1 Interactions and dyadic transformations of covariates . . . 34
  - 5.2.2 Centering . . . 34
  - 5.2.3 Centering of behavioral variables . . . 36
  - 5.2.4 Monotonic dependent variables . . . 37
  - 5.2.5 Multivariate dependent variables with constraints . . . 37
- 5.3 Further data specification options . . . 37
  - 5.3.1 Structurally determined values . . . 37
  - 5.3.2 Missing data . . . 39
  - 5.3.3 Composition change: joiners and leavers . . . 41

6 Model specification **43**
- 6.1 Definition of the model . . . 43
  - 6.1.1 Elementary effects . . . 45
  - 6.1.2 Specification in RSiena . . . 45
  - 6.1.3 Mathematical specification . . . 46

6.2 Important structural effects for network dynamics: one-mode networks . . . 48
6.2.1 Rate function . . . 52
6.3 Important structural effects for network dynamics: two-mode networks . . . 52
6.4 Effects for network dynamics associated with covariates . . . 53
6.5 Cross-network effects for dynamics of multiple networks . . . 54
6.6 Hierarchy requirements . . . 55
6.7 Constraints between networks . . . 57
6.8 Effects on behavior evolution . . . 57
6.9 Model Type: non-directed networks . . . 59
6.10 Model Type: directed networks . . . 60
6.11 Model Type: behavior . . . 60
6.12 Interaction effects . . . 61
6.12.1 Internal effect parameters for interaction effects . . . 62
6.12.2 The `interactionType` of interaction effects . . . 63
6.12.3 Interaction effects for network dynamics . . . 63
6.12.4 Interaction effects for behavior dynamics . . . 64
6.13 Time heterogeneity in model parameters . . . 65
6.14 Limiting the maximum outdegree . . . 65
6.15 Goodness of fit with auxiliary statistics: test_gof . . . 67
6.15.1 Auxiliary statistics . . . 67
6.15.2 Plots . . . 69
6.15.3 p-values . . . 69
6.15.4 Composition change, missing data, and structural values in test_gof . . . 70

7 Estimation **71**
7.1 An overview of the estimation procedure . . . 71
7.2 Arguments of the algorithm . . . 72
7.3 Results of the estimation . . . 74
7.3.1 Initial Values . . . 75
7.3.2 Convergence Check . . . 76
7.3.3 Continued estimation to obtain convergence . . . 77
7.4 Use of the algorithm arguments . . . 78
7.5 What to do if there are convergence problems . . . 79
7.6 Some important components of the sienaFit object . . . 81
7.7 Algorithm . . . 83
7.8 Output . . . 84
7.8.1 Convergence check . . . 85
7.8.2 Parameter values and standard errors . . . 85
7.8.3 Collinearity check . . . 86
7.9 Other estimation procedures . . . 87
7.10 Generalized Method of Moments estimation . . . 88
7.10.1 Using the function siena . . . 88
7.11 Maximum Likelihood and Bayesian estimation . . . 92
7.11.1 Treatment of missing data . . . 94
7.12 Other remarks about the estimation algorithm . . . 95
7.12.1 Conditional and unconditional estimation . . . 95
7.12.2 Fixing parameters . . . 95

7.12.3 Automatic fixing of parameters . . . 96
7.12.4 Required changes from conditional to unconditional estimation . . . 96
7.13 Using multiple processes . . . 96

8 Standard errors **98**
8.1 Multicollinearity . . . 98
8.2 Precision of the finite differences method . . . 99

9 Tests **100**
9.1 Wald-type tests . . . 100
9.1.1 Multi-parameter Wald tests . . . 101
9.1.2 Tests of linear combinations . . . 103
9.1.3 Tests of equality of parameters . . . 104
9.2 Score-type tests and one-step estimates . . . 105
9.2.1 One-step estimates . . . 106
9.3 Alternative application: convergence problems . . . 106
9.4 Testing differences between independent groups or periods . . . 107
9.5 Testing time heterogeneity in parameters . . . 108

10 Simulation **110**
10.1 Accessing the generated networks . . . 111
10.2 Conditional and unconditional simulation . . . 113
10.3 Simulating with variable parameter values . . . 113

11 Getting started **114**
11.1 Model choice . . . 115

12 Multilevel network analysis **116**
12.1 Function make_group_rsiena . . . 118
12.1.1 Group-level variables . . . 118
12.2 Multi-group Siena analysis . . . 118
12.3 Meta-analysis of Siena results . . . 119
12.3.1 Meta-analysis directed at the mean and variance of the parameters . . . 120
12.3.2 Meta-analysis directed at testing the parameters . . . 122
12.3.3 Contrast between the two kinds of meta-analysis . . . 123
12.4 Random coefficient multilevel Siena analysis . . . 123
12.4.1 Treatment of missing data . . . 124
12.4.2 Which data sets to use for multi_siena . . . 125
12.4.3 Model specification . . . 125
12.4.4 How to enter your data in multi_siena . . . 127
12.4.5 How to choose the parameter settings for multi_siena . . . 127
12.4.6 Prior distributions . . . 128
12.4.7 Operation of multi_siena . . . 131
12.4.8 Assessing convergence . . . 131
12.4.9 Interpreting results of multi_siena . . . 132

13 Formulas for effects **134**
13.0.1 Internal effect parameters . . . 134
13.1 Network evolution . . . 135
13.1.1 Network evaluation function . . . 135
13.1.1.1 Structural effects . . . 135
13.1.1.2 Dyadic covariate effects . . . 150
13.1.1.3 Dyadic double covariate effects . . . 152
13.1.1.4 Monadic covariate effects . . . 153
13.1.1.5 Monadic double covariate effects . . . 164
13.1.2 Special effects for non-directed networks . . . 166
13.1.3 Multiple network effects . . . 167
13.1.3.1 Dyadic covariate effects . . . 179
13.1.3.2 Monadic covariate effects . . . 181
13.1.3.3 Effects for three networks . . . 190
13.1.4 Network creation and endowment functions . . . 191
13.1.5 Network rate function . . . 192
13.2 Behavioral evolution . . . 194
13.2.1 Behavioral evaluation function . . . 194
13.2.1.1 Effects of multiple networks . . . 204
13.2.1.2 Monadic covariate effects . . . 205
13.2.1.3 Dyadic covariate effects . . . 211
13.2.1.4 User-defined interaction effects . . . 213
13.2.2 Behavioral creation function . . . 213
13.2.3 Behavioral endowment function . . . 214
13.2.4 Behavioral rate function . . . 214
13.3 Effects for estimation by Generalized Method of Moments . . . 217

14 Parameter interpretation **218**
14.1 Networks . . . 218
14.2 Behavior . . . 219
14.3 Ego – alter selection tables . . . 220
14.4 Ego – alter influence tables . . . 226
14.4.1 Average alter . . . 226
14.4.2 Average similarity . . . 228
14.5 Effect sizes and measures of fit . . . 232
14.5.1 Entropy / relative certainty . . . 232
14.5.2 Standard deviation of change statistics . . . 233
14.5.3 Relative importance of effects . . . 233

15 Error messages **234**
15.1 After updating . . . 234
15.2 During estimation . . . 235
15.3 As the result of a score-type test (including time test) . . . 237
15.4 In test_gof . . . 237

16 For programmers: Get the source code **239**

17 For programmers: Other tools you need **239**

18 For programmers: Building, installing and checking the package **239**

19 For programmers: Understanding and adding an effect **240**
19.1 The use of the hash (#) sign
in names of effects and estimation statistics 242
19.2 Example: adding the truncated out-degree effect 243
19.3 Notes on effectGroups and two-mode networks 247

A Appendix: Changes compared to earlier versions **250**

B References **304**

1 General information

SIENA[1], shorthand for Simulation Investigation for Empirical Network Analysis, is a set of methods implemented in a computer program that carries out the statistical estimation of models for repeated measures of social networks according to the Stochastic Actor-oriented Model ('SAOM') of Snijders and van Duijn (1997), Snijders (2001), Snijders et al. (2007), Snijders et al. (2010a), Snijders et al. (2013), Greenan (2015a), and Koskinen and Snijders (2023); also see Steglich et al. (2010). A tutorial for these models is in Snijders et al. (2010b). A review article is Snijders (2017). Books are in preparation (Snijders and Steglich, 2026, 2023).

A website for SIENA is maintained at `https://www.stats.ox.ac.uk/~snijders/siena/` . At this website ('publications' tab) you shall also find references to introductions in various other languages, as well as the file `Siena_algorithms.pdf` which gives a sketch of the main algorithms used in RSiena. The website further contains references to many published examples, example scripts illustrating various possibilities of the package, course announcements, etc.

RSiena is a contributed package for the R statistical system (R Core Team, 2025) and can be installed in the usual way from CRAN; see `http://cran.r-project.org`. Development versions of RSiena are at `https://github.com/stocnet/rsiena/`. Here also the most recent version is available at `https://github.com/stocnet/rsiena/releases`; it is recommended to use this version.

This is a manual for RSiena; the manual is provisional in the sense that it is continually being updated, taking account of updates in the package. For the operation of R, the reader is referred to this manual, the help pages, the website, and the corresponding literature.

For working with RSiena, please use jointly the following information:

1. The help pages for the functions in RSiena, which can be consulted in the usual R way by entering ? followed by the function name;
   note that by clicking on the version mentioned at the bottom of each help page, you are brought to a directory of all help pages for the package;

2. this manual (of which the web version is updated quite frequently);

3. the website `http://www.stats.ox.ac.uk/~snijders/siena/` , e.g., its `News` page and its `RSiena scripts` page;

4. the users' group for SIENA meant to exchange information and technical advice; the address is `https://groups.io/g/RSiena` .

RSiena was originally programmed by Ruth Ripley and Krists Boitmanis, in collaboration with Tom Snijders. Starting in May 2012, maintainer of the software was Tom Snijders. Major contributions were made by Josh Lospinoso, Charlotte Greenan, Michael Schweinberger, Felix Schönenberger, Johan Koskinen, Mark Ortmann, Nynke Niezink, Viviana Amati, and Mark Huisman. Other contributors include Christian Steglich, Christoph Stadtfeld, Per Block, Robert Krause, Marion Hoffman, Alvaro Uzaheta, Robert Hellpap, James Hollway, and Steffen Triebel.

[1] This program was first presented at the International Conference for Computer Simulation and the Social Sciences, Cortona (Italy), September 1997, which originally was scheduled to be held in Siena. See Snijders and van Duijn (1997).

Currently, RSiena is maintained by a research group at the University of Groningen, seconded by researchers in Zürich, Stockholm, Geneva, and elsewhere. Since July 2025, Christian Steglich is responsible for package maintenance on CRAN.

1.1 Acknowledgements

The development of RSiena was carried out by researchers of the Universities of Oxford, Groningen, Konstanz, Zürich, Manchester, and Melbourne. We are grateful to NIH (National Institutes of Health, USA) for their funding of programming RSiena during 2008-2014. This was done as part of the project *Adolescent Peer Social Network Dynamics and Problem Behavior*, funded by NIH (Grant Number 1R01HD052887-01A2), Principal Investigator John M. Light (Oregon Research Institute).

For earlier work on SIENA, we are grateful to NWO (Netherlands Organisation for Scientific Research) for their support to the project *Models for the Evolution of Networks and Behavior* (project number 461-05-690), the integrated research program *The dynamics of networks and behavior* (project number 401-01-550), the project *Statistical methods for the joint development of individual behavior and peer networks* (project number 575-28-012), the project *An open software system for the statistical analysis of social networks* (project number 405-20-20), and to the foundation ProGAMMA, which all contributed to the work on SIENA.

1.2 Giving references to RSiena

When using SIENA, it is appreciated that you refer to the package as such or this manual; and to one or more relevant references of the methods implemented in the program. The reference to this manual and to the package can be obtained in R from

```
citation("RSiena")
```

For the methods implemented in RSiena, a standard general reference is Snijders (2017). Tutorials are Snijders et al. (2010b) and Kalish (2020). A basic reference for the network dynamics model is Snijders (2001) or Snijders (2005). Basic references for the model of network-behavior co-evolution are Snijders et al. (2007) and Steglich et al. (2010). Review articles are Snijders (2017) and Snijders and Pickup (2017). The latter is also the reference for the model for dynamics of non-directed networks.

More specific references are Koskinen and Edling (2012) for the dynamics of two-mode networks and Snijders et al. (2013) for the co-evolution of multiple networks – two mode or one-mode or both. For the model for diffusion of innovations in dynamic networks, please refer to Greenan (2015a). More technical references are Schweinberger (2012) for the score-type tests and Schweinberger and Snijders (2007b) for the calculation of standard errors of the Method of Moments estimators. For assessing and correcting time heterogeneity, refer to Lospinoso et al. (2011). For goodness of fit assessment and associated model selection considerations, refer to Lospinoso and Snijders (2019). A basic reference for the Bayesian estimation is Koskinen and Snijders (2007) and for the maximum likelihood estimation Snijders et al. (2010a). For Generalized Method of Moments estimators, references are Amati et al. (2015, 2019). For the integrated multilevel network analysis implemented in function multi_siena, the reference is Koskinen and Snijders (2023).

A general reference for R is R Core Team (2025).

2 Getting started with SIENA

There may be various strategies for getting acquainted with RSiena. In any case, it is a good idea to study the tutorial Snijders et al. (2010b). Two recommended options for learning the 'how to' are the following:

1. One excellent option is to read the User's Manual from start to finish (leaving aside the Programmer's Manual).

2. A second option is to read this Minimal Introduction, to get a sense of the rest by looking at the table of contents, and then follow the references to specific sections of your interest. The searchable pdf file makes it easy to look for the relevant words.

This Minimal Introduction explains the basics of Stochastic Actor-Oriented Models and gives practical information on running RSiena. We start with section 2.1 which gives a brief and non-technical introduction to the types of Stochastic Actor-Oriented Models, to the most important concepts related to them, to the data required to apply SIENA, and to further features of the program. In Section 2.2 we explain how to install and run SIENA as the package RSiena from within R. Section 2.4 and Section 2.5 provide example R scripts and guidance for understanding the results. If you are looking for help with a specific problem, read Section 2.6.

2.1 The logic of Stochastic Actor-Oriented Models

SIENA (Simulation Investigation for Empirical Network Analysis) is a statistical tool developed for the analysis of longitudinal network data, collected in a network panel study with two or more 'waves' of observations. It incorporates different variants of a dynamic network model family: the Stochastic Actor-Oriented Model (SAOM). In this section, we give a very concise introduction to how these models work in principle and what type of data they are suitable to analyze. For sake of simplicity, SAOMs implemented in SIENA are often referred to as 'SIENA models'. In this subsection, we only consider the case of network evolution; see below for the more complex cases of coevolution. For a further introduction, consult Snijders et al. (2010b). An introduction for applications in the context of adolescent development is Veenstra et al. (2013).

The defining characteristic of Stochastic Actor-Oriented Models is their 'actor-oriented' nature which means that they model change from the perspective of the actors (nodes). That is, Stochastic Actor-Oriented Models always "imagine" network evolution as individual actors creating, maintaining or terminating ties to other actors. When thinking about network dynamics, researchers usually assume that these decisions (conscious or subconscious) of actors are influenced by the structure of the network itself and the characteristics and behaviors of the focal actor (ego) who is making a decision and those of other actors in the network (alters). Stochastic Actor-Oriented Models provide a means to quantify the ways, the extent and the uncertainty with which these factors are associated with network evolution between observations.

The Stochastic Actor-Oriented Model can be regarded as an agent-based ('actor-based') simulation model of the network evolution; where all network changes are decomposed into very small steps, so-called *ministeps*, in which one actor creates or terminates one outgoing tie. These ministeps are probabilistic and made sequentially. The transition from the observation at one wave to

the next is done by means of normally a large number of ministeps. The actors respond to the network in the sense that the probabilities of these changes depend on the current (unobserved) state of the network. Each further ministep changes the network state and therefore the actors are each others' ever changing context (Zeggelink, 1994). This allows the model to represent the feedback process that is typical for network dynamics. These changes are not individually observed, but they are simulated; what is observed is the state obtained at the next observation wave.

This simulation model implements the statistical model for the network dynamics. The statistical procedures utilize a large number of repeated simulations of the network evolution from each wave to the next. They estimate and test the parameters producing a probabilistic network evolution that 'could have' brought these observations to follow one another.

To avoid misunderstandings, two notes have to be made about the meaning of actor "decisions" and the role of Stochastic Actor-Oriented Models in causal inference. First, the fact that SIENA models are actor-oriented does not imply the assumption that the actors take decisions in any real sense. It means that the changes in the network are organized, so to say, by the nodes in the network. This aligns very well with a substantive standpoint where the nodes have agency (Snijders, 1996) but it does not necessarily reflect a commitment to or belief in any particular theory of action elaborated in the scientific disciplines. In fact, the purpose of SIENA in this matter is to assist substantive researchers in further developing their theories of action by e.g. exploring the relative importance of individual, contextual, and social factors in network change. The second, and related, point is that, like other generalized regression models, SIENA does not by itself solve all causal questions. When inferring causality from model results, one has to face difficulties very similar to those with other statistical methods; see, e.g., Lomi et al. (2011) and Goldthorpe (2001). In any case, causal interpretations should be supported by further results from the discipline the explanations originate in. However, Stochastic Actor-Oriented Models do allow research to profit from a longitudinal design – therefore, they may be helpful in tackling some issues related to causality, like the selection-influence problem (Steglich et al., 2010; Lomi et al., 2011).

2.1.1 Types of Stochastic Actor-Oriented Models: evolution of one-mode networks, two-mode networks and behaviors

So far, we have mostly talked about SIENA as a tool to analyze the evolution of a single network. However, there are different variants of Stochastic Actor-Oriented Models that can be applied to more complex data structures. The availability of these options depends on the research question and the quantity and type of data one has. In this section, we briefly discuss the currently implemented model types, which will help researchers determine what kind of analyses they are able to carry out with Stochastic Actor-Oriented Models given the data at hand.

A minimal dataset suitable for analysis with RSiena consists of two observations of a single network defined on the same set of nodes. In this case, one is able to test how the structure of the network contributes to its own evolution. However, depending on the data available, further modeling options may be applicable. Currently, the implemented Stochastic Actor-Oriented Models are suitable for the analysis of

1. the evolution of a directed or non-directed one-mode network (e.g., friendships in a classroom) (Snijders, 2001);

2. the evolution of a two-mode network (e.g., club memberships in a classroom: the first mode is constituted by the students, the second mode by the clubs) (Koskinen and Edling, 2012);

3. the evolution of an individual behavior (e.g., smoking), and

4. the co-evolution of one-mode networks, two-mode networks and individual behaviors (e.g., the joint evolution friendship and smoking; or of friendship and club membership) (Steglich et al., 2010; Snijders et al., 2013).

In all these cases, the data can also include covariates: observed variables that influence the dynamics, but of which the values are not themselves modeled.

In the first two cases, one can assess with RSiena the ways and the extent to which changes in a given one- or two-mode network depend on the network structure itself and on covariates. The third option, modeling changes in an individual behavior on its own, without reference to its embeddedness in a network, is rarely used. For this type of data numerous alternative longitudinal modeling techniques exist.

Accordingly, the fourth model type has been becoming widely used. Analyzing the joint evolution of networks and behavior allows researchers to address questions related to selection and influence processes, for example, whether smokers tend to become friends with each other or friends tend to become similar in their smoking habits. The strength of the SIENA co-evolution models is that one can simultaneously take into account the impact of network structure on network evolution, the actual level of a behavior on behavior change, the network structure on behavior change, and the actual level of behavior on network evolution. Besides network and behavior co-evolution, this class of Stochastic Actor-Oriented Models also allow for the joint analysis of multiple networks (e.g. friendship and advice, friendship and dislike, or all three of them), and the analysis of ordered multiple networks (where the presence of a tie in one network presumes the existence of a tie in the other network, like in the case of friendships and best friend relations).

2.1.2 Data, variables and effects

Now that we have discussed some core features of Stochastic Actor-Oriented Models and introduced the different implemented model types, we turn our attention, still just presenting an outline, to data types and the specification of a model. In general, the number of waves must be at least two in order to analyze a data set with Stochastic Actor-Oriented Models. In case of modeling evolution across more than two observations in time, estimated parameter values are assumed to be equal in all periods (unless time heterogeneity is specifically represented by changing parameters – see Section 6.13 for further details).

This section focuses on three related topics: the type of network and behavioral data SIENA works with, the meaning of explanatory variables, or so called effects, in Stochastic Actor-Oriented Models, and the different dependent variables with which SIENA captures network and behavior evolution.

Network data

Stochastic Actor-Oriented Models operate on binary networks, that is, on relations on a given set of actors, where tie variables between actors have two states: existent (1) or non-existent (0). Weighted networks are not allowed, but as mentioned above, it is possible to define multiple networks representing discrete levels of relationships. It is possible to specify that some ties in the network are impossible ("structural zeros") or necessary ("structural ones") (see Section 5.3.1 for more details). For the network evolution, Stochastic Actor-Oriented Models how ties are being created, maintained or terminated by actors.

Behavioral data

Behavioral variables in Stochastic Actor-Oriented Models can be thought of as indicating the presence or intensity of a behavior. For example, behavioral data can represent whether an actor is a smoker or not, as well as a number of ordered categories expressing the number of cigarettes usually smoked. The term "behavior" should not be taken literally here, it is possible to model changes in attitudes or other actor attributes. In the models, behavioral variables can be binary, ordinal discrete, or continuous. For the ordinal discrete case, the number of categories should be small (mostly 2 to 5; larger ranges are possible). In the case of behaviors, Stochastic Actor-Oriented Models express how actors increase, decrease, or maintain the level of their behavior.

A special case of the fourth type is the *diffusion of innovations in dynamic networks* (Greenan, 2015a): here the behavior variable representing having adopted the innovation is binary, coded 0 or 1, and once an actor has the value 1 s/he is stuck with it. The only possible transitions are $0 \Rightarrow 1$, representing that the actor adopts the innovation. See Section 13.2.4.

Covariates

In every model type, it is possible to define and use covariates, which are variables that are exogenous in the sense that their values are not modeled, but used to explain network or behavior change. Covariates can be dummy variables (e.g., sex) or continuous (e.g., attitudes or age). Also, they may have constant values across all observations or their value may change across time periods – this is the distinction between constant and varying covariates (e.g., sex and salary). Finally, there are individual (monadic) and dyadic covariates that refer, respectively, to characteristics of individual actors (e.g., sex) and to attributes of pairs of actors (e.g., living in the same neighborhood or kinship).

Missing data and composition change

Stochastic Actor-Oriented Models distinguish between two types of missing values: absence of actors from the network and random missingness. The first case refers to changing composition: it is possible to specify that some actors leave or join the network between two observations (during the simulation process). This then applies to all dependent variables (networks, behaviors) simultaneously (see Section 5.3.3 for more details). In the second case, missing values are treated as randomly missing (see Section 5.3.2 for more details). Stochastic Actor-Oriented Models can deal with some, but not too much, randomly missing data (as a rule of thumb, more than 20%

is considered to be too much). With too many missing values, the simulation can become unstable, and also the estimated parameters may not be substantively reliable anymore. And of course, missing data are likely to are caused by processes that are not totally random, and therefore risk to bias the resuls.

Explanatory variables: the effects

When defining Stochastic Actor-Oriented Models, we have to specify the exact ways in which current network structure or covariates may affect network or behavior change. This is defined by combinations of configurations (or situations) which are called “effects" in Stochastic Actor-Oriented Models. Effects can be treated as the explanatory variables of the models. Effects can be structural (depending on the network structure itself, also called endogenous), or covariate-related; also various combinations between structure and covariates are possible. Some examples for effects:

- *structural effects:* reciprocity, transitivity;
- *covariate effects:* sex of the tie sender, sex of the receiver, same sex, similarity in salary;
- *combinations:* average level of smoking of friends, interaction between sex of the sender and reciprocity.

Dependent variables: network evaluation, creation and endowment functions

As we discussed earlier, RSiena is capable of analyzing and modeling the evolution of networks and behavior, jointly or separately. Consequently, a model may have more than one dependent variable. Here we introduce the ways network and behavior dependent variables can be defined in Stochastic Actor-Oriented Models. We start with network evolution.

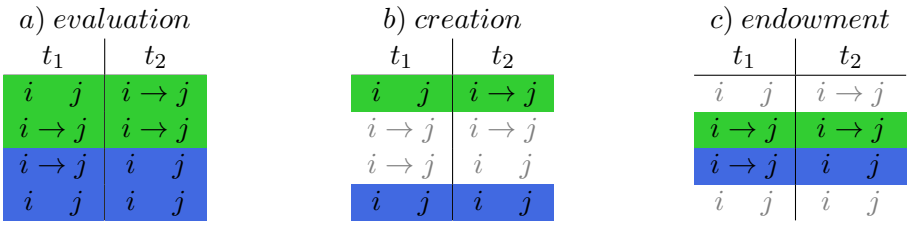
Table 1: Possible tie change patterns for two observations (t_1 and t_2)

t_1	t_2	
$i \quad j$	$i \to j$	creation of a tie
$i \to j$	$i \to j$	maintenance of a tie
$i \to j$	$i \quad j$	termination of a tie
$i \quad j$	$i \quad j$	maintenance of a ‘no-tie’

Given two observations of a binary network, a single network tie variable can follow four patterns, as shown in Table 1. In Stochastic Actor-Oriented Models, however, tie change can be defined in three ways: we can model the creation of previously not existing ties (creation), the maintenance of existing ties (endowment), or the presence of ties regardless of whether they were newly created or maintained (evaluation). These are the three possible values of the change in tie variables, constituting the dependent variables of the network evolution model. The effects

model the probabilities (more precisely: they are components of the linear predictor for the log-probability) for the creation, maintenance or presence of network ties. Table 2 helps to imagine what the probabilities refer to in each case: we compare the probability of green cases to that of blue cases.

Table 2: Tie changes considered by the evaluation, creation and endowment functions



According to this distinction, network evolution may be modeled in RSiena by three functions: the evaluation, creation and endowment functions. Effects can appear as components of one or two of these functions in a single model, but never in all three (this would lead to perfect collinearity). Using only the evaluation effect assumes that the creation and endowment effects are equal (and equal to the evaluation effect). The estimated parameters for each effect should be interpreted as log-probability ratios (comparable to the log-odds ratios known from logistic regression; since the choice here is multinomial rather than binary, the correct term is probability ratios rather than odds ratios). From a practical point of view, it is meaningful to start modeling with evaluation effects, unless one has a clear idea about how tie creation and endowment may be different in the analyzed data set. Separating the contribution of an effect into two functions requires more of the data, and if a given effect is similarly strong for the creation and maintenance of ties the statistical power will decrease by this split. For these reasons, most SI studies limit their attention to evaluation effects. However, if there is enough data, the distinction between creation and maintenance of ties can produce powerful insights (e.g., Cheadle et al., 2013).

Dependent variables: behavior evaluation, creation and endowment functions

The distinction between the different behavior evolution functions follows a logic similar to the case of network evolution. The three possibilities for change in behavior are increasing or decreasing the level of behavior by one unit, or maintaining its actual level. In case of the evaluation function, the model does not distinguish between upward and downward changes, only looks at the resulting level of behavior. By using the creation and endowment functions, we can obtain separate parameters (and assess the different impact) of effects for the increase and the decrease of behavior.

2.1.3 Outline of estimation procedure

RSiena estimates parameters by the function siena, using the following procedure:

1. Certain statistics are used that reflect the parameter values;
   the finally obtained parameters should be such that the *expected values* of the statistics are

equal to the *observed values*.
Expected values are approximated as the averages over a lot of simulated networks.
Observed values are calculated from the data set. These are also called the *target values*.

2. To find these parameter values, an *iterative stochastic simulation algorithm* is applied. This works as follows:

   (a) In Phase 1, the sensitivity of the statistics to the parameters is roughly determined.

   (b) In Phase 2, provisional parameter values are updated iteratively:
   this is done by simulating a network according to the provisional parameter values, calculating the statistics and the deviations between these simulated statistics and the *target values*, and making a little change (the 'update') in the parameter values that hopefully goes into the right direction. A lot of such updating steps are taken, each using the parameter that was produced in the preceding step.
   (Only a 'hopefully' good update is possible, because the simulated network is only a random draw from the distribution of networks, and not the expected value itself.)

   (c) In Phase 3, the final result of Phase 2 is used, and it is checked if the average statistics of many simulated networks are indeed close to the target values. This is reflected in the so-called `overall maximum convergence ratio` and the `t statistics for deviations from targets`. If some of these are too high (a threshold of 0.25 is used for the overall maximum convergence ratio, and a threshold of 0.1 for the absolute value of the t statistics for deviations from targets), the estimation must be repeated. Standard errors for the parameters are also estimated in this phase.
   If the estimation has to be repeated, this can be done by employing the argument `prevAns` in the call of siena. See the help page for siena.

If you are using Windows, when the estimation is running you will see a screen with intermediate results: current parameter values, the differences ('deviation values') between simulated and observed statistics (these should average out to 0 if the current parameters are close to the correct estimated value), and the quasi-autocorrelations discussed in Section 7. (If you are running RSiena from `RStudio`, this screen is visible when you click on the little feather that appears in the taskbar when the estimation is running.) It is possible to intervene in the algorithm by clicking on the appropriate buttons: the algorithm may be restarted or terminated. In most cases this is not necessary.

2.1.4 Design issues for longitudinal network modeling

Issues concerning the design of longitudinal network models according to the Stochastic Actor-Oriented Model are discussed in Stadtfeld et al. (2020).

2.1.5 Identifiability

Identifiability of a statistical model means that different parameter vectors imply different probability distributions. This is a basic requirement for any statistical model.

The Stochastic Actor-Oriented Model is so complicated mathematically that it is hard to give proofs of any properties. For this type of model, which comes to terms with strongly dependent random variables, mathematical proofs of desirable properties are so hard that they should not be prerequisites for the use of the model. Identifiability depends, of course, on the model specification. The Stochastic Actor-Oriented Model is a combination of generalized linear models (one for the timing and another one for the choices of tie or behavior changes) with very much missing data (the ministeps); the basis in generalized linear models suggests that – unless there are redundant effects, e.g., effects of covariates that are constant across actors and across waves – the model is identifiable indeed. The estimation algorithm is such that the fact that there is convergence (see Section 7.3.2) strongly suggests that there is identifiability.

2.1.6 Further useful options in RSiena

- Checking for time heterogeneity (Sections 6.13 and 9.5)
- Goodness of fit (Section 6.15)
- Meta-analysis of RSiena results (Section 12.3)
- Simulation without estimation (Section 10)

2.2 Installing R and RSiena

This and the next section give an overview of steps one needs to go through from installing R to running models in RSiena. Installing needs to be done only once (but should be repeated when next versions of the software appear).

1. Install R.
   This can be done from `http://cran.r-project.org/` .
   Many users prefer some kind of additional environment, such as `RStudio`, or the combination of `Notepad++` with `NppToR`.
   For Mac, you may also need X11, which is used for tcltk; but this is not a necessity. See the FAQ list for R on Mac.

2. Install the package RSiena, with dependencies. The other packages used are parallel and tools (all included in the basic R distribution); Matrix, MASS, lattice, codetools ('recommended' packages included in most R distributions), network and xtable. tcltk is optional. For goodness of fit testing it will be useful also to install sna and igraph.

   You can download the latest version of RSiena from GitHub or the SIENA website. To do this, go to
   `https://github.com/stocnet/rsiena/releases`
   or to
   `http://www.stats.ox.ac.uk/~snijders/siena/siena_downloads.htm`
   and there download the appropriate version of the package appropriate for your operation system (Windows, Mac, Unix).

Installation from GitHub can be done as follows. Download the latest release from https://github.com/stocnet/rsiena/releases/. Once the file has been downloaded, unzip it, and install the binary as appropriate for your Operating System. This can be done in Windows using the dropdown menu under `Packages`, and for all Operating Systems using function install.packages with `repos = NULL`.
In RStudio, go to `Tools` → `Install packages` and choose
`Install From: Package archive file (zip; tar.gz)`.

Alternatively, install the `remotes` package in R, and run

```
remotes::install_github("stocnet/rsiena", build=FALSE)
```

2.3 Using RSiena within R

1. Load data (networks, behavior, covariates) into R (see Section 5.1):
   (a) Network data should be in objects of class matrix. An alternative is to use sparse matrices of class `"TsparseMatrix-class"` from package Matrix.
   (b) Behavioral data should be in objects of class matrix.
   (c) Individual constant covariates should be in objects of class vector or should be in columns or rows of a matrix.
   (d) Individual varying covariates should be in objects of class matrix.
   (e) Dyadic covariates should be in objects of class matrix; again, an alternative is sparse matrices of class `"TsparseMatrix-class"`.
2. All missing data should be set to `NA` (see Section 5.3.2).
3. Check whether your data objects meet the following criteria:
   (a) Each object contains the same nodes/actors;
   (b) Nodes are in the same order in each object;
   (c) Nodes are in the same order in rows and columns of matrix objects (in case of one-mode networks)[2].

   If a two-mode network is studied, then of course there will be two node sets.
4. Create data objects suitable for RSiena using the appropriate functions (see Section 5.1):
   (a) as_dependent_rsiena for networks and behavior variables;
   (b) only for two-mode networks, as_nodeset_rsiena for defining nodesets;
   (c) as_covariate_rsiena and as_covariate_rsiena for constant and changing/varying individual covariates respectively;
   (d) as_covariate_rsiena and as_covariate_rsiena for constant and changing/varying dyadic covariates respectively;

[2]For directions on how to handle composition change, see Section 5.3.3

(e) In case of two-mode networks, for each object it should be specified which nodeset it is defined on, using the nodeSets argument in the above functions.

5. Create a siena data object containing all the objects specified above using the function make_data_rsiena (see Section 5.1).

6. Use make_specification to create an effects object. This already gives a very simple model specification containing the outdegree and a reciprocity effects (see Section 6.2 - for two-mode networks see Section 6.3).

7. You can use various algorithm settings. Alternative settings of the estimation algorithm can be defined by set_algorithm_saom (see Section 7.2); special model definitions can be chosen using function set_model_saom; and output options can be chosen by set_output_saom. For a first model try these can be skipped.

8. Use write_report to produce an output file presenting some descriptive statistics for the objects included in the model.

9. Use functions set_effect, set_effect and set_interaction to further specify the model (see Sections 6.2 – 6.8).

10. Use siena to run the estimation procedure.[3]

11. Basic output is written to a text file in the actual working directory. The default filename is the name of the data object to which `_out.txt` is appended. Results can also be inspected in R using various functions.

2.4 Example R scripts for getting started

The following scripts on the RSiena website go through the steps outlined in the previous section, providing additional details and options:

- basicRSiena.r: a minimal example of a basic sequence of commands for estimating a model by function siena of RSiena.
- Rscript01DataFormat.R: gives a brief overview of R functions and data formats that are essential for using RSiena.
- Rscript02SienaVariableFormat.R: shows how to prepare data for a RSiena analysis, including the creation of RSiena objects; and how to specify effects for RSiena models.
- Rscript03SienaRunModel.R: shows how to carry out the estimation and look at the results;

[3] The use of multiple processes can speed up the estimation. For directions on how to utilize multiple processors, see Section 7.13.

- Rscript04SienaBehaviour.R: illustrates how to specify models for dynamics of networks and behaviour.

The website contains a lot of other scripts illustrating other functionalities of RSiena.

2.5 Steps for looking at results: Executing SIENA.

1. Look at the start of the output file obtained from write_report for general data description (degrees, etc.), to check your data input and get a general overview of the data set.

   In this file, there is a section 'Change in networks' which contains some basic descriptives. Some of these refer to the *periods*: these are the combinations of two successive waves. For example, a two-wave data set has one period, and a three-wave data sets has periods $1 \Rightarrow 2$ and $2 \Rightarrow 3$. The `Distance` mentioned there is the Hamming distance between successively observed networks, i.e., the number of tie variables that differ. The `Jaccard` index is the Jaccard similarity index between the successive networks:

   $$\frac{N_{11}}{N_{01} + N_{10} + N_{11}},$$

   where N_{hk} is the number of tie variables with value h in one wave and value k in the next wave. The Jaccard index is a measure for stability; see Snijders et al. (2010b). Both for the Hamming distance and the Jaccard index, only those cells in the adjacency matrix are counted that have available data in the wave at the start and the wave at the end of the period concerned.

   If Jaccard indices are very low while the average degree is not strongly increasing, this indicates that the turnover in the network may be too high to consider the data as an evolving network, and perhaps the SIENA method is not suitable for the data set. For networks of the type that are mostly used for this method (sparse but not too sparse, with average degrees not too different from wave to wave and between 2 and 15 for all waves), Jaccard values of .3 and higher are good; values lower than .2 indicate that there might be difficulties in estimation; values lower than .1 are quite low indeed. Using the SIENA method for two waves with an extremely low Jaccard index and average degrees that remain more or less constant will mean that the first wave hardly plays a role in the results, and for non-conditional estimation it will be close to treating the second wave as a sample from the stationary distribution of the network dynamics.

   If Jaccard indices are low because the network is mainly increasing (creation of new ties) or decreasing (termination of ties), this is no problem for the SIENA method. Very sparse networks (with most degrees less than 2) also may have lower Jaccard values without negative consequences for estimation.

2. When parameters have been estimated, first look at the `overall maximum convergence ratio` and the `t statistics for deviations from targets`. We say that the estimation has converged if the former is less than 0.25, and the latter all are smaller than 0.1 in absolute value; and that it has nearly converged if the former is less than 0.35, and the latter are all smaller than 0.15. Results obtained for non-converged estimation

runs may be misleading. (Very small deviations from these values are of course immaterial.) See Section 7.3.2.

3. In rare circumstances, when the data set leads to instability of the algorithm, the following may be of use. The Initial value of the gain parameter determines the step sizes in the parameter updates in the iterative algorithm. This is the parameter called firstg in function set_algorithm_saom. A too low value implies that it takes very long to attain a reasonable parameter estimate when starting from an initial parameter value that is far from the 'true' parameter estimate. A too high value implies that the algorithm will be unstable, and may be thrown off course into a region of unreasonable (e.g., hopelessly large) parameter values. It usually is unnecessary to change this, but in some cases it may be useful.

4. If all this is to no avail, then the conclusion may be that the model specification is incorrect for the given data set.

5. Further help in interpreting output is in Section 7.8 of this manual.

2.6 Getting help with problems

For methodological help, consult the tutorial Snijders et al. (2010b) or this manual. The website, `http://www.stats.ox.ac.uk/~snijders/siena/` , contains various further publications (also in other languages than English) that may be helpful, as well as example scripts. There is a users' group for SIENA to exchange information and seek technical advice; the address is `https://groups.io/g/RSiena` .

For technical problems running RSiena, follow the following points.

Help pages Study the R help page for the function you are using and that seems to give the problems. This manual complements the help pages, but does not replace them!

Check your version of RSiena The 'News' page of the SIENA website gives information about new versions of RSiena. Details of the latest version available can be found at the 'Downloads' page. The version is identified by a version number (e.g., 1.6.14). You can find the number of your current installed version by opening R , and typing
`packageVersion("RSiena")`
or
`packageDescription("RSiena")`. The version is near the top. It is also displayed at the start of RSiena output files.

Check your version of R When there is a new version or revision of RSiena it will only be available to you automatically if you are running the most recent major version of R. (You can force an installation if necessary by downloading the tarball or binary and installing from that, but it is better to update your R.)

Check both repositories We have two repositories in use for RSiena: CRAN and GitHub. The latest version will always be available from GitHub. (Frequent updates are discouraged on CRAN, so bug-fixes are likely to appear first on GitHub.)

Installation When using the repository at GitHub, *install* the package rather than updating it. Then check the version and revision numbers.

Users' group Consult the archives of the Users' Group mentioned above, or post a message to the Users' Group. In your message, please tell which operating system, which version of R, and which version of RSiena you are using.

3 New function names from Version 1.6

In Version 1.6, many functions in RSiena have been given new names. This was done in order to make them more readily comprehensible, to give a somewhat more logical structure, and for coordination with other packages in the StOCNET suite. The old function names still are available in order for existing scripts still to be useful also for the newer versions of RSiena.

The same holds for multiSiena since version 1.2.36.

Some example scripts on the web still use mainly the old function names, but this soon will be changed. The descriptions of the old and the new function names are in the same help pages in the R system.

For many functions this is just a name change. The main differences are the following.

1. The algorithm now does not have to be specified for standard use. The name of the output text file generated by the estimation is called `dataName_out.txt`, where `dataName` is the name of the data object.

2. Constructing the algorithm is split into three functions:

   (a) set_model_saom for defining the general model (e.g., `modelType`);

   (b) set_algorithm_saom for defining the method and algorithm for estimation (e.g., settings of the Robbins-Monro algorithm, maximum likelihood estimation);

   (c) set_output_saom for defining the output to be created (e.g., different name for the text output file).

3. The estimation function now is called siena instead of siena07. Its first argument is the data set; omitting the algorithm objects is usual, and will lead to a default estimation.

4. The estimation function in multiSiena now is called multi_siena instead of sienaBayes. The help page
   `?multi_siena`
   gives enough information.

5. Constructing covariates is done by one function, as_covariate_rsiena; the distinction between monadic and dyadic, and between changing or time-constant, is made by the structure of the input object and the further arguments (see the table below).

6. Including effects is done by one function, set_effect.
   If more than one effect is to be included by one function call, the `shortNames` are to be given in a list.
   The argument names now are `depvar`, `covar1`, and `covar2`, replacing the earlier (less fortunate) names `name`, `interaction1`, and `interaction2`.

7. For defining interactions, the function set_interaction requires to give the `shortNames` of the two or three interacting effects in a list.
   Like for set_effect, argument names now are `depvar`, `covar1`, and `covar2`.

In departure of the general rule that the old names are retained in view of compatibility, the following names were canceled:

8. The names sienaRI and sienaRIDynamics are not retained, and replaced by interpret_size and interpret_size_dynamics; these functions still are not final.

9. The name siena.table is not retained but replaced by siena_table; however, the new (and preferred) name is write_result.

10. model.create, an alias for sienaAlgorithmCreate, was dropped.

Table 3 gives the correspondence between the old and new names.

Old name	new name	how
sienaDependent	as_dependent_rsiena	A
coCovar	as_covariate_rsiena	A
varCovar	as_covariate_rsiena(..., type="monadic")	C
coDyadCovar	as_covariate_rsiena(..., type="oneMode")	C
	or as_covariate_rsiena(..., type="bipartite")	
varDyadCovar	as_covariate_rsiena	A
sienaCompositionChange	as_composition_rsiena	A
sienaCompositionChangeFromFile	as_composition_file_rsiena	A
sienaNodeSet	as_nodeset_rsiena	A
sienaDataCreate	make_data_rsiena	A
sienaGroupCreate	make_group_rsiena	A
getEffects	make_specification	A
print01Report	write_report	A
includeEffects	set_effect	C
setEffect	set_effect	C
includeInteraction	set_interaction	C
sienaAlgorithmCreate	set_model_saom,	
	set_algorithm_saom,	
	set_output_saom	
sienaDataConstraint	make_constraint	A
siena07	siena	C
siena08	meta_siena	A
siena.table ⊗	write_result	A
Multipar.RSiena	test_parameter	A
Wald.RSiena	test_parameter	C
testSame	test_parameter(..., method="same", ...)	C
score.Test	test_parameter(..., method="score", ...)	C
sienaTimeTest	test_time	A
sienaGOF	test_gof	A
descriptives.sienaGOF ⊗	descriptives	A
selectionTable	interpret_selection	A
influenceTable	interpret_influence	A
sienaRI ⊗	interpret_size	C
sienaRIDynamics ⊗	interpret_size_dynamics	A
updateTheta	update_theta	A
updateSpecification	update_specification	A
	and for multiSiena :	
sienaBayes ⊗	multi_siena	C
simulateData ⊗	simulate	C
glueBayes ⊗	multi_glue	A
extract.sienaBayes ⊗	extract_sienaBayesFit	A
extract.posteriorMeans ⊗	extract_posteriorMeans	A

Table 3: Correspondence between old and new function names.
'how' column: 'A' means that the change is just an alias, or can be treated as such; 'C' indicates that the arguments also changed. ⊗ means that the function with the old name is not retained.

3.1 Function transformScript

RSiena contains function transformScript, which transforms R scripts with calls of RSiena and multiSiena functions to scripts utilizing the new names. See its help page!

The function transformScript operates on the script (or other text files, e.g., LaTeX) line by line. It gives output in the new script line by line, with an exception for calls of sienaAlgorithmCreate. As this function now is split into three, no line in the new script is created; but calls of siena07 and sienaBayes are checked for the algorithms they use, and transformed into calls of siena and mult_siena, respectively, preceded by calls of set_model_saom, set_algorithm_saom, and/or set_output_saom, corresponding to the algorithm object used by this call of siena07 or sienaBayes. These algorithm objects constructed always have the names, respectively, `alg_model`, `alg_alg`, and `alg_out`.

Function transformScript also produces a log file. If it leads to an error, it may be helpful to use
`traceback()`
immediately after the error occurs; and to look at the log file, which will have line numbers until the last line that was treated.

If you want to transform to the new names all files in the working directory with extension name `.R`, you could use the following script.

```
listed <- list.files(pattern = "\\.R")
listeddat <- list.files(pattern = "\\.RData")
listeddat1 <- list.files(pattern = "\\.Rmd")
diflist <- setdiff(listed, listeddat) # exclude all file names containing ".RData"
diflist <- setdiff(diflist, listeddat1) # exclude all file names containing ".Rmd"
(nfiles <- length(diflist))
for (h in (1:nfiles)){
   oldname <- diflist[h]
   thescript <- readLines(oldname)
   newname <- paste("new", oldname, sep="")
   logname1 <- sub(".R",  ".log", oldname, fixed=TRUE)
   logname <- paste("new", logname1, sep="")
   cat("\n", h, ":  ", oldname, "\n")
   flush.console()
   trs <- transformScript(thescript, fileName=newname,
            linelength=80, initialblanks=6, logfile=logname )
}
```

If and when errors in a newly created script occur, it is good to understand how the function transformScript operates. It passes three times through the old script:

1. The first time, to replace the names of functions in category 'A' in Table 3.

2. In the second pass, calls of RSiena and multiSiena functions in category 'C' are transformed to calls utilizing the new names.
   The only RSiena functions in the script that are executed themselves here are calls of sienaAlgorithmCreate; their results are used for transforming calls of siena07 and sienaBayes.
   All function calls in category 'C' except those of sienaAlgorithmCreate are replaced with calls of the new functions.

3. In the final pass, if any old function names are left and used without being a function call (i.e., without being followed by "("), they are replaced with the new names.

These three passes are documented in the log file.

If in the algorithm used for siena or multi_siena (or its call) a variable occurs, then this variable may be unknown to transformScript which will lead to an error. For example:

```
Error in eval(parse(text = newcommand1)) :  object 'nodes' not found
```

Just create the missing object and give it a value, and run transformScript again. In the generated script you probably will have to modify this object.

If you cannot easily make sense of the error, it may be helpful to look at the log file, which will have line numbers until the last line that was treated.
If this does not help, you could use

```
traceback()
```

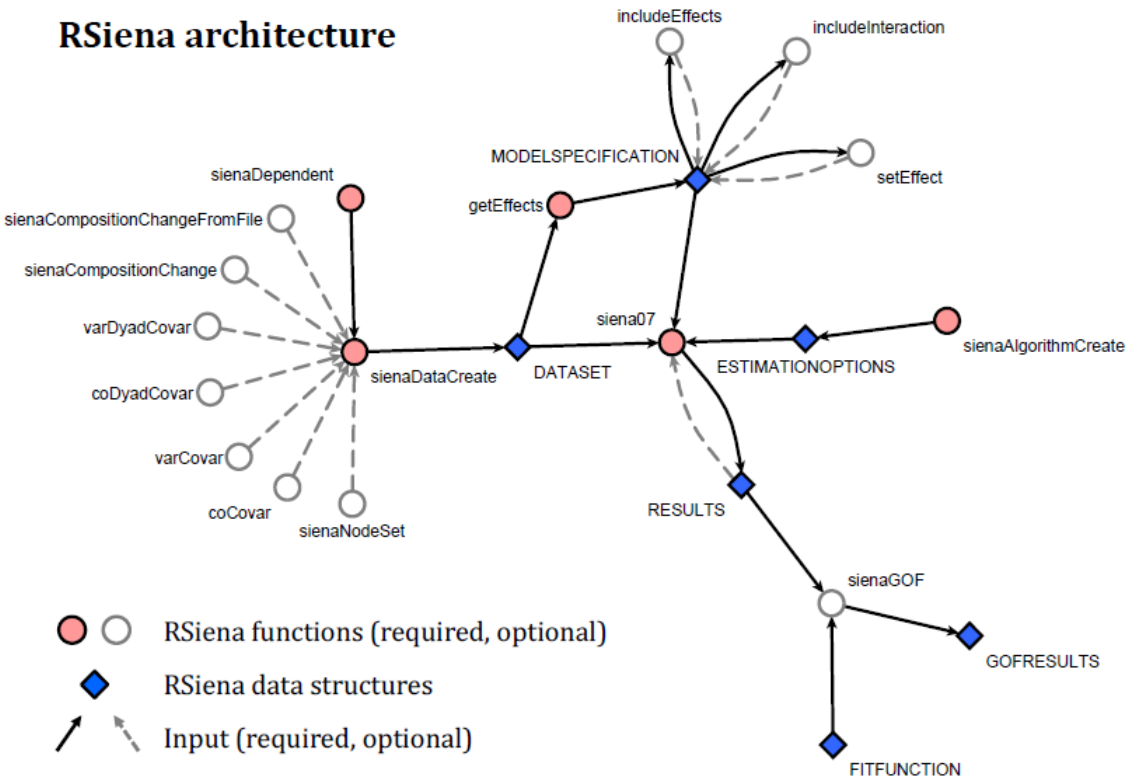
immediately after the error occurs; and try to understand.

4 Steps of modelling

The operation of the RSiena program is comprised of five main parts:

1. input of basic data description (see Section 5);
2. model specification (see Section 6);
3. estimation of parameter values using stochastic simulation (see Section 7); the main output of the estimation procedure is written to a text file named *dname*_out.txt, where *dname* is the name of the data set; a different name can be specified using function set_output_saom;
4. testing parameters and assessing goodness of fit (see Sections 8 and 9);
5. simulation of the model with given and fixed parameter values (see Section 10).

The normal operation is to start with data input, then specify a model and estimate its parameters, assess goodness of fit and the significance of the parameters, and then possibly continue with new model specifications followed by estimation or simulation.



5 Input data

RSiena is a program for the statistical analysis of repeated measures of social networks, and requires, at the very least, network data collected at two or more time points. It is also possible to include other types of variables in the models – these are discussed in Section 5.1. Section 5.2 describes the most commonly occuring data transformations that are done internally by SIENA. Finally, Section 5.3 shows further options for users to define their data.

5.1 Data types

As we discussed in Section 2.1.2, dependent variables in Stochastic Actor-Oriented Models are defined from network or behavioral data. Independent variables (effects) are defined from individual or dyadic covariate data, which can be constant or varying. RSiena requires each of these data types to have a specific format – this is presented in the current section.

In general, data specification in RSiena consists of two steps. First, the role of each variable to be used must be defined using the functions as_dependent_rsiena, as_covariate_rsiena, as_covariate_rsiena, as_covariate_rsiena, as_covariate_rsiena, or as_composition_rsiena. Second, the variables must be combined into one siena data set by the function make_data_rsiena. This function puts together the data set and carries out some preliminary calculations.

It is advisable to use names of variables consisting of at most 12 characters. This is because they are used as parts of the names of effects which can be included in the model, and the effect names should not be too long.

RSiena does not work with case numbers. The correspondence between cases in the different components of the data set is by the order of the rows in the data matrices. For a data set with n actors, each data matrix should have n rows and always the i'th row should correspond to the i'th actor.

It is also useful to note here that in case of co-evolution models (those with more than one dependent networks and/or behaviors), data for all dependent variables must be available for the same set of time points.

5.1.1 Network data

There can be one or more networks in the data set. Networks can be one-mode or two-mode. Modes are identified with node sets. All one-mode networks must have the same node set, also referred to as 'actors'. If there are some one-mode networks and any two-mode networks, the first node set of all two-mode networks should be the node set (actor set) of the one-mode networks. The choice between one- and two-mode is specified by the argument `type` in the as_dependent_rsiena function, which for networks can be `oneMode` or `bipartite`; the latter signifies a two-mode network. If there are one-mode as well as two-mode dependent networks, the one-mode networks should be mentioned first in the call of as_dependent_rsiena.

For data specification by as_dependent_rsiena, the network must be specified as a matrix or array or list of sparse matrix of triples. The help page for as_dependent_rsiena has an example of input by sparse matrices.

For data specification by the function make_data_rsienaFromSession, edge list formats are also allowed. This can be either the format of the Pajek program, or a raw edge list, here called Siena format. For large number of nodes (say, larger than 100), the edge list format is more efficient in use of computer memory.

Sparse matrices, which can be used by input via as_dependent_rsiena, have for running the estimations the same efficiency as Pajek or Siena format. The three possible formats for digraph input are as follows.

1. *Adjacency matrices*.
   These can be used in as_dependent_rsiena and in make_data_rsienaFromSession.
   In the usual case of a one-mode network the adjacency matrix is given in a matrix of n rows and n columns containing integer numbers. The diagonal values are meaningless but must be present. In the case of a two-mode network (which is a network with two node sets, and all ties are between the first and the second node set) the matrix does not have to be square, as usually the number of nodes in the first set will not be equal to the number of nodes in the second set; and if it would be square, the diagonal still would be meaningful.

   Although this section talks only about digraphs (directed graphs), for one-mode networks it is also possible that all observed adjacency matrices are symmetric. This will be automatically detected by SIENA, and the program will then utilize methods for non-directed networks.

   The values of the ties must be 0, 1, or `NA` (not available = missing); or 10 or 11 for structurally determined values (see below).

   The help file for as_dependent_rsiena shows by examples how the specification can be given by sparse matrices.

Missing values must be indicated in the way usual for R, by `NA`.

If the data set is such that it is never observed that ties are terminated, then the network dynamics is automatically specified internally in such a way that termination of ties is impossible. (In other words, in the simulations of the actor-based model the actors have only the option to create new ties or to retain the status quo, not to delete existing ties.) Similarly if ties never are created (but only terminated), then this will be respected in the simulations. See Section 5.2.4 and note the possibility of using allowOnly=TRUE.

5.1.2 Transformation between matrix and edge list formats

The following R commands can be used for transforming an adjacency matrix to an edge list, and back again. If `a` is an adjacency matrix, then the following commands can be used to create the corresponding edge list, called `edges` here.

```
# create indicator matrix of non-zero entries of a
ones <- !a %in%  0
# create empty edge list of desired length
edges <- matrix(0, sum(ones), 3)
# fill the columns of the edge list
```

```
edges[, 1] <- row(a)[ones]
edges[, 2] <- col(a)[ones]
edges[, 3] <- a[ones]
# if desired, order edge list by senders and then receivers
edges      <- edges[order(edges[, 1], edges[, 2]), ]
```

Some notes on the commands used here:
These commands can be used not only if the adjacency matrix contains only 0 and 1 entries, but also if it contains values `NA`, 10, or 11. The possibility of `NA` entries requires special attention; `%in%` does just what we need, as it quietly says that `NA`'s are not `%in%` anything, returning `FALSE`, which is transformed to `TRUE` by the `!` function. The edge list is created having all 0 values and at the end should have no 0 values at all.

It is more efficient, however, to work with sparse matrices; this also is done internally in RSiena. Using the `Matrix` package for sparse matrix manipulations, the same results can be obtained as follows.

```
library(Matrix)
tmp <- as(a, "dgTMatrix")
edges2 <- cbind(tmp@i + 1, tmp@j + 1, tmp@x)
```

Conversely, if `edges` is an edge list, then the following commands can be used to create the corresponding adjacency matrix, called `adj`, with n nodes. (For a bipartite network the two dimensions will normally be distinct numbers.)

```
# create empty adjacency matrix
adj <- matrix(0, n, n)
# put edge values in desired places
adj[edges[, 1:2, drop = FALSE]] <- edges[, 3, drop = FALSE]
```

Note that this starts with a matrix having all 0 entries, and results in a matrix with no 0 entries at all. To check the results, after doing these two operations, the command

```
length(which(a != adj))
```

should return the value 0.

Note that the basic edge list, `edges`, lacks information as to the size of the adjacency matrix. `tmp` above is a sparse matrix which is in edge list format but includes information on the size of the adjacency matrix, and can be used in a similar way to the original matrix `a` while saving memory space.

For some insight into how this is done with RSiena, you could request

```
RSiena:::sparseMatrixExtraction0
```

to see how this function is defined, using either the data set (for `i == NULL`) or the simulated data from siena (for `i != NULL`).

5.1.3 Behavioral data

RSiena also allows dependent behavior variables. This can be used in studies of the co-evolution of networks and behavior, as described in Snijders et al. (2007) and Steglich et al. (2010). These behavior (or 'action') variables represent the actors' behavior, attitudes, beliefs, etc. The difference between dependent behavior variables and changing actor covariates (see below) is that the latter have values determined by the input data and are assumed to change exogenously, i.e., according to mechanisms not included in the model, while the dependent action variables change endogenously, i.e., depending on their own values and on the changing network. Unlike the changing individual covariates, the values of dependent action variables are not assumed to be constant between observations.

Dependent behavioral variables can be ordinal discrete or continuous. This is indicated by the values `behavior` or `continuous`, respectively, for argument `type` in as_dependent_rsiena.

Ordinal discrete behavioral variables must have nonnegative integer values; e.g., 0 and 1, or a range of integers like 0,1,2 or 1,2,3,4,5. The number of different values should not be too high: ten values is on the high side, but can still be totally adequate. For categorization of variables that originally are continuous or have many values, it often is advisable to take approximately equal intervals on the original scale, and to achieve a rather smooth distribution of values, which might resemble a discretized normal or skew distribution, as the case may be. Using a small number of values may amount to a loss of information. If the original variable is continuous then 8 to 10 categories may be a good number. This is studied in Niezink (2018, Chapter 6).

Each dependent action variable must be given in one matrix, containing M columns, corresponding to the M observation moments.

If any values are not integers but `type=behavior`, a warning will be printed on the initial report given by write_report and the values will be truncated towards zero.

A special case of behavioral data can be used for *diffusion of innovations* (Greenan, 2015a): here the behavior variable representing having adopted the innovation is binary, coded 0 or 1, and changes $1 \Rightarrow 0$ are impossible. Model specifications that are especially useful for this data type are presented in Section 13.2.4.

5.1.4 Continuous behavioral data

For continuous behavioral data, see `https://www.stat.cmu.edu/~nynke/code.html`.

5.1.5 Individual covariates

Individual (i.e., actor-bound, or monadic) variables are defined by the functions as_covariate_rsiena in the case they are constant over time, and as_covariate_rsiena if they are changing over time.

Each constant actor covariate has one value per actor valid for all observation moments, and has the role of an independent variable.

Changing variables can change between observation moments; then they are called 'changing individual covariates', and have the role of independent variables.

The internal operation of RSiena is not scale-independent. It is advisable to scale the covariates so that they have standard deviations in the range between 0.1 and 10.

Changing individual covariates are assumed to have constant values from one observation moment to the next. If observation moments for the network are $t_1, t_2, ..., t_M$, then the changing covariates should refer to the $M-1$ moments t_1 through t_{M-1} , and the m-th value of the changing covariates is assumed to be valid for the period from moment t_m to moment t_{m+1} . The value at t_M, the last moment, does not play a role. Changing covariates, as independent variables, are meaningful only if there are 3 or more observation moments, because for 2 observation moments the distinction between constant and changing covariates is not meaningful.

Each changing individual covariate must be specified in a separate call of as_covariate_rsiena, using for input an $n \times (M-1)$ matrix where the columns correspond to the $M-1$ periods between observations.

By default, the mean is subtracted from the covariates. This can be turned off by constructing the covariate with the argument `centered=FALSE`. Centering of covariates improves convergence of the estimation algorithm. Therefore the advice is to use the default (centering) unless there are reasons to use non-centered covariates, such as when they are used as weights in effects (e.g., for `inPopX`). It is possible to include a given actor variable in the Siena data set as a covariate in the centered as well as the non-centered version. See Section 5.2.2 on centering.

5.1.6 Dyadic covariates

Like the digraph data, also each measurement of a dyadic covariate must be contained in a separate matrix. For one-mode data this is a square data matrix, and the diagonal values are meaningless.

A distinction is made between constant and changing dyadic covariates, where change refers to changes over time. Each constant covariate has one value for each pair of actors, which is valid for all observation moments, and has the role of an independent variable. Changing covariates, on the other hand, have one such value for each period between measurement points. If there are M waves (i.e., observation moments) of network data, this covers $M-1$ periods, and accordingly, for specifying a single changing one-mode dyadic covariate, a $n \times n \times (M-1)$ array is needed.

Two-mode dyadic covariates also are possible; see the help pages for as_covariate_rsiena and coDyadCovar,

The internal operation of RSiena is not scale-independent. It is advisable to scale the covariates so that they have standard deviations in the range between 0.1 and 10.

Like is the case for monadic covariates, changing dyadic covariates are assumed to have constant values from one observation moment to the next. If observation moments for the network are $t_1, t_2, ..., t_M$, then the changing covariates refer to the $M-1$ moments t_1 through t_{M-1} , and the m-th value of the changing covariates is assumed to be valid for the period from moment t_m to moment t_{m+1} . The value at t_M, the last moment, does not play a role.

Constant dyadic covariates are specified using function as_covariate_rsiena, and changing dyadic covariates by as_covariate_rsiena.

The mean is always subtracted from the covariates. See Section 5.2.2 on centering.

5.2 Internal data treatment

5.2.1 Interactions and dyadic transformations of covariates

For actor covariates (also called monadic covariates), two kinds of transformations to dyadic covariates are made internally in SIENA, by the definition of the corresponding effects. Denote the actor covariate by v_i, and the two actors in the dyad by i and j. Suppose that the range of v_i (i.e., the difference between the highest and the lowest values) is given by r_V. The two transformations are the following:

1. *dyadic similarity* is defined by

$$\text{sim}_{ij}^{V} = 1 - \frac{|v_i - v_j|}{r_V} \,. \tag{1}$$

   This similarity measure is 1 if the two actors have the same value, and 0 if one has the highest and the other the lowest possible value. For the various similarity effects, this measure is centered by subtracting the mean over all dyads, so the mean of this similarity variable becomes 0.
   The advantage of this similarity measure is that its coefficients are comparable in the sense that the measure always ranges between 0 and 1.
   The mean of (1) is calculated by function make_data_rsiena and stored as the simMean attribute of `mydata$cCovars$myvar`, where `mydata` is the name of the object created by make_data_rsiena, and `myvar` is the name of the variable used as the argument for make_data_rsiena, while the name `cCovars` applies for constant monadic covariates, and is to be replaced by `vCovars` for changing (varying) monadic covariates; for centering issues, further see Section 5.2.2.

2. *same V* is defined by

$$\text{same}_{ij}^{V} = \begin{cases} 1 & \text{if } v_i = v_j \\ 0 & \text{if } v_i \neq v_j \,. \end{cases} \tag{2}$$

   This can also be referred to as *dyadic identity* with respect to V.

Dyadic similarity is relevant for variables that can be treated as interval-level variables; dyadic identity is relevant for categorical variables.

In addition, RSiena offers the possibility of user-defined two- and three-variable interactions between covariates; see Section 6.12.

5.2.2 Centering

Individual as well as dyadic covariates can be centered by the program in the following way.

For individual covariates, the mean value is subtracted by function make_data_rsiena. The centered values then are stored (see below), and all calculations use these centered variables. For the changing covariates, the mean value used is the global mean (averaged over all periods). The values of these subtracted means are reported in the output of write_report. For the multi-group option (section 12.2), the subtracted values are the global means across all groups.

Centering is the default. Centering of covariates can be turned off by specifying `centered=FALSE` in the call of as_covariate_rsiena, as_covariate_rsiena, as_covariate_rsiena, or as_covariate_rsiena, respectively. Centering of covariates improves convergence of the estimation algorithm. Therefore the advice is to use the default (centering) unless there are reasons to use non-centered covariates, such as when they are used as weights in effects (as is the case, e.g., for effect `inPopX`). If some actor variable is to be used as a non-centered covariate, it is recommended to include it as a covariate in the centered as well as the non-centered version in the Siena data set, and use the non-centered version only when it is necessary to get the appropriate effect definition.

For the dyadic covariates and the similarity variables derived from the individual covariates, the grand mean is calculated and stored by function SienaDataCreate; the stored values of the variables are not centered, but the means are subtracted during the program calculations. (Thus, dyadic covariates are treated internally by the program differently than individual covariates in the sense that the mean is subtracted at a different moment, but the effect is the same; except for multi-group data sets, see below.) Unlike the 'covariate similarity' effect, the 'same covariate' effect is not centered but keeps its 0-1 values.

For the multi-group option (section 12.2), dyadic covariates are treated differently from individual covariates: for dyadic covariates in multi-group data sets, centering is done by the within-group mean; actor covariates in multi-group data sets created by make_group_rsiena are centered by the overall mean.

For dependent behavioral variables, the effects are defined in Section 13.2 as functions of centered variables.

The means of covariates are stored as attributes on the object created by make_data_rsiena. If you wish to access them, the following steps can show where these means can be found. For example, suppose that the command given was

```
mydata <- make_data_rsiena( friendship, smoke1, alcohol )
```

The structure of this object is obtained by requesting

```
str(mydata, 1)
```

Looking at the response, you will see that this object contains (among other things):

1. the constant actor covariates as `mydata$cCovars`
2. the varying actor covariates as `mydata$vCovars`
3. the constant dyadic covariates as `mydata$dycCovars`
4. the varying dyadic covariates as `mydata$dyvCovars`
5. the dependent behavioral variables as `mydata$depvars`

Since `smoke1` is a constant covariate and `alcohol` a changing covariate, their means can be requested by

```
attr(mydata$cCovars$smoke1, "mean")
attr(mydata$vCovars$alcohol, "mean")
```

and the centered values for, e.g, the variable `alcohol` by

```
mydata$vCovars$alcohol
```

The mean of the similarity variable is stored as the simMean attribute, and is obtained by, e.g.,

```
attr(mydata$cCovars$smoke1, "simMean")
```

The formula for balance is a kind of dissimilarity between rows of the adjacency matrix. The mean dissimilarity is subtracted in this formula, having been calculated according to a formula given in Chapter 13. It is also reported in the output file produced by write_report and available – e.g., for the first dependent variable – as

```
attr(mydata\$depvars[[1]], "balmean")
```

Instead of `[[1]]` you can request a different number or the name of the variable.

For multi-group data sets constructed by function make_group_rsiena, the attribute `"vCovarMean"` contains the overall means of the actor covariates.

5.2.3 Centering of behavioral variables

Dependent behavior variables are not centered by make_data_rsiena, but internally during the simulations. Therefore their mean is not stored in the Siena data set. However, the values and their frequencies are stored in (using `mydat` and `smoke` as example names)

```
attr(mydata$depvars$smoke, "vals")
```

From this, the mean can be calculated by using the functions

```
theMean <- function(x){
# Mean of a named vector of frequencies, where the names are the values
    sum(as.integer(names(x)) * x, na.rm = TRUE)/sum(x, na.rm = TRUE)
}
meanByWave <- function(x, vari, w){
    theMean(attr(x$depvars[[vari]], "vals")[[w]])
}
grandMean <- function(x, vari){
    xxx <- attr(x$depvars[[vari]], "vals")
    mean(sapply(xxx, theMean))
}
```

For comprehensibility, these three functions are given separately here. The first is a general utility function, the second and third specific for RSiena. They can be applied, e.g., as

```
grandMean(mydata, "smoke")
sapply(1:mydata$observations, meanByWave, x = mydata, vari = "smoke")
```

and for multi-group data as

```
sapply(myGroupData, grandMean, vari = "smoke")
```

5.2.4 Monotonic dependent variables

In some data sets, a dependent variable only increases, or only decreases. For a network, this means that ties can be created but not terminated, or the other way around. This may be the case for all periods (a period is defined by the two consecutive observation waves at its start and end points) or just in some of the periods. RSiena will note when a dependent variable only increases or only decreases in any given period, and mention this in the output file generated by write_report. This constraint then is also respected in the simulations, in the periods where it is observed. This is represented internally by a variable called `uponly` indicating that the dependent variable cannot decrease, and a variable `downonly` indicating that the dependent variable cannot increase. The constraints signaled by the `uponly` and `downonly` variables can be lifted by using `allowOnly = FALSE` in the call of as_dependent_rsiena (see the help file for this function).

If a dependent variable is only increasing or only decreasing for all periods and as_dependent_rsiena was called with allowOnly=TRUE (the default), then two basic effects are not identified. These are the outdegree effect for a dependent network variable, and the linear shape effect for a dependent behavior variable; these effects define the balance between the probabilities of going up and going down. These effects then are dropped automatically from the effects object. If this is not desired, this can be prevented by calling as_dependent_rsiena with allowOnly=FALSE.

5.2.5 Multivariate dependent variables with constraints

When analysing multiple dependent networks, it is possible that they are related in some deterministic way. For example, two networks might be mutually exclusive (this could be the case for negative and positive ties – although there are many cases where positive and negative ties may coexist!), or networks could be ordered, i.e., one network could be a sub-network of the other (networks with ordinal tie values represented as a multivariate network). This is treated in Section 6.7.

5.3 Further data specification options

5.3.1 Structurally determined values

It is allowed that some of the values in the digraph are structurally determined, i.e., deterministic rather than random. This is analogous to the phenomenon of 'structural zeros' in contingency tables, but in RSiena not only structural zeros but also structural ones are allowed. A structural zero means that it is certain that there is no tie from actor i to actor j; a structural one means that it is certain that there is a tie. This can be, e.g., because the tie is impossible or formally imposed, respectively.

For estimation by the Method of Moments, structural zeros provide an easy way to deal with actors leaving or joining the network between the start and the end of the observations: specify all their incoming and outgoing tie variables, at the moment that they are not present, as structural zeros. Note that actors having all values specified as structural zeros in this way take part of the simulations only starting at the observation moment where they are not totally structurally zero; therefore, this way of representing partially absent actors is not meaningful for actors who are

present only at the very last wave. In particular, this includes the case where there are two waves only for actors who join the network after the first wave.

Another way (more complicated but more flexible, because it gives possibilities to represent actors entering or leaving at specified moments between observations) is the method of joiners and leavers, described in Section 5.3.3. For actors present only at the last wave, the method of joiners and leavers is preferable.

When endowment or creation effects are to be included in the model specification, changing structural values should not be used, and the method of joiners and leavers then also is preferable.

It should be noted that if the model also contains behavioral dependent variables and structural zeros are used to represent changing composition of the network (or networks), the behavioral dependent variables still will be possible to change. To keep them from changing for the absent actors, a non-centered dummy actor variable can be used which is 1 for the absent actors and 0 for all others, and which is given a fixed `RateX` effect for the behavioral variables of a large negative value, e.g., –100; this effect should be fixed and not tested.

Structurally determined values are defined by reserved codes in the input data: the value 10 indicates a structural zero, the value 11 indicates a structural one. Structurally determined values can be different for the different time points. (The diagonal of the data matrix for a one-mode network always is composed of structural zeros, but this does not have to be indicated in the data matrix by special codes.) The correct definition of the structurally determined values can be checked from the brief report of this in the output file of write_report — which is given only, however, if the diagonal entries of the one-mode networks are also set to 10.

If there are a lot of structurally determined values then unconditional estimation (see Section 7.12.1) is preferable.

When RSiena simulates networks including some structurally determined values, if these values are constant across all observations then the simulated tie values are likewise constant. If the structural fixation varies over time, the situation is more complicated. Consider the case of two consecutive observations m and $m+1$, and let X_{ij}^{sim} be the simulated value at the end of the period from t_m to t_{m+1}. If the tie variable X_{ij} is structurally fixed at time t_m at a value $x_{ij}(t_m)$, then X_{ij}^{sim} also is equal to $x_{ij}(t_m)$, independently of whether this tie variable is structurally fixed at time t_{m+1} at the same or a different value or not at all. This is the direct consequence of the structural fixation. On the other hand, the following rule is also used. If X_{ij} is *not* structurally fixed at time t_m but it is structurally fixed at time t_{m+1} at some value $x_{ij}(t_{m+1})$, then in the course of the simulation process from t_m to t_{m+1} this tie variable can be changed as part of the process in the usual way, but after the simulation is over and before the statistics are calculated it will be fixed to the value $x_{ij}(t_{m+1})$.

The target values for the algorithm of the Method of Moments estimation procedure are calculated for all observed digraphs $x(t_{m+1})$. However, for tie variables X_{ij} that are structurally fixed at time t_m, the observed value $x_{ij}(t_{m+1})$ is replaced by the structurally fixed value $x_{ij}(t_m)$. This gives the best possible correspondence between target values and simulated values in the case of changing structural fixation.

For estimation by Maximum Likelihood, treated in Section 7.11, structural zeros operate differently than for the Method of Moments, and probably it is not a good idea to use changing structural zeros; see Section 7.11.

Structural zeros offer the possibility of analyzing several networks simultaneously under the assumption that the parameters are identical. However, a preferable option to do this is the multi-group option of Section 12.

The following explanation of the option of using structural zeros to analyze several networks simultaneously is given mainly for historical reasons (to understand earlier publications); this is not a recommended option.

E.g., if there are three networks with 12, 20 and 15 actors, respectively, then these can be integrated into one network of 12 + 20 + 15 = 47 actors, by specifying that ties between actors in different networks are structurally impossible. This means that the three adjacency matrices are combined in one 47×47 data matrix, with values 10 for all entries that refer to the tie from an actor in one network to an actor in a different network. In other words, the adjacency matrices will be composed of three diagonal blocks, and the off-diagonal blocks will have all entries equal to 10. In this example, the number of actors per network (12 to 20) is rather small to obtain good parameter estimates, but if the additional assumption of identical parameter values for the three networks is reasonable, then the combined analysis may give good estimates.

In such a case where K networks (in the preceding paragraph, the example had $K = 3$) are combined artificially into one bigger network, it will often be helpful to define $K - 1$ dummy variables at the actor level to distinguish between the K components. These dummy variables can be given effects in the rate function and in the evaluation function (for 'ego"), which then will represent that the rate of change and the out-degree effect are different between the components, while all other parameters are the same.

It will be automatically discovered by RSiena when monadic covariates depend only on these components defined by structural zeros, between which tie values are not allowed. For such variables, only the ego effects are defined and not the other effects defined for the regular actor covariates and described in Section 6.4. This is because the other effects then are meaningless. If at least one case is missing, then the other covariate effects are made available.

5.3.2 Missing data

RSiena allows that there are some missing data on network variables, on covariates, and on dependent action variables. Missing data must be indicated by the usual missing data code for R, `NA`.

Missingness of data is treated as non-informative. One should be aware that having many missing data can seriously impair the analyses: technically, because estimation will be less stable; substantively, because the assumption of non-informative missingness often is not quite justified. Up to 10% missing data will usually not give many difficulties or distortions, provided missingness is indeed non-informative (Huisman and Steglich, 2008). When one has more than 20% missing data on any variable, however, one may expect problems in getting good estimates.

In the current implementation of SIENA, missing data are treated in a simple way, trying to minimize their influence on the estimation results. This method is further explained in Huisman and Steglich (2008), where comparisons are also made with other ways of dealings with the missing information. The default method in RSiena for handling missings in the dependent variable is the fourth method as described in Huisman and Steglich (2008).

The basic idea is the following.

A brief sketch of the procedure is that missing values are imputed to allow meaningful simulations; for the calculation of the target statistics in the Method of Moments, tie variables and actor variables with missings are not used. More in detail, the procedure is as follows.

The simulations are carried out over all variables, as if they were complete. To allow this, missing data are imputed. In the initial observation, missing entries in the adjacency matrix are set to 0, i.e., it is assumed that there is *no* tie; this is done because normally data are sparse, so 'no tie' almost always is the modal value of the tie variable. In the further observations, for any tie variable, if there is an earlier observed value of this variable then the last observed value is used to impute the current value (the 'last observation carry forward' option, cf. Lepkowski (1989)); if there is no earlier observed value, the value 0 is imputed. For the dependent behavior variables a similar principle is used: if there is a previous observation of the same variable then this value is imputed, if there is none but there is a next observation then this is imputed, if this also is absent then the observationwise mode of the variable is imputed. Missing covariate data are, by default, replaced by the variable's global mean; but in the definition of actor covariates, by the functions as_covariate_rsiena or as_covariate_rsiena, the user has the option to supply other values for imputation of the missings. In the course of the simulations, however, the imputed values of the dependent behavior variables and of the network variables are allowed to change.

In order to ensure a minimal impact of missing data treatment on the results of parameter estimation (Method of Moments estimation) and/or simulation runs, the calculation of the target statistics used for estimation by the Method of Moments, and reporting in these procedures uses only non-missing data. When for an actor in a given period, any variable is missing that is required for calculating a contribution to such a statistic, this actor in this period does not contribute to the statistic in question. For network and dependent behavior variables, the tie variable or the actor variable, respectively, must provide valid data both at the beginning and at the end of a period for being counted in the respective statistics. This is implemented as follows: if a tie variable is missing in wave m, for the calculation of observed as well as simulated target statistics for periods $m-1$, m, and $m+1$ it is replaced by the value 0; if a dependent behavior variable is missing in wave m, for the calculation of observed as well as simulated target statistics, where always the centered values are used, for periods $m-1$, m, and $m+1$ the value is replaced by 0 (which, given the centering, is equivalent to the overall mean).

For non-centered covariates, the treatment of missing data is less well thought out; this may cause problems in the analysis of data with a lot of missing values for non-centered covariates.

By using the argument `imputationValues` in as_covariate_rsiena and as_covariate_rsiena, other values (i.e., values different from the mean that is used by default for imputation) can be given for imputation of missings in monadic covariates. These are then used for the simulations; since they were indicated as missings (`NA`) in the data themselves, they will not be used for the calculation of target statistics in the Method of Moments.

For estimation by Maximum Likelihood, missing values in the dependent variables at the end of a period are treated in a model-based way. For missings at the start of a period, independent priors are used. However, periods are treated separately: simulations in period m do not help provide information for period $m+1$. See `Siena_algorithms.pdf` for a further description.

The default method in RSiena for treating missing behavioral data in RSiena was compared with several alternative methods in Zandberg and Huisman (2019), and was found to perform quite well.

There is ongoing work by Robert Krause and others about using multiple imputation for RSiena modeling; see Krause et al. (2018). A script illustrating this for longitudinal network analysis is at the Siena scripts webpage, `http://www.stats.ox.ac.uk/~snijders/siena/multipleImputation_for_RSiena.R` Methods for co-evolution studies are in preparation.

5.3.3 Composition change: joiners and leavers

RSiena can also be used to analyze networks of which the composition changes over time, because actors join or leave the network between the observations. This can be done in two ways: using the method of Huisman and Snijders (2003), or using structural zeros. (For the maximum likelihood estimation option, the Huisman-Snijders method is not implemented, and only the structural zeros method can be used.) Structural zeros can specified for all elements of the tie variables toward and from actors who are absent at a given observation moment. How to do this is described in subsection 5.3.1. This is straightforward and not further explained here. This subsection explains the method of Huisman and Snijders (2003), also called the method of joiners and leavers, which uses the information about composition change in a somewhat more efficient way.

Network composition change, due to actors joining or leaving the network, is handled separately from the treatment of missing data. The data matrices must contain all actors who are part of the network at any observation time. If adjacency matrices are used as data input, they must therefore all have the same number of n rows, each actor having a separate (and fixed) line in these matrices, even for observation times where the actor is not a part of the network (e.g., when the actor did not yet join or the actor already left the network).

The *times of composition change* can be given either in a data file or in a list available in the R session. For networks with constant composition (no entering or leaving actors), this file or list is omitted and the current subsection can be disregarded.

If there is composition change, estimation by the Method of Moments is forced to be unconditional (see Section 7.12.1).

For these waves, where the actor is not in the network, the entries of the adjacency matrix can be specified in two ways. First as missing values using missing value code `NA`. In the estimation procedure, these missing values of the joiners before they joined the network are regarded as 0 entries, and the missing entries of the leavers after they left the network are fixed at the last observed values. This is different from the regular missing data treatment. Note that in the initial data description the missing values of the joiners and leavers are treated as regular missing observations. This will increase the fractions of missing data and influence the initial values of the density parameter.

A second way is by giving the entries a regular observed code, representing the absence or presence of a tie (as if the actor was a part of the network). In this case, additional information on relations between joiners and other actors in the network before joining, or leavers and other actors after leaving can be used if available. Note that this second option of specifying entries always supersedes the first specification: if a valid code number is specified this will always be

used.

The functions used to specify the times actors join or leave the network (i.e., the times of composition change) are as_composition_rsienaFromFile in case a file is used, and as_composition_rsiena in case a list is used. How to use a separate input file, called the *exogenous events file*, is described in the help page for as_composition_rsienaFromFile.

In the second case, a list must be given of length n, where n is the number of actors in the node set. The i'th element of this list must be a vector of numbers (characters are also allowed), composed of an even number of elements, indicating the intervals during which actor i was present. For example, `1 4` indicates that the actor was present from wave 1 to wave 4 (end points included) and `1 3.2 5.01 7` indicates that the actor was present from wave 1 to 20% of the time between waves 3 and 4, and then again from just after wave 5 to wave 7.

As an example, suppose we have 50 actors and 6 waves; almost all actors were present all the time, but actor 11 was present from wave 3 onward, actor 20 was present until wave 4, and actor 33 was present from mid-way between waves 1 and 2 until wave 3, and then again from just after wave 4 to wave 6. Then the list can be created by the following commands.

```
comp <- rep(list(c(1,6)), 50)
comp[[11]] <- c(3,6)
comp[[20]] <- c(1,4)
comp[[33]] <- c(1.5,3,  4.01,6)
changes <- as_composition_rsiena(comp)
```

(The use of blanks in the line for `comp[[33]]` is only for visually keeping the pairs of start-end times together.)

The first line, creating a list with the (default) first and last end point for everybody, could also be replaced by

```
comp <- vector("list", 50)
comp[] <- list(c(1,6))
```

Here it may be noted that `[]` keeps structures etc. unchanged while replicating the expression to fit.

The object `changes` created by the functions as_composition_rsienaFromFile or as_composition_rsiena is of class compositionChange and can be used in the function make_data_rsiena.

The method of joiners and leavers for representing composition change does not combine properly with the test_gof function (Section 6.15). But if you use `NA` codes for the tie variables of absent actors, and these are indicated as such by a compositionChange object, then test_gof will work properly.

The implementation of endowment and creation effects does not agree with changing structural zeros or ones. Therefore, if there is changing composition in a data set and the model should contain endowment and/or creation effects, the changing composition should be reflected by a changing composition and not by structural zeros.

6 Model specification

6.1 Definition of the model

After defining the data, the next step is to specify a model. The model specification consists of a selection of 'effects' for the evolution of each dependent variable (network or behavior). To understand this, first a brief review of the definition of the actor-oriented model is given (for further explanations see Snijders, 2001, 2005; Snijders et al., 2007, 2010b).

The model is based on four functions, which first are explained in an intuitive way. They are defined specifically for all dependent variables (network, behavior, or more of these if included in the model). These functions depend on the actor (hence the name 'actor-oriented') and on the state of the network, behavior, and covariates. All these functions are constituted by a weighted sum of so-called *effects*, which define the characteristics of the network (and behavior, if this is included as a dependent variable) that determine the probabilities of changes.

- *rate function*
  The rate function models the speed by which the dependent variable changes; more precisely: the speed by which each network actor gets an opportunity for changing her score on the dependent variable.
  Advice: in most cases, start modeling with a constant rate function without additional rate function effects.

  When there are important differences between actors in some measure of size or importance, it is possible that different advice must be given. This can be the case, e.g., when the network has a clear core-periphery structure; or when the outdegree distribution is very skewed. Then it may be necessary to let the rate function depend on the outdegrees or on an individual covariate indicating this size.

- *evaluation function*
  The evaluation function[4] is the primary determinant of the probabilities of changes. Probabilities are higher for moving towards states with a higher value of the evaluation function. One way of representing this is that the evaluation function models the actor's 'satisfaction'[5] with her/his local network neighborhood configuration. It is assumed that actors change their scores on the dependent variable such that they improve their total satisfaction – with a random element to represent the limited predictability of behavior. In contrast to the creation and endowment functions (described below), the evaluation function evaluates only the local network neighborhood configuration that results from the change under consideration, without considering 'where you come from'. In most applications, the evaluation function will be the main focus of model selection.

[4] The evaluation function was called *objective function* in Snijders (2001).

[5] The term 'satisfaction' should be interpreted here in a very loose sense; the satisfaction interpretation is not necessary at all, but it does give a convenient intuitive way of thinking about the model.

- *creation function*
  The creation function[6] distinguishes between new and old network ties (when evaluating possible network changes) and between increasing or decreasing behavioral scores (when evaluating possible behavioral changes). It is a component of the probabilities of change only for changes in an upward direction: creation of new ties, augmentation of values of the behavior dependent variable. Creation effects can be the creation parts of an evaluation effect, or elementary effects (see below).

  In the interpretation using satisfaction, the creation function models the gain in satisfaction incurred when network ties are created or behavioral scores are increased.

- *endowment or maintenance function*
  The endowment function[7], which also may be called *maintenance function*, also distinguishes between new and old network ties (when evaluating possible network changes) and between increasing or decreasing behavioral scores (when evaluating possible behavioral changes). It is a component of the probabilities of change only for changes in a downward direction: maintenance vs. termination of existing ties, decrease of values of the behavior dependent variable.

  Again, endowment effects can be the maintenance parts of an evaluation effect, or elementary effects (see below).

  In the interpretation using satisfaction, the endowment function models the loss in satisfaction incurred when network ties are dissolved or behavioral scores are decreased (hence the label 'endowment').

Leaving aside the rate effects, a given effect can normally be included in the model in any of the three 'types' or 'roles' of evaluation, creation, or endowment effect. In almost all cases, the advice is to start modeling without any creation or endowment effects, and add them perhaps at a later stage. For example, if the network dynamics in a given data set is such that ties mainly are created, and they are dissolved rather rarely, then the data will contain little information about the question whether creating ties follows different rules than dissolving ties, and if one would try to include creation or endowment effects for effects already included in the evaluation function, this would lead to large standard errors. Creation and endowment effects for behavior for behavior variables with more than 2 values are still under investigation, and their interpretation for practical research still is uncertain.

A model specification with only evaluation effects and without creation and endowment effects leads to exactly the same network dynamics as a specification where these effects are turned into creation and endowment effects, if the same parameters would have exactly the same values. For any given effect, it makes no sense to include the effect in all three roles: evaluation, creation, endowment. If one wishes to go beyond evaluation effects, then the user has to choose between adding an effect in either the creation or the endowment role.

For any effect, the model specifications of (evaluation & creation), (evaluation & endowment), and (creation & endowment) are equivalent, and pairs of parameters can be directly transformed

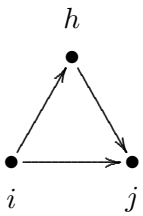
[6] A special case of the *gratification function* in Snijders (2001).

[7] The endowment function also is a special case of the *gratification function* in Snijders (2001).

into each other (cf. Section 13.1.4), but the estimates by the Method of Moments will not have the same correspondence because different pairs of target statistics are used. Each of the pair specifications could be tried out, in principle. Perhaps the first two pair specifications, containing the evaluation effect and one of the other two, give the most precise estimates; but this is not necessarily always the case.

6.1.1 Elementary effects

Not all contributions to the probability of change can be written as the change in some basic function (evaluation function). Therefore we sometimes need to directly represent contributions to a tie change or behavior change, without invoking an evaluation function. This can be done by using elementary effects. (In Snijders (2001) this was called a gratification function; as a more neutral term, we now use the word 'elementary effects'.)



The basic example here is transitive closure, which can be represented by the tendency toward forming closed triplets as in this figure. When the focal actor is i, ties that lead to the closure are $i \to j$ and $i \to h$; but the first of these ties means the closing of a two-path $i \to h \to j$, while the second means forming a tie to an actor h who made the same outgoing choice to the third actor j, a sign of structural equivalence; so these are distinct processes. The evaluation effect corresponding to the tendency toward forming closed triples is the `transTrip` effect, which is composed of the two distinct elementary effects `transTrip1`, contributing to creating or maintaining the $i \to j$ tie, and `transTrip2`, contributing to the $i \to h$ tie; see Section 13.

An elementary effect is a contribution to the creation or maintenance of a tie, defined directly, i.e., without expressing it based on the change in some evaluation function. This means that elementary effects are more general than evaluation effects, and all effects could be represented as elementary effects. For the sake of interpretation, however, the evaluation function formulation is used whenever possible.

Elementary effects can apply similarly to the creation and maintenance of a tie; or they can apply exclusively to tie creation, or exclusively to tie maintenance. In RSiena the difference between elementary effects and evaluation effects is only in the internal programming code, and the possible values of the `type` of effect as specified in the effects object and the functions set_effect and set_effect are only `eval`, `creation`, and `endow`. In Chapter 13 almost all effects are evaluation effects, and the effects that are elementary (and not evaluation) effects are mentioned as such.

6.1.2 Specification in RSiena

The model specification is defined in RSiena by the so-called *effects object*, which formally is an object of class sienaEffects or, for multiple groups as discussed in Chapter 12 of class sienaGroupEffects. It is a special kind of data.frame. The effects object is originally created by the function make_specification and subsequently modified by functions set_effect, set_effects, and others. The scripts on the SIENA website give examples. An important ingredient here is the so-called shortName of each effect, a column of the data frame, which is used to identify it.

For the practical use of SIENA, the shortNames are very important. Effects of covariates (and of other dependent variables used in the role of explanatory variable for the effect in question) need, in addition, the name of the covariate because the shortName does not specify the covariate. If there are several dependent variables (networks and/or behavioral variables), the name of the dependent variable (depvar) also is required to specify the effect, and will be given in the column covar1 (etc.) of the effects object.

The distinction between evaluation, endowment, and creation effects is specified by the type of the effect; this can be `eval`, `endow`, or `creation`. The default is `eval`. Only for rate effects this distinction does not exist; their type is `rate`. The type can be set in the functions set_effect, set_effect, and set_interaction.

A list of all effects available in the current version of RSiena with their shortNames can be displayed in a browser by using the function:

```
effectsDocumentation()
```

A list of all effects available for the given effects object, e.g., `myeff` – which depends on the set of variables and the number of periods in the data object – can be displayed in a browser by requesting

```
effectsDocumentation(myeff)
```

For example, the command

```
cbind(myeff$effectName, myeff$type, myeff$shortName)[1:20,]
```

gives a list of the first 20 effects in the `myeff` object. As another example,

```
cbind(myeff$effectName, myeff$type, myeff$shortName)[myeff$type == "eval",]
```

lists all evaluation effects in `myeff`, and

```
unique(myeff$shortName)
```

gives the set of all shortNames.

6.1.3 Mathematical specification

To attach precise meaning to the intuitive explanations above, the mathematical definition of the model is given as follows. To keep notation simple, we leave all statistical parameters out of the formulae. To keep the section short, we do not give a lot of explanation, but refer to the mentioned literature for that purpose.

As explained in Snijders et al. (2010b), the model is a continuous-time Markov chain, and represents how the network (and behavior) has changed in small steps (the so-called *ministeps*) from one observed to a later observed value. Each ministep entails a change in only one tie value, or one behavioral variable, and is modeled as follows.

First consider the network dynamics. At any given moment, let the network be denoted x^0. The rate function for actor i is denoted $\lambda_i(x)$; the evaluation function is $f_i(x)$; the creation function is $c_i(x)$; and the endowment function is $e_i(x)$.

At any given moment, let the current network be denoted x^0. The time duration until the next opportunity of change is exponentially distributed with parameter

$$\lambda_+(x^0) \;=\; \sum_i \lambda_i(x^0)\;.$$

This means that the expected time duration is

$$\frac{1}{\lambda_+(x^0)}\;.$$

The probability that actor i will be the next to have an opportunity for change is

$$\frac{\lambda_i(x^0)}{\lambda_+(x^0)}\;.$$

Now suppose that actor i is the one who has the next opportunity for change; one could say, this is the focal actor. Actor i then has the possibility to change one network tie, or to keep the network as it is. Denote by $\mathcal{C}$ the set of all networks that can be obtained as a result. Then the probability of the network obtained from this step depends on something called the objective function $u_i(x^0, x)$ which will be defined in a moment. Let $x \in \mathcal{C}$ be some network that could be obtained as a result of the ministep; then x will be the same as x^0, except perhaps for the tie $i \to j$ for some j; this tie might exist in x but not in x^0, or vice versa. The probability that the next network is x is given by

$$\frac{\exp\big(u_i(x^0, x)\big)}{\sum_{x' \in \mathcal{C}} \exp\big(u_i(x^0, x')\big)}\;. \tag{3}$$

The numerator is required to make all probabilities for this step sum to 1.

The objective function is defined as follows. If there is only an evaluation function (mathematically, this means that the creation and endowment functions are 0), then the objective function is equal to the evaluation function for the new state,

$$u_i(x^0, x) \;=\; f_i(x)\;.$$

Because of the properties of the exponential function one can just as well define the objective function as the gain in evaluation function,

$$u_i(x^0, x) \;=\; f_i(x) - f_i(x^0)\;. \tag{4}$$

To define the general case, note that if x^0 and x are not the same, then they differ in only one tie variable x_{ij}. Define $\Delta^+(x^0, x) = 1$ if x has one tie more than x^0, meaning that a tie is created by this change, and $\Delta^+(x^0, x) = 0$ otherwise. Similarly, define $\Delta^-(x^0, x) = 1$ if x has one tie less than x^0, meaning that a tie is dissolved by this change, and $\Delta^-(x^0, x) = 0$ otherwise. Then the general definition of the objective function is

$$\begin{aligned} u_i(x^0, x) \;=& \big(f_i(x) - f_i(x^0)\big) \\ &+\; \Delta^+(x^0, x)\,\big(c_i(x) - c_i(x^0)\big) \;+\; \Delta^-(x^0, x)\,\big(e_i(x) - e_i(x^0)\big)\;. \end{aligned} \tag{5}$$

This shows that the change in creation function plays a role only if a tie is created ($\Delta^+(x^0, x) = 1$), and the change in endowment function plays a role only if a tie is dissolved ($\Delta^-(x^0, x) = 1$).

If also elementary effects are included, then denote the linear combination for a tie variable x_{ij} for general (evaluation-type) elementary effects by $f_{ij}^{\mathrm{el}}(x)$, for creation elementary effects by $c_{ij}^{\mathrm{el}}(x)$, and for endowment elementary effects by $e_{ij}^{\mathrm{el}}(x)$. To the objective function $u_i(x^0, x)$ we then still have to add

$$f_{ij}^{\mathrm{el}}(x) \; + \; \Delta^+(x^0, x)\, c_{ij}^{\mathrm{el}}(x) \; + \; \Delta^-(x^0, x)\, e_{ij}^{\mathrm{el}}(x) \; .$$

For behavior dynamics the definitions are analogous. Here a basic assumption is that, when there is an opportunity for change, the possible new values for the behavior variable are the current value, this value +1, and this value –1, as long as these changes do not take the value out of the permitted range. More elaborate explanations are in (Snijders et al., 2007, 2010b; Steglich et al., 2010; Veenstra et al., 2013).

The evaluation, creation, and endowment functions are constructed as a linear combination of the effects:

$$f_i(x) \; = \; \sum_k \beta_k \, s_{ik}(x) \tag{6}$$

where β_k are parameters and $s_{ik}(x)$ are effects. For models with more than one dependent variable (networks, behaviors), this is specific to each dependent variable; e.g., in a network and behavior study with one network and one behavioral variable, there will be two evaluation functions. Because of formula (4), what is essential for the effects for networks is the difference

$$\Delta_{kij}(x) \; = \; s_{ki}\big(x^{+ij}\big) \; - \; s_{ki}\big(x^{-ij}\big) \tag{7}$$

where x^{+ij} is network x with the tie $i \to j$ and x^{-ij} is network x without this tie. This is called the *change statistic*.

Elementary effects are defined by

$$s_{ijk}^{\mathrm{el}}(x) \; = \; x_{ij} \, s_{ijk}^{\mathrm{el0}}(x) \; ,$$

for a statistic $s_{ijk}^{\mathrm{el0}}(x)$ which does not depend on x_{ij}. Therefore, the corresponding change statistic is

$$\Delta_{kij}^{\mathrm{el}}(x) \; = \; s_{ijk}^{\mathrm{el0}}(x) \; .$$

6.2 Important structural effects for network dynamics: one-mode networks

Advice for model specification for one-mode networks is given also in the slide set *Model Specification Recommendations for Siena*, available as
https://www.stats.ox.ac.uk/~snijders/siena/Siena_ModelSpec_s.pdf

For the structural part of the model for network dynamics, for one-mode (or unipartite) networks, the most important effects are as follows. The mathematical formulae for these and other effects are given in Chapter 13. Here we give a more qualitative description.

A default model choice could consist of (1) the out-degree (density) and reciprocity effects; (2) one network closure effect, e.g. transitive triplets, transitive ties, or gwesp; the transitive reciprocated triplets effect and/or the 3-cycles effect; (3) the in-degree popularity effect (raw or square root version); the out-degree activity effect (raw or square root version); and either the in-degree activity effect or the out-degree popularity effect (raw or square root function). The two effects (1) are so basic they cannot be left out. The effects selected under (2) represent the dynamics in local (triadic) structure (also see Block, 2015, for the transitive reciprocated triplets effect); and the three effects selected under (3) represent the dynamics in in- and out-degrees (the first for the dispersion of in-degrees, the second for the dispersion of out-degrees, and the third for the covariance between in- and out-degrees) and also should offer some protection, albeit imperfect, for potential ego- and alter-effects of omitted actor-level variables.

Information about basic contact opportunities, if available, should also be taken into account. This can be, e.g., membership of the same classroom or department, or some measure of proximity used as a dyadic covariate. Especially for large networks (more than 100 nodes) this is important.

The basic list of these and other effects is as follows.

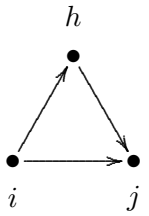
1. The *out-degree (density) effect* which always must be included, to give the balance between creation and termination of ties (conditionally on all other effects), and has the role of an intercept.

2. The *reciprocity effect* which practically always must be included.

3. There is a choice among several network closure effects. Usually it will be sufficient to express the tendency to network closure by including one or two of these. They can be selected by theoretical considerations and/or by their empirical statistical significance and contribution to a good fit.

   In older publications, when the gwesp effect was not available, it was sometimes advised to include the transitive triplets together with the transitive ties effect. Currently, the impression is that often the gwesp effect is at least as good as a combination of transitive triplets and transitive ties. For many data sets, a model containing either the transitive triplets effect or the gwesp effect (not both) is adequate to represent transitivity, and the user may try out which of these yields the best fit.

   a. The *transitive triplets effect*, which is the classical representation of network closure by the number of transitive triplets. For this effect the contribution of the tie $i \to j$ is proportional to the total number of transitive triplets that it forms – which can be transitive triplets of the type $\{i \to j \to h;\ i \to h\}$ as well as $\{i \to h \to j;\ i \to j\}$;

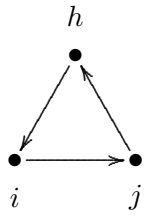


   b. The *transitive ties effect* is similar to the transitive triplets effect, but instead of considering for each other actor j how many two-paths $i \to h \to j$ there are, it is only considered whether there is at least one such indirect connection. Thus, one indirect tie suffices for the network embeddedness.

c. The *gwesp* effect; see later in this manual. Figure 4 shows that the contribution of the number of two-paths for the gwesp effect is between that for the transitive ties effect ($\alpha = 0$) and for the transitive triplets effect ($\alpha = \infty$). The default for gwesp is $\alpha = 0.69$, corresponding to an internal effect parameter of 69 (see Section 13.0.1 for internal effect parameters).

d. There are various other effects that may express network closure, such as the *balance* effect, the *Jaccard similarity effects* for incoming and for outgoing ties, and the *number of actors at distance two effect*. The balance and Jaccard similarity effects are ways to express tendencies to structural equivalence. The number of actors at distance two effect expresses network closure inversely: stronger network closure (when the total number of ties is fixed) will lead to fewer geodesic distances equal to 2. When this effect has a negative parameter, actors will have a preference for having few others at a geodesic distance of 2 (given their out-degree, which is the number of others at distance 1); this is one of the ways for expressing network closure. However, it is theoretically ambiguous, because the number of others at a geodesic distance of 2 can be decreased in two quite different ways: dropping ties to those who have many connections 'further out', or making ties to those who have many connections to others to whom one is already directly connected.

4. The *three-cycles effect*, which can be regarded as generalized reciprocity (in an exchange interpretation of the network) but also as the opposite of hierarchy (in a partial order interpretation of the network). A negative three-cycles effect, together with a positive transitive triplets or transitive ties effect, may be interpreted as a tendency toward local hierarchy. The three-cycles effect also contributes to network closure.



However, Block (2015) has argued convincingly that instead of the three-cycles effect, it is often advisable to use the transitive reciprocated triplets effect.
For models where the gwesp effect is used instead of the transitive ties effect, the interaction gwesp×reciprocity can fulfill the role of the transitive reciprocated triplets effect.

In a non-directed network, the three-cycles effect is identical to the transitive triplets effect.

5. Another triadic effect is the *betweenness effect*, which represents brokerage: the tendency for actors to position themselves between not directly connected others, i.e., a preference of i for ties $i \to j$ to those j for which there are many h with $h \to i$ and $h \not\to j$. This can, however, not be combined with the transitive triplets effect because it is a different mechanism leading to the same structures.
The betweenness effect has the property that it is defined using configurations characterized by the absence of certain ties, not only by their presence. This leads to difficulties in interpretation, and may be undesirable. Therefore this effect should be used only with caution, in cases where it is important for the specific research question.

⊙ The following nine degree-related effects may be important especially for networks where degrees are theoretically important and represent social status or other features important for network dynamics; and/or for networks with high dispersion in in- or out-degrees (which may be an empirical reflection of the theoretical importance of the degrees).
The out-degree popularity and in-degree activity effects are closely related; their raw (not-sqrt) versions are exactly collinear in the Method of Moments.
Degree assortativity are higher-order effects, and will be included mainly if there are strong theoretical reasons for them.

6. The *in-degree popularity effect* (again, with or without 'sqrt', with the same considerations applying) reflects tendencies to dispersion in in-degrees of the actors; or, tendencies for actors with high in-degrees to attract extra incoming ties 'because' of their high current in-degrees.

7. The *out-degree popularity effect* (again, with or without 'sqrt', with the same considerations applying) reflects tendencies for actors with high out-degrees to attract extra incoming ties 'because' of their high current out-degrees. This leads to a higher correlation between in-degrees and out-degrees.

8. The *in-degree activity effect* (with or without 'sqrt') reflects tendencies for actors with high in-degrees to send out extra outgoing ties 'because' of their high current in-degrees. This leads to a higher correlation between in-degrees and out-degrees. The in-degree activity and out-degree popularity effects are not distinguishable in Method of Moments estimation; then the choice between them must be made on theoretical grounds.

9. The *out-degree activity effect* (with or without 'sqrt') reflects tendencies for actors with high out-degrees to send out extra outgoing ties 'because' of their high current out-degrees. This also leads to dispersion in out-degrees of the actors.

10. The *reciprocal degree activity effect* (where internal effect parameter with value 2 will be the 'sqrt' version; see Section 13.0.1 for internal effect parameters). This reflects tendencies for actors with high reciprocal degrees (number of other actors to whom the actor has incoming as well as outgoing ties) to send out extra outgoing ties 'because' of their high current reciprocal degrees. For this effect, usually negative parameters β_k are expected ('fewer' instead of 'extra').

11. The *in-in degree assortativity effect* (where parameter 2 is the same as the sqrt version, while parameter 1 is the non-sqrt version) reflects tendencies for actors with high in-degrees to preferably be tied to other actors with high in-degrees.

12. The *in-out degree assortativity effect* (with parameters 2 or 1 in similar roles) reflects tendencies for actors with high in-degrees to preferably be tied to other actors with high out-degrees.

13. The *out-in degree assortativity effect* (with parameters 2 or 1 in similar roles) reflects tendencies for actors with high out-degrees to preferably be tied to other actors with high in-degrees.

14. The *out-out degree assortativity effect* (with parameters 2 or 1 in similar roles) reflects tendencies for actors with high out-degrees to preferably be tied to other actors with high out-degrees.

6.2.1 Rate function

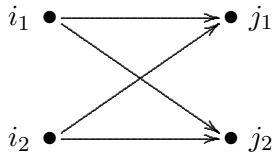
In some cases the rate function needs special attention. This can be the case, e.g., when the network has a clear core-periphery structure; or when the outdegree distribution is very skewed. Then it may be necessary to let the rate function depend on the outdegrees, using one of the three effects `outRateLog`, `outRateInv`, or `outRate`; or on an individual covariate indicating this size. Experience shows that often a rate dependence on outdegrees gives best results, and that `outRateLog` often is preferable. See Section 13.1.5.

6.3 Important structural effects for network dynamics: two-mode networks

The Stochastic Actor-Oriented Model for two-mode (or bipartite) networks is treated in Koskinen and Edling (2012). The co-evolution of one-mode and two-mode networks is treated in Snijders et al. (2013). The most important effects are as follows. The mathematical formulae for these and other effects are given in Chapter 13. Here we give a more qualitative description.

1. The *out-degree effect* which always must be included.

2. Transitivity in two-mode networks is expressed in the first place by the number of *four-cycles* (Robins and Alexander, 2004). This reflects the extent to which actors who make one choice in common also make other choices in common.



⊙ The following three degree-related effects may be important especially for networks where degrees are theoretically important and represent social status or other features important for network dynamics; and/or for networks with high dispersion in in- or out-degrees (which may be an empirical reflection of the theoretical importance of the degrees). Include them if there are theoretical reasons for doing so, but only in such cases.

3. The *out-degree activity effect* (with or without 'sqrt'; often the sqrt version, which transforms the degrees in the explanatory role by the square root, works better) reflects tendencies to dispersion in out-degrees of the actors.

4. The *in-degree popularity effect* (again, with or without 'sqrt', with the same considerations applying) reflects tendencies to dispersion in in-degrees of the column units.

5. The *out-in degree assortativity effect* (where parameter 2 is the same as the sqrt version, while parameter 1 is the non-sqrt version) reflects tendencies for actors with high out-degrees to preferably be tied to column units with high in-degrees.

6.4 Effects for network dynamics associated with covariates

For each individual covariate, there are several effects which can be included in a model specification, both in the network evolution part and in the behavioral evolution part (should there be dependent behavior variables in the data). Of course for two-mode networks, the covariates must be compatible with the network with respect to number of units (rows/columns).

- *network rate function*

    1. the covariate's effect on the rate of network change of the actor;

- *network evaluation, creation, and endowment functions*

    1. the covariate-similarity effect, which is suitable for variables measured on an interval scale (or at least an ordinal scale where it is meaningful to use the absolute difference between the numerical values to express dissimilarity); a positive parameter implies that actors prefer ties to others with similar values on this variable – thus contributing to the network-autocorrelation of this variable not by changing the variable but by changing the network;
       for categorical variables, see the 'same covariate' effect below;
    2. the effect on the actor's activity (covariate-ego); a positive parameter will imply the tendency that actors with higher values on this covariate increase their out-degrees more rapidly;
    3. the effect on the actor's popularity to other actors (covariate-alter); a positive parameter will imply the tendency that the in-degrees of actors with higher values on this covariate increase more rapidly;
    4. the effect of the squared variable on the actor's popularity to other actors (squared covariate-alter) (included only if the range of the variable is at least 2). This normally makes sense only if the covariate-alter effect itself also is included in the model. A negative parameter implies a unimodal preference function with respect to alters' values on this covariate;
    5. the interaction between the value of the covariate of ego and of the other actor (covariate ego $\times$ covariate alter); a positive effect here means, just like a positive similarity effect, that actors with a higher value on the covariate will prefer ties to others who likewise have a relatively high value; when used together with the alter effect of the squared variable this effect is quite analogous to the similarity effect, and for dichotomous covariates, in models where the ego and alter effects are also included, it even is equivalent to the similarity effect (although expressed differently), and then the squared alter effect is superfluous;
    6. the 'same covariate', or covariate identity, effect, which expresses the tendency of the actors to be tied to others with exactly the same value on the covariate; whereas the preceding four effects are appropriate for interval scaled covariates (and mostly also for ordinal variables), the identity effect is suitable for categorical variables;
    7. the interaction effect of covariate-similarity with reciprocity;

8. the effect of the covariate of those to whom the actor isf indirectly connected, i.e., through one intermediary but not with a direct tie; this value-at-a-distance can represent effects of indirectly accessed social capital.

The usual order of importance of these covariate effects on network evolution is: evaluation effects are most important, followed by creation, endowment and rate effects. Inside the group of evaluation effects, for variables measured on an interval scale (or ordinal scale with reasonable numerical values), it is the covariate-similarity effect that is most important, followed by the effects of covariate-ego and covariate-alter.

When the network dynamics is not smooth over the observation waves — meaning that the pattern of ties created and terminated, as reported in the initial part of the output file produced by write_report under the heading *Initial data description – Change in networks – Tie changes between subsequent observations*, is very irregular across the observation periods — it can be important to include effects of time variables on the network. Time variables are changing actor covariates that depend only on the observation number and not on the actors. E.g., they could be dummy variables, being 1 for one or some observations, and 0 for the other observations.

For actor covariates that have the same value for all actors within observation waves, or – in the case that there are structurally determined values – that are constant for all actors within the same connected components, only the ego effects are defined, because only those effects are meaningful. This exclusion of the alter, similarity and other effects for such actor variables applies only to variables without any missing values.

For each dyadic covariate, the following network evaluation effects can be included in the model for network evolution:

- *network evaluation, creation, and endowment functions*

  1. main effect of the dyadic covariate;
  2. the interaction effect of the dyadic covariate with reciprocity.

The main evaluation effect is usually the most important. But dyadic covariates can also be transformed, e.g., to patterns of indirect connections, which then are again usable as dyadic covariates. Dyadic covariates can also be used in various ways as weights for behavioral evolution.

6.5 Cross-network effects for dynamics of multiple networks

If there are multiple dependent network variables, these can be one-mode networks, two-mode networks, or a combination of these. The co-evolution of one-mode and two-mode networks is treated in Snijders et al. (2013), but this paper can also be used as an introduction to the dynamics of multiple one-mode networks. For multiple dependent network variables, the following effects may be important. This is explained here jointly for the case of one-mode and two-mode networks. The *number of columns* is defined as the number of actors for one-mode networks, and as the number of units/nodes/... in the second node set for two-mode networks. For cross-network effects the network in the role of dependent variable is denoted by X and the network in the role of explanatory variable by W; thus, effects go from W to X. All these effects are regarded as effects determining the dynamics of network X.

1. If both networks have the same number of columns, then the basic effect is the entrainment of X by W, i.e., the extent to which the existence of a tie $i \stackrel{W}{\rightarrow} j$ promotes the creation or maintenance of a tie $i \stackrel{X}{\rightarrow} j$.

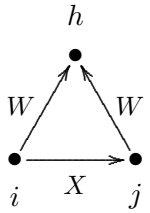
2. If both networks are one-mode, then a next effect is the reciprocity effect with W on X, representing the extent to which the existence of a tie $j \stackrel{W}{\rightarrow} i$ promotes the creation or maintenance of a tie, in the reverse direction, $i \stackrel{X}{\rightarrow} j$.

3. If both networks are one-mode, then a next effect is the mutuality effect with W on X, representing the extent to which the existence of a mutual tie $i \stackrel{W}{\leftrightarrow} j$ promotes the creation or maintenance of a tie $i \stackrel{X}{\rightarrow} j$.

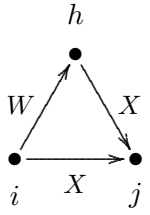
4. The *outdegree W activity effect* (where parameter 2 is the sqrt version, while parameter 1 is the non-sqrt version – see above for explanations of this) reflects the extent to which actors with high outdegrees on W will make more choices in the X network.

⊙ Several mixed transitivity effects can be important.

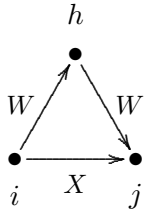
5. If X is a one-mode network, the *from W agreement* effect represents the extent to which agreement between i and j with respect to outgoing W-ties promotes the creation or maintenance of a tie $i \stackrel{X}{\rightarrow} j$.



6. If W is a one-mode network, the *W to agreement* effect represents the extent to which a W tie $i \stackrel{W}{\rightarrow} h$ leads to agreement between i and h with respect to outgoing X-ties to others, i.e., X-ties to the same third actors j, $i \stackrel{X}{\rightarrow} j$ and $h \stackrel{X}{\rightarrow} j$.



7. If X and W both are one-mode networks, the *closure of W* effect represents the tendency closure of $W - W$ two-paths $i \stackrel{W}{\rightarrow} h \stackrel{W}{\rightarrow} j$ by an X tie $i \stackrel{X}{\rightarrow} j$.



6.6 Hierarchy requirements

From linear regression we know that when a model contains an interaction effect, usually also the corresponding main effects should be included. Similarly for higher-order interactions, their lower-order terms will usually be included. This is called the *hierarchy* principle in model building.

The situation for structural effects in Stochastic Actor-Oriented Models is similar but more complex.

For covariate effects the considerations are similar to those in regression analysis and do not need reiteration. The hierarchy principle for the Stochastic Actor-Oriented Model is that when an effect represented by some graphical structure is included in the model for the purpose of testing and interpretation, the effects for the sub-structures that are part of the structure should usually also be included to avoid misinterpretation. In other words, testing of higher-order structures implies that the corresponding lower-order structures should be controlled for. This can be done by including the effects representing them in the model and estimating their parameters; or including them in the model with a parameter fixed at 0, tested by a score-type test (Section 9.2), and found to be non-significant.

An example is the transitive triplets effect *transTrip*.

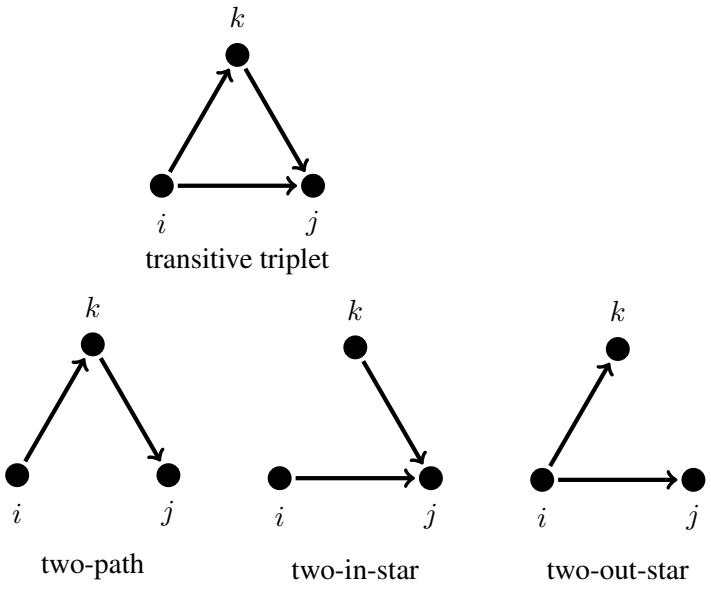


Figure 1: Transitive triplets and included subgraphs

In Figure 1 we see the transitive triplet together with the three two-arc configurations that it contains as subgraphs. These correspond, respectively, to the indegree-activity or outdegree-popularity effect, the indegree-popularity effect, and the outdegree-activity effect. To use the test of the transitive triplets effect as evidence for or against transitivity it is necessary that the model includes these three effects (for the first it does not matter whether *inAct* or *outPop* is chosen). Note the condition "*to use the test of the transitive triplets effect as evidence for or against transitivity*"; if the purpose is just to obtain a well-fitting model, then it might be superfluous to include all three lower-order effects.

This principle also applies to multivariate networks as discussed in the previous section.

6.7 Constraints between networks

When analysing multiple dependent networks, the following constraints between networks can be considered:

disjoint: for any given pair of actors (i, j), it is impossible that ties exists in both networks;

higher: the tie variables for the first network are higher than or equal to those for the second network: for any given pair of actors (i, j), it is impossible that a tie exists in the second but not in the first network;

atleastone: for any given pair of actors (i, j), there must be a tie in at least one of the networks.

Whether this is so, is indicated in the attributes of the data set (of class `siena`). These constraints are observed and set during the creation of the `siena` object in make_data_rsiena. When one of the constraints is true, this will be reported in the output of write_report. These constraints will continue to be respected during the simulations (and by implication for the estimations). The constraints for the simulations can be changed (e.g., turned off) by function make_constraint.

Script `http://www.stats.ox.ac.uk/~snijders/siena/RscriptSienaOrdered.R` gives a demonstration of how this feature can be used for analyzing networks with ordered values.

Constraints will work for all pairs of networks of the same kind: both networks directed one-mode, both symmetric (non-directed) one-mode, or both two-mode with the same node sets. For the combination of a one-mode directed and a symmetric non-directed network, the constraint **disjoint** is possible; and the constraint **higher** is possible, if the symmetric network is higher than the directed network.

Constraints do not work for multi-group data sets! By implication, they do not work for multi_siena.

6.8 Effects on behavior evolution

For models with one or more dependent behavior variables, i.e., models for the co-evolution of networks and behavior, the most important effects for the behavior dynamics are the following; see Steglich et al. (2010). In these descriptions, with the 'alters' of an actor we refer to the other actors to whom the focal actor has an outgoing tie. The dependent behavior variable is referred to as Z.

1. The shape effect, expressing the basic drive toward high values on Z. A zero value for the shape will imply a drift toward the midpoint of the range of the behavior variable.

2. The effect of the behavior Z on itself, or quadratic shape effect, which is relevant only if the number of behavioral categories is 3 or more. This can be interpreted as giving a quadratic preference function for the behavior. When the coefficient for the shape effect is β_1^Z and for the effect of Z on itself, or quadratic shape effect, is β_2^Z, then the contributions of these two effects are jointly $\beta_1^Z\,(z_i - \bar{z}) + \beta_2^Z\,(z_i - \bar{z})^2$. With a negative coefficient β_2^Z, this is a unimodal preference function, with the maximum attained for $z_i = \bar{z} - 2\,\beta_1^Z/\beta_2^Z$. (Of

course additional effects will lead to a different picture; but as long as the additional effects are linear in z_i – which is not the case for similarity effects! –, this will change the location of the maximum but not the unimodal shape of the function.) This can also be regarded as negative feedback, or a self-correcting mechanism: when z_i increases, the further push toward higher values of z_i will become smaller and when z_i decreases, the further push toward lower values of z_i will become smaller. On the other hand, when the coefficient β_2^Z is positive, the feedback will be positive, so that changes in z_i are self-reinforcing. This can be an indication of addictive behavior.

3. The average similarity effect, expressing the preference of actors to being similar with respect to Z to their alters, where the total influence of the alters is the same regardless of the number of alters.

4. The total similarity effect, expressing the preference of actors to being similar to their alters, where the total influence of the alters is proportional to the number of alters.

5. The average alter effect, expressing that actors whose alters have a higher average value of the behavior Z, also have themselves a stronger tendency toward high values on the behavior.

6. The total alter effect, expressing that actors whose alters have a higher total value of the behavior Z, also have themselves a stronger tendency toward high values on the behavior.

7. The indegree effect, expressing that actors with a higher indegree (more 'popular' actors) have a stronger tendency toward high values on the behavior.

8. The outdegree effect, expressing that actors with a higher outdegree (more 'active' actors) have a stronger tendency toward high values on the behavior.

Effects 1 and 2 will practically always have to be included as control variables. (For dependent behavior variables with 2 categories, this applies only to effect 1.) When the behavior dynamics is not smooth over the observation waves — meaning that the pattern of steps up and down, as reported in the initial part of the output file produced by write_report under the heading *Initial data description – Dependent actor variables – Changes*, is very irregular across the observation periods — it can be important to include effects of time variables on the behavior. Time variables are changing actor covariates that depend only on the observation number and not on the actors. E.g., they could be dummy variables, being 1 for one or some observations, and 0 for the other observations.

The average similarity, total similarity, average alter, and total effects are different specifications of social influence. The choice between them will be made on theoretical grounds and/or on the basis of statistical significance. Do not include them all together in one model, as this would most likely lead to multicollinearity and non-convergence.

For each actor-dependent covariate as well as for each of the other dependent behavior variables, the effects on Z which can be included is the following.

1. The main effect: a positive value implies that actors with a higher value on the covariate will have a stronger tendency toward high Z values.

2. Various effects of the combination of covariate values for members of the personal network of the focal actor (outgoing ties, incoming ties, distance-two ties): search in this manual for `avXAlt, avXInAlt, avXAltDist2, avXInAltDist2` and their manifold variations.

3. Interactions between two or three actor variables, see Section 6.12.

6.9 Model Type: non-directed networks

Dynamics for non-directed networks is explained in Snijders and Pickup (2017).

RSiena detects automatically when the networks all are non-directed, and then employs a model for this special case. For non-directed networks, the Model Type has five possible values, as described in Snijders and Pickup (2017). This is specified by the parameter modelType in function set_model_saom. Value `modelType=1` is for directed networks, values `2-6` for non-directed networks.

1. Directed networks option `modelType=1` is not used for non-directed networks.

2. Forcing model, `modelType=2`:
   one actor takes the initiative and unilaterally imposes that a tie is created or dissolved.

3. Unilateral initiative and reciprocal confirmation, `modelType=3`:
   one actor takes the initiative and proposes a new tie or dissolves an existing tie; if the actor proposes a new tie, the other has to confirm, otherwise the tie is not created; for dissolution, confirmation is not required.
   For the confirmation decision, an offset is added to the objective function, representing the difference between a binary yes/no choice and the multinomial choice of one among $n-1$ tie variables to change. This offset is a fixed parameter, not estimated. The higher the offset, the more likely it is that the confirmation will be given. It is determined by the value of `Offset` in function set_model_saom. This parameter must be given as a named vector; see the help page for set_model_saom. Plausible values for this offset are in the range from 0 to 3, and should slowly increase with n (roughly proportionally to $\log(n)$).

4. Pairwise disjunctive (forcing) model, `modelType=4`:
   a pair of actors is chosen and reconsider whether a tie will exist between them; the tie will exist if at least one of them chooses for the tie, it will not exist if both do not want it.

5. Pairwise conjunctive model, `modelType=5`:
   a pair of actors is chosen and reconsider whether a tie will exist between them; the tie will exist if both agree, it will not exist if at least one does not choose for it.

6. Pairwise compensatory (additive) model, `modelType=6`:
   a pair of actors is chosen and reconsider whether a tie will exist between them; this is based on the sum of their objective functions for the existence of this tie.
   If the evaluation function is symmetric in ego and alter, and there are no creation and/or maintenance functions, this is the Longitudinal Exponential Random Graph Model ('LERGM') of Koskinen and Snijders (2013).

In the first two of these models, where the initiative is one-sided, the rate function is comparable to the rate function in directed models. In the last three models, however, the pair of actors is chosen at a rate which is the *product* of the rate functions λ_i and λ_j for the two actors. This means that opportunities for change of the single tie variable x_{ij} occur at the rate $\lambda_i \times \lambda_j$. The numerical interpretation is different from that in the first two models.

Some effects specially designed for non-directed networks are presented in Section 13.1.2.

6.10 Model Type: directed networks

As of version 1.3-22, there are additional model options for dynamics of directed one-mode networks, specified by the parameter modelType in function set_model_saom. The value `modelType=1` is the default for directed networks, values `modelType = 7, 8, 9, 10` are options included for the purpose of investigating the consequences of details of the specification of the ministep.

These values implement the so-called *double step*. The ministeps are explained above in Section 2.1 and more in detail in Section 6.1.3. A double step is defined by a ministep taken by some focal actor i; suppose this means the creation or termination of the tie $i \rightarrow j$ for some j; then a following ministep is taken by actor j. In these model specifications the 'pure' ministeps and the double steps are mixed randomly. For the respective values `modelType = 7, 8, 9, 10`, the probability that a double step is taken is, respectively, 0.25, 0.50, 0.75, and 1.00.

Since version 1.4.19, `modelType=11` can be used for dependent networks. This is called the *Contemporaneous evaluation statistics model*. This model type implies that, for co-evolution models for this dependent network, which may involve other dependent networks as well as dependent behavioral variables, all evaluation effects are estimated using simultaneous statistics for estimation by the Method of Moments (for `modelType=1`, the default, the MoM estimation statistics are cross-lagged). For this model type, creation and endowment effects are estimated as usual, i.e., using cross-lagged creation and maintenance statistics.

These model types are meant for research purposes, not intended for use in regular empirical applications.

6.11 Model Type: behavior

As of version 1.1-306, there are two model options for dynamics of behavior, specified by the parameter behModelType in function set_model_saom. They differ in the treatment of the boundaries of the range of the behavioral variable. Recall that the behavioral microsteps consist of the addition of –1, 0, or +1 to the dependent variable, under the condition of remaining within the integer range from the minimum z^- to the maximum z^+.

If the current value of the behavior is at the boundary, z^- or z^+, then only two possible outcomes remain. The default is the option available before, with `behmodelType=1`. This is also called the *restricting* model, because the probabilities of the two options are given by (3), where the set of possible new states $\mathcal{C}$ is just the restricted set $\{z^-, z^- + 1\}$ in the case of the minimum, and $\{z^+ - 1, z^+\}$ in the case of the maximum.

The second option, for `behmodelType=2`, is called the *absorbing* model, because the probabilities of the two options are calculated as if the variable could go outside of the range, but when

such an excursion occurs it is absorbed into the range again. Thus, probabilities are calculated according to (3), with $\mathcal{C} = \{z^- - 1, z^-, z^- + 1\}$ or $\mathcal{C} = \{z^+ - 1, z^+, z^+ + 1\}$, respectively; but the outcome $z^- - 1$ when it occurs is replaced by z^-, and $z^+ + 1$ when it occurs is replaced by z^+. The objective function for the virtual outcome $z^- - 1$ is taken to be the same as for z^-, and similarly for $z^+ + 1$ it is taken to be the same as for z^+.

6.12 Interaction effects

For the Stochastic Actor-Oriented Model and other network models, the definition of interaction effects is not evident. For example, what is the interaction of reciprocity and transitive triplets? One possibility is the transitive reciprocated triplets effect (`transRecTrip`) effect defined in Section 13.1.1. This definition makes perfect sense; nevertheless, the reciprocal tie could have been put at a different place than in dyad (i, j) in the picture there — which would have led to a different type of interaction 'reciprocity $\times$ transitivity'. Also for interactions between covariates and structural effects, when the structural effect is expressed by some configuration of ties, there will usually be several possibilities to choose the actor in the configuration for which the covariate should be considered, and these will often lead to different effects.

This leads to a distinction between ad hoc interaction effects and standard interaction effects. Ad hoc interaction effects have to be defined specifically in each case, like `transRecTrip`; Section 13.1.1 contains many other examples. Standard interaction effects can be defined directly. They are also called 'user-defined interaction effects'. The change statistic of a standard interaction effect is the product of the change statistics of the interacting effects. This does not always make sense; whether it is permitted is defined by the '`interactionType`' of the effects. This is elaborated for network effects in Section 6.12.3 and for behavior effects in Section 6.12.4.

Interactions can be specified by the function set_interaction. The basis of the implementation is provided by the definition, by SIENA, of 'unspecified interaction effects'. The interacting effects are given by the columns `effect1` and `effect2`, and for three-way interaction effects, `effect3`, in the effects object; they contain the `effectNumber` (sequence number in the effects object) of the effects that are interacting. The interaction effect must be 'included' to be part of the model, but the underlying effects need only be 'included' if they are also required individually. (In most cases this is advisable.) The number of possible user-defined interaction effects is limited, and is set in the call of make_specification.

The information necessary for working with interaction effects – the interaction types, short names, and sequence numbers of the effects – is contained in the document produced for a given effects object, say `myeff`, by the function call

```
effectsDocumentation(myeff)
```

Further see the help page for the function effectsDocumentation. Chapter 13 of this manual also gives the short names of all effects. The short name of all unspecified interaction effects is `unspInt` for network effects, `behUnspInt` for discrete behaviour effects, and `contUnspInt` for continuous behaviour effects.

6.12.1 Internal effect parameters for interaction effects

Internal effect parameters are treated in Section 13.0.1. Function set_effect has an argument `parameter` to set the internal effect parameter, but the function set_interaction does not have this argument. The reason is that the internal effect parameters for the interaction are borrowed from the main effects; even if these are not included in the model. Setting the internal effect parameters for main effects, without including them in the model, can be done by

```
set_effect(..., parameter=..., include=FALSE)
```

as is done in the following example:

```
myeff <- set_effect(myeff, outActIntnX, depvar="mynet2", covar1="mynet1",
           covar2="alc", parameter=1, include=FALSE)
myeff <- set_effect(myeff, divOutEgoIntn, depvar="mynet2", covar1="mynet1",
           parameter=1, include=FALSE)
(myeff <- set_interaction(myeff, list(outActIntnX, divOutEgoIntn),
       depvar="mynet2", covar1=c("mynet1", "mynet1"), covar2=c("alc", "")))
```

The main effects are defined by

$$\sum_j x_{ij} \sum_h w_{ih}\, v_h$$

and

$$\sum_j x_{ij}\, \text{divi}(1, w_{i+})$$

where $\text{divi}(a, b)$ is defined as a/b if $b \neq 0$ and 0 if $b = 0$; so the interaction effect included in this way is

$$\sum_j x_{ij} \left(\sum_h w_{ih}\, v_h \right) \times \text{divi}(1, w_{i+}) = \sum_j x_{ij} \frac{\sum_h w_{ih}\, v_h}{w_{i+}}$$

(where the ratio 0/0 is defined as 0), i.e., the ego effect of the average value of covariate V for those to whom actor i has a W-outgoing tie.

For a sienaFit object `ans` created from any effect object, the effect specification is given by

```
ans$requestedEffects
```

while the effects object also including the main effect is given by

```
ans$effects
```

The latter will show, for any included interaction effects, the internal effect parameters used for main effects even if they are not included themselves; this gives a possibility for checking the internal effect parameters used for the interactions.

6.12.2 The `interactionType` of interaction effects

All effects have a so-called `interactionType`, defined by the column `interactionType` in the effects data frame. This interaction type defines what is allowed for definition of interaction effects; the product of their change statistics should make sense as the change statistic of some effect. Further explanation is given in section 'Statistics for MoM' of `Siena_algorithms.pdf`.

For network effects, the interaction type is `"ego"`, `"dyadic"`, or `""` (blank); for behaviour effects, it is `"OK"` or `""`.

6.12.3 Interaction effects for network dynamics

The following kinds of user-defined interactions are possible for network dynamics of directed one-mode networks as well as bipartite networks.

a. Ego effects of actor variables can interact with all effects.

b. Dyadic effects can interact with each other.

For undirected networks, ego effects can only interact with ego effects and dyadic effects can only interact with dyadic effects. Whether an effect is an ego effect or a dyadic effect is defined by the column `interactionType` in the effects data frame. This column is shown in the list of effects that is displayed in a browser by using the function:

```
effectsDocumentation()
```

Thus, in directed networks, a two-way interaction must be between two dyadic effects or between one ego effect and another effect. A three-way interaction may be between three dyadic effects, two dyadic effects and an ego effect, or two ego effects and another effect. In undirected networks all three effects must be either ego or dyadic effect.

All effects used in interactions must be defined on the same network (in the role of dependent variable): that for which the 'unspecified interaction effects' is defined. And all must be of the same type (evaluation, endowment, or creation effects).

Examples of the use of set_interaction are as follows.

```
myeff <- set_interaction(myeff, list(egoX, recip),
                                covar1=c("smoke1", ""))
myeff <- set_interaction(myeff, list(egoX, egoX),
                                covar1=c("smoke1", "alcohol"))
```

Note the `covar1` argument; this argument is used also when defining these effects using set_effect. In this case, however, two effects are defined, and accordingly the `covar1` argument has two components, combined by the c() function. For effects such as `recip` that need no `covar1` argument, the corresponding string is just the empty string, `""`.

Interactions between three effects are defined similarly, but now the `covar1` argument must combine three components.

The number of network interaction effects that can be created is defined by the argument `nintn` in function make_specification. If you need more interactions than permitted, just make a new effects object with the suitably increased value of `nintn`.

The list of effects in Chapter 13 contains a variety of interaction effects that cannot be created in this way; for example, those with short names `transRecTrip`, `simRecipX`, `avSimEgoX`, and `covNetNet` (there are many more).

For non-directed (symmetric) networks, what makes sense as standard interaction effects depends on the `modelType` as well as on the `interactionType` of the effects. These distinctions have not been fully implemented.

6.12.4 Interaction effects for behavior dynamics

For behavior dynamics, interaction effects can be defined by the user, for each dependent behavior variable separately, as interactions of two or three actor variables, again using the function set_interaction. These are interactions on the ego level, in line with the actor-oriented nature of the model.

There are some restrictions on what is permitted as interactions between behavior effects. Of course, they should refer to the same dependent behavior variable. What is permitted depends on the `interactionType` of the effects, which for behavior effects can be `"OK"`[8] or blank. A further explanation is given under the heading 'User-defined interaction effects' in Section 13.2. The `interactionType` of the effects is shown in the list of effects displayed in a browser by using the function:

```
effectsDocumentation()
```

The behavioral effects with non-`"OK"` (i.e., blank) `interactionType` include, in particular, all effects of which the name includes the word `"similarity"`, or alternatively, the short name includes the string `"sim"`.

The requirement for behavior interactions is that, of the interacting effects, all or all but one have the value `"OK"`. Thus, for an interaction between two effects, one or both should be `"OK"`; for a three-effect interaction, two or all three should be `"OK"`.

The number of behavior interaction effects that can be created is defined by the argument `behNintn` in function make_specification. If you need more interactions than permitted, just make a new effects object with the suitably increased value of `behNintn`.

As an example, suppose that we have a data set with a dependent network variable `friendship` and a dependent behavior variable `drinkingbeh` (drinking behavior), and we are interested whether social influence, as represented by the 'average alter' effect, differs between actors depending on whether currently they drink little or much. Then the commands

```
myeff <- set_effect(myeff, avAlt, depvar="drinkingbeh",
                                  covar1="friendship")
myeff <- set_interaction(myeff, list(quad, avAlt), depvar="drinkingbeh",
                                  covar1=c("", "friendship"))
```

define a model with the average alter effect (representing social influence) and an interaction between this and the quadratic shape effect. Recall that the latter can be regarded as the effect of

[8] The value is `"OK"` for the effects of which the formula as defined in Section 13.2.1 is given by z_i multiplied by something not dependent on z_i.

drinking behavior on drinking behavior. Briefly, the interaction is between current drinking behavior and the average drinking behavior of friends. By consulting Section 13.2.1 on the mathematical definitions of the effects one can derive that this leads to the following objective function; where it is assumed that also the linear and quadratic shape effects are included in the model.

$$f_i^{\text{beh}}(x,z) = \beta_1^{\text{beh}} z_i + \beta_2^{\text{beh}} z_i^2 + \beta_3^{\text{beh}} z_i \frac{\sum_j x_{ij}\, z_j}{\sum_j x_{ij}} + \beta_4^{\text{beh}} z_i^2 \frac{\sum_j x_{ij}\, z_j}{\sum_j x_{ij}} .$$

In addition, there are predefined interactions available between actor variables and influence, as described in Section 13.2.1.

6.13 Time heterogeneity in model parameters

When working with two or more periods, i.e., three or more waves, there is the question whether parameters are constant across the periods. This can be tested by the test_time function, as explained in Section 9.5. To specify a model with time heterogeneous parameters, the function includeTimeDummy can be used, as follows. Consider the reformulation of the evaluation function into

$$f_{ij}^{(m)}(\mathbf{x}) = \sum_k \left(\beta_k + \delta_k^{(m)} h_k^{(m)}\right) s_{ik}\big(\mathbf{x}(i \rightsquigarrow j)\big) \tag{8}$$

where m denotes the period (from wave m to wave $m+1$ in the panel data set) and $\delta_k^{(m)}$ are parameters for the effects interacted with time dummies. You can include these in your model simply via the function

```
myeffects <- includeTimeDummy(myeffects,
                     density, reciprocity, timeDummy = "2,3,6")
```

which would add three time dummy terms to each effect listed in the function.

We recommend that you start with simple models, and base the decision to include time heterogeneous parameters on your theoretical and empirical insight in the data (e.g., whether the different waves cover a period where the importance of some of the modeled 'mechanisms' may have changed) and the score type test that is implemented in the test_time function, see Section 9.5.

See Lospinoso et al. (2011) for a technical presentation and examples of how the test works, and Lospinoso and Snijders (2019) for relations with testing goodness of fit (further treated in Section 6.15).

6.14 Limiting the maximum outdegree

It is possible to request that all networks simulated have a maximum outdegree less than or equal to some given value. This is meaningful only if the observed networks also do not have a larger outdegree than this number, for any actor at any wave.

This is carried out by specifying the maximum allowed value in the MaxDegree parameter of the set_model_saom function, which determines the details of the model specification.

MaxDegree is a named vector, which means that its elements have names. The length of this vector is equal to the number of dependent networks. Each element of this vector must have a name which is the name of the corresponding network. E.g., for one dependent network called `mynet`, one could use

```
MaxDegree = c(mynet = 10)
```

to restrict the maximum degree to 10. For two dependent networks called `friends` and `advisors`, one could use

```
MaxDegree = c(friends = 6, advisors = 4)
```

For a single network, the default value 0 is used to specify that the maximum is unbounded. For multiple networks, if for one network there is a bound for the maximum outdegree but for another network this should not be bounded, then the value 0 will not work, but one should use a bound which is at least $n - 1$, where n is the number of actors in the network (or the largest number, if there are multiple groups).

If the MaxDegree parameter is used for data where all, or almost all, degrees are equal to this maximum value, then it is likely that the estimation algorithm will not converge. A fixed choice design for network data collection is not compatible with the free choice nature of the Stochastic Actor-Oriented Model. See Holland and Leinhardt (1973) for a discussion of fixed choice designs and Žnidaršič (2012) for references to more recent literature.

The MaxDegree option does not work for likelihood-based estimation; therefore, neither ML estimation using siena nor multilevel estimation using multi_siena support this option. However, you can obtain the same result of avoiding the creation of outdegrees larger than a given value by using the effect with shortName `outMore` or `outThreshold`. For example, to avoid outdegrees larger than 12,

```
effs <- set_effect(effs, outMore, parameter=12, initialValue=-30, fix=TRUE)
```

(On the log-probability scale, a contribution of `-30` is practically minus infinity.)

6.15 Goodness of fit with auxiliary statistics: test_gof

The function test_gof in RSiena permits users to assess the fit of the model with respect to auxiliary statistics of networks, e.g. geodesic distributions, that are not explicitly fit by a particular effect, but are nonetheless important features of the network to represent by the probability model. This can be used to check, when one has followed the approach to model specification explained in Sections 6.2 to 6.8 – and explained also in Snijders et al. (2010b) –, whether the end result gives a good representation also of these other statistics.

The test_gof function, proposed and elaborated by Lospinoso (2012) and presented further in Lospinoso and Snijders (2019), operates basically by comparing the observed values, at the ends of the periods, with the simulated values for the ends of the periods. The differences are assessed by combining the auxiliary statistics using the Mahalanobis distance.

The examples in the help pages for test_gof and sienaGOF-auxiliary give ample help for how to use this function. Also see the script on the SIENA website.

6.15.1 Auxiliary statistics

Various auxiliary statistics are available directly ('out of the box'), e.g., distributions of indegrees and outdegrees and of behavioral variables. These are listed in the help page for test_gof-auxiliary.

Some other auxiliary statistics, requiring additional packages such as `igraph` and `sna`, are available by copying script from the examples in the same help page. (The reason for this construction is to reduce dependencies of RSiena on other packages.) This includes the auxiliary statistics for the distribution of geodesic distances.

Note that for the indegree and outdegree distributions, you need to specify a parameter `levls`, which is the set of values for which the distribution is calculated. The default value `0:8` often is not adequate, use a different one!!! For example if outdegrees ranging mainly from 0 to 10 but with outliers up to 27, you could use options `cumulative=TRUE` and `levls=c(0:10,15,20,30)` (the details of the distribution in the tail do not matter so much).

A very useful auxiliary variable is dyadicCov. Have a look at the example on the help page for test_gof-auxiliary to see how this can be used to check the adequacy of the representation of ties for ego-alter combinations of actor covariates. For dependent behavioral variables, there is the auxiliary variable egoAlterCombi.

Triad census

Another auxiliary statistic is TriadCensus, which checks the fit for the *triad census* introduced by Holland and Leinhardt (1970, 1976). These authors noted that much of local network structure is expressed by triads, the configurations of ties between triples of nodes. There are 16 different possible configurations of ties between three nodes. The *triad census* is the vector of frequencies of these 16 configurations. These are shown in Figure 2. Holland and Leinhardt (1976) showed that many interesting characteristics of networks can be expressed as linear combinations of the triad census. See, e.g., Chapter 14 of Wasserman and Faust (1994) for more information.

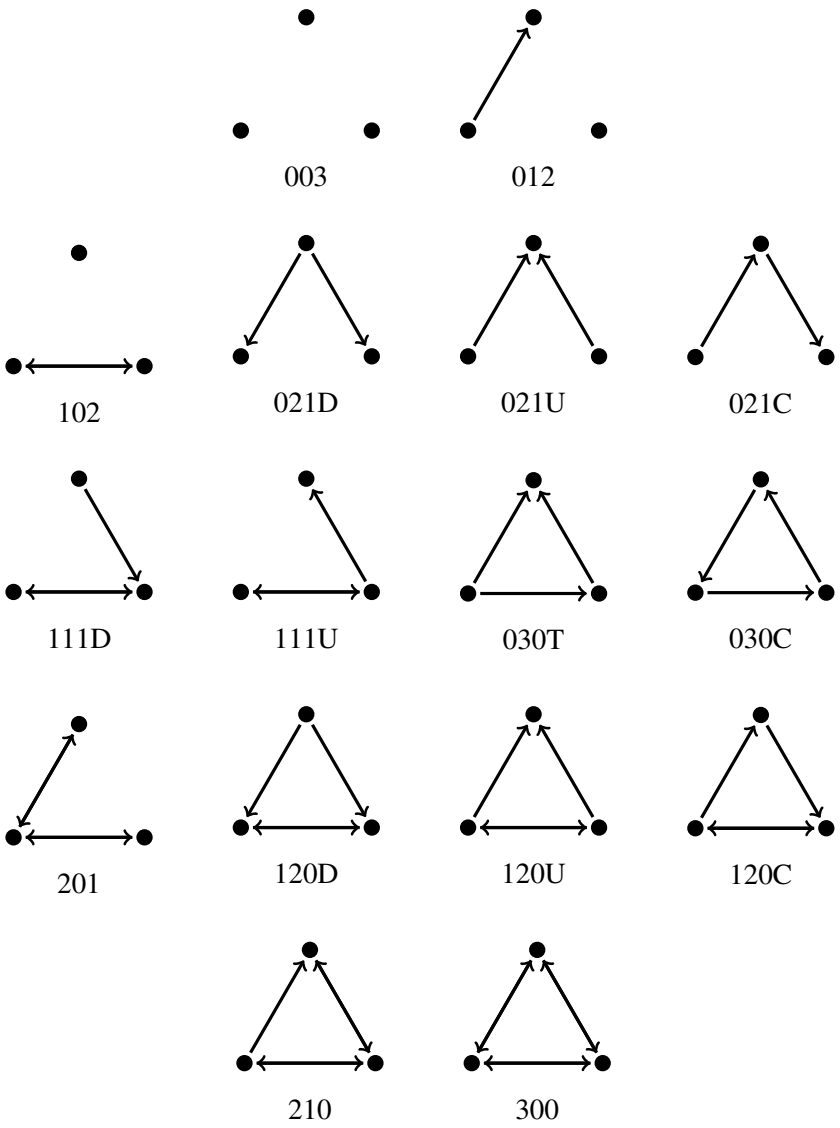


Figure 2: The 16 different configurations possible for three nodes in a digraph (after Holland and Leinhardt, 1976).

6.15.2 Plots

The results of `test_gof` can be plotted which then produces *violin plots* (Hintze and Nelson, 1998), which present the distribution of the statistic as a combination of a box plot and a smooth approximation to the density (by a kernel density estimate), with the observed values superimposed. The violin plots tend to become squiggly when the probability distribution is concentrated on a few points (integers usually) and, as a consequence, the density plot tries to approximate a discrete distribution. For the associated `plot` function, options center and scale are available to equalize the centers and scales of the various statistics plotted. For distributions and cumulative distributions over sets of integers (e.g., of degrees or geodesic distances) it often is advisable to use the defaults center = FALSE, scale = FALSE, whereas for sets of statistics for which a common scale is less important, e.g., triad counts, a clearer picture may be obtained by plotting with center = TRUE, scale = TRUE.

6.15.3 p-values

The p-values for test_gof compare, in the space of outcomes of the auxiliary statistic (which is a vector, with a dimension usually between 4 and 20), the position of the observed data to the cloud of points formed by the simulated data sets that correspond to the estimated model. This comparison is with respect to the 'distance' from the center of the cloud of points, where 'distance' is between quotation marks because it is the Mahalanobis distance, which takes into account the correlations and different variances of the components of the auxiliary statistic.

This means that very small values of p indicate poor fit. There is an important difference here between $p = 0$ and $p > 0$. Suppose that there are, e.g., 1,000 simulated data sets (this number is given by `n3`, the 'length of Phase 3'); then the smallest possible non-zero value of p is 0.001. If $p = 0$, the observed data are further away from the center of the simulated data than any of the simulated data points – this could be very far away indeed. If $p = 0.001$ or larger, then the observed data point must be at least somewhere close to or within the cloud of simulated points, because one or more of the simulated data points are further away from the center than the observed data. Concluding, if $p = 0$, then with respect to the auxiliary statistic the fit is poor; it might be rather poor or extremely poor, and you do not know how extreme it is.

The customary value of $p = 0.05$ may be used as a threshold determining whether the fit is adequate, but this threshold is of even less importance here than it is in the case of regular hypothesis testing.

If you have a multi-group estimation (see Section 12.2), test_gof needs to be applied to each group separately, using the `groupName` argument, usually `groupName="Data1"`, `groupName="Data2"`, etc. (see the help page for test_gof).

For combining test_gof results for multiple groups, the resulting p-values as calculated for each single group can be combined. The most suitable method here perhaps is the inverse normal method, also called Lipták's method; see Hedges and Olkin (1985, Section C.3). This methods transforms each p-value to a standard normal variate, adds these and divides by $\sqrt{N}$ where N is the number of combined studies, and tests the result in the standard normal distribution. For the transformation to standard normal, you can use the R function qnorm (check whether you need `qnorm(p)` or `qnorm(1-p)`). For testing in the standard normal you can use pnorm.

Another good combination procedure is Fisher's method for combining independent p-values, described also in Hedges and Olkin (1985) and (briefly) in Snijders and Bosker (2012, Chapter 3). The basic idea is that, for groups $k = 1, \ldots, K$ which have given the p-values p_k, the test statistic is $-2 \sum_k \ln(p_k)$ and it is tested in a chi-squared distribution with d.f. = $2K$.

An illustration is in script
http://www.stats.ox.ac.uk/~snijders/siena/RscriptMultipleGroups.R.

6.15.4 Composition change, missing data, and structural values in test_gof

The method of joiners and leavers for representing composition change (Section 5.3.3) does not combine properly with the test_gof function.

Missing tie values and structurally determined tie values are treated in the estimation in such a way that they do not contribute directly to the target statistics. This behavior is mirrored in their treatment in test_gof. The aim is that such values do not contribute to any differences between observed and simulated values.

Tie variables that are missing at either the beginning or the end of the period are replaced by 0, both in the observed and in the simulated networks. For behavioral variables they are replaced by missings (NA).

If there are any differences between structural values at the beginning and at the end of a period, these are dealt with as follows. For tie variables that have a structural value at the start of the period, this value replaces the observed value at the end of the period (for the goodness of fit assessment only). For tie variables that have a structural value at the end of the period but a free value value at the start of the period, the reference value for the simulated values is lacking; therefore, the simulated values at the end of the period then are replaced by the structural value at the end of the period (again, for the goodness of fit assessment only).

7 Estimation

The model parameters are estimated under the specification given during the model specification part, using an iterative stochastic approximation algorithm. Four estimation procedures are implemented: the Method of Moments ('MoM'; Snijders, 2001; Snijders et al., 2007); the Generalized Method of Moments ('GMoM'; Amati et al., 2015, 2019); the Method of Maximum Likelihood ('ML'; Snijders et al., 2010a); and a Bayesian method (Koskinen, 2004; Koskinen and Snijders, 2007; Schweinberger and Snijders, 2007a). For non-constant rate functions, currently only MoM and GMoM estimation is available. The Method of Moments is the default; ML and Bayesian estimation methods require much more computing time. Given the greater efficiency but longer required computing time for the ML and Bayesian methods, these can be useful especially for smaller data sets and relatively complicated models (networks and behavior; creation or endowment effects).

In this chapter, the number of parameters is denoted by p. The algorithm is based on repeated (and repeated, and repeated...) simulation of the evolution process of the network. These repetitions are called 'runs' in the following. The MoM estimation algorithm is based on comparing the observed network (obtained from the data files) to the hypothetical networks generated in the simulations.

Note that the estimation algorithm is of a stochastic nature, so the results can vary! This is of course not what you would like. For well-fitting combinations of data set and model, the estimation results obtained in different trials will be very similar. It is good to repeat the estimation process at least once for the models that are to be reported in papers or presentations, to confirm that what you report is a stable result of the algorithm. If you wish to have identical results over different estimations, you can set the seed; see Section 7.2.

7.1 An overview of the estimation procedure

The estimation process is implemented in function siena, using an algorithm specification defined in function set_algorithm_saom. Estimation starts with initial values for the parameters, and the function siena returns a so-called sienaFit object containing the estimates and their standard errors and a lot of further information. Since the estimation procedure is iterative (i.e., depending on the initial value) and stochastic, the results are not always completely satisfactory. We shall see below how the satisfactory convergence of the algorithm can be checked, and how to go on if this is not satisfactory.

The estimation algorithm is determined by a call of functions such as

```
results1 <- siena(data = mydata, effects = myeff)
```

The function set_algorithm_saom can be used to define a different algorithm for the estimation; and function siena carries out the estimation. If you do not want to see the graphical interface with intermediate results (see below), or if your computer has problems showing this, then add the option `batch = TRUE`, as in

```
results1 <- siena(data=mydata, effects=myeff, batch=TRUE,
           control_algo=alg_alg)
```

If you wish to have detailed information at the console about the intermediate steps taken by the algorithm, then add the option `verbose = TRUE`, as in

```
results1 <- siena(data=mydata, effects=myeff, verbose=TRUE,
           control_algo=alg_alg)
```

The graphical interface is available in Windows, and consists of a screen with intermediate results: current parameter values, the differences ('deviation values') between simulated and observed statistics (these should average out to 0 if the current parameters are close to the correct estimated value), and the quasi-autocorrelations discussed in Section 7. (If you are running RSiena from `RStudio`, this screen is visible when you click on the little feather symbol that appears on the right-hand side of the taskbar when the estimation is running.) It is possible to intervene in the algorithm by clicking on the appropriate buttons: the algorithm may be restarted or terminated. In most cases this is not necessary.

Experience shows that parameters estimated by siena are not very large in magnitude, mostly they are less than 20 in absolute value. Therefore, when the algorithm at some moment leads to provisional values larger than, say, 50, this is a signal that the algorithm is diverging. This will lead to a stop of the algorithm; the default threshold value of 50 can be changed by the argument `thetaBound`. E.g., one could use

```
results1 <- siena(data=mydata, effects=myeff, thetaBound=100)
```

Some effects associated with covariates will have values that scale inversely with the covariates. It was mentioned in Section 5.1.5 that actor covariates should be preferably have standard deviations between, say, 0.1 and 10. This will normally lead to parameter values less than 10. The same goes for dyadic covariates.

7.2 Arguments of the algorithm

Function set_algorithm_saom defines the algorithm. The algorithm is described more in detail in Section 7.7. This function has a lot of arguments (keywords) that you will never need to touch, and which have been put there for the purpose of investigating the best settings of the algorithm. The following arguments are worth knowing about.

1. `n3`: the number of iterations in the third phase of the estimation algorithm, determining the precision of the standard errors.

2. `seed`: the seed of the random number algorithm. This seed determines the starting value of the random number algorithm, and thereby leads to exactly reproducible results. Normally, if the estimation procedure has converged as indicated by the convergence criterion (see the next section) and argument `n3` is large enough, different random seeds will give results for converged models that are equivalent for practical purposes. (Well, this is a way to check whether indeed your convergence criterion is strict enough, and whether argument `n3` is large enough.)

Some other arguments are sometimes important in the case of convergence difficulties. Their role is described in the following sections.

3. `useStdInits`: whether the initial values in the effects object will be ignored and default values used instead (default: `FALSE`).

4. `nsub`: the number of subphases in the estimation algorithm (default: 4).

5. `n2start`: the number of iterations in the first subphase.

6. `firstg`: the initial value of the multiplier for the size in the stochastic approximation algorithm.

The default estimation algorithm is the Method of Moments. But it is also possible to use the method of Maximum Likelihood; further explanations are in Section 7.11. This is determined by the following arguments.

7. `maxlike`: whether to use Maximum Likelihood (ML) estimation (default: `FALSE`).

8. `mult`: the multiplication factor for the ML algorithm.

As described in Section 7.12.1, there is a choice between conditional and non-conditional estimation. Conditioning means that for one of the dependent variables, the simulated number of differences is always equal to the observed number of differences (note that the number of ministeps usually will be somewhat larger than the number of differences, because some ministeps may cancel earlier ministeps, and ministeps can have the outcome of 'no change'). The associated arguments are

9. `cond`: whether to use conditional estimation;

10. `condvarno`: the number of the conditioning variable;

11. `condname`: the name of the conditioning variable (an alternative to using `condvarno`).

Furthermore, there is an argument with special purposes that sometimes can be used.

12. `simOnly`: to indicate that the interest is in simulation, not estimation; see Chapter 10.

Finally, there is an argument for which the default is always good; but you might try it out anyway.

13. `dolby`: the default is `TRUE`, which implies that the statistics are regressed on the augmented data score, which is known to have expected value 0. This is explained in Snijders (2013). This yields a considerable reduction of the variance of the statistics without affecting their expected values.
    The regression coefficients are calculated in Phase 1. If `prevAns = ans` is used and the model has remained identical, the regression coefficients from Phase 3 of `ans` are used. These values are given in `ans$regrCoef`; the corresponding correlations are in `ans$regrCor`.

7.3 Results of the estimation

The estimation produces an output file in the current working directory. The default name is determined by the name of the data set. For instance, if the data set has name `mydata`, the output file will be called `mydata_out.txt`. Other names can be set by function set_output_saom. To look at the information, you may either look at this file (which can be opened by any text editor), or produce results on the R console.

A brief summary of the results is given in the R console by typing the name of the sienaFit object. For example, if this name is `results1`,

```
results1
```

could give a summary such as

```
Estimates, standard errors and convergence t-ratios
                             Estimate   Standard   Convergence
                                          Error      t-ratio
Rate parameters:
  0       Rate parameter       6.0803  ( 1.0220   )
  1. eval outdegree (density) -2.5270  ( 0.1589   )    0.0152
  2. eval reciprocity          2.1021  ( 0.3038   )    0.0039
  3. eval transitive triplets  0.5470  ( 0.1988   )    0.0214
  4. eval 3-cycles             0.0805  ( 0.3845   )    0.0369
  5. eval smoke1 similarity    0.4400  ( 0.2560   )   -0.0427

Overall maximum convergence ratio:    0.1608
```

Requesting a longer summary by a command such as

```
summary(results1)
```

will produce more information, including, e.g., the covariance/correlation matrix of the estimators.

Convergence check

The column `Convergence t-ratio` shown above, also called `t statistics for deviations from targets`, is an indicator of convergence. If some of these values are higher in absolute value than 0.1, convergence is not adequate. The value `Overall maximum convergence ratio` is another, stricter, indicator of convergence. For adequate convergence, this value should be less than 0.25. The overall maximum convergence ratio for sienaFit object `results1` is given by `results1$tconv.max`.

In this example, convergence is good. If convergence is not adequate, the estimation must be repeated. Usually the best way to do this is by employing the argument `prevAns` in the call of siena. Given that the earlier result was already called `results1`, this is done, e.g., by

```
results1 <- siena(data=mydata, effects=myeff, prevAns=results1)
```

The practical procedure therefore, in usual cases, is that one runs siena, if necessary repeatedly with subsequent runs using the `prevAns` argument as above, until the `Overall maximum convergence ratio` is less than 0.25 and all `convergence t ratios` are less than 0.1. (The threshold value of 0.25 is not hewn in stone, and small deviations are acceptable.)

In some cases this aim is not easily reached; what to do then is treated in the following.

7.3.1 Initial Values

The *initial values* can be determined in three ways.

1. The default: if `useStdInits = FALSE` and no prevAns argument is given in the call of siena, the initial values are taken from the sienaEffects object, in this example called `myeff`.
   Requesting

   ```
   myeff
   ```

   will show the initial values. As long as no time dummies have been requested using sienaTimeFix, the initial values for the requested effects are in the vector

   ```
   myeff$initialValue[myeff$include]
   ```

   Changing these values is not often necessary, because the argument prevAns, as explained in the next item, does this behind the scenes.
   If one does wish to change the initial values contained in the effects object, this can be done using the function update_theta, which copies the estimates from earlier results, contained in a sienaFit object, to the effects object. For a single effect the initial value can be changed by the set_effect function in which the initialValue then must be set.

2. If `useStdInits = FALSE` and the prevAns ('previous answer') argument is used, such as in

   ```
   results1 <- siena(data=mydata, effects=myeff, prevAns=results0)
   ```

   the initial parameter estimates are taken from the results of what is given as the prevAns argument. This must be a sienaFit object; in this example it is given as `results0`.
   If the specification of the effects object used to obtain `results0` was the same as `myeff`, then not only the initial values are copied, but also Phase 1 of the algorithm is skipped, because information for the sensitivity of the statistics with respect to the parameters is taken from the results of Phase 3 of `results0`.
   If the specification of the effects object used to obtain `results0` was not the same as `myeff`, then for those parameters that do match, the initial values are copied from `results0` and Phase 1 is carried out as usual.

3. If `useStdInits = TRUE` is used in the call of set_algorithm_saom, standard initial values are used.
   These consist of some reasonable values for the rate parameters and the outdegree parameter, as well as for the linear shape parameter for behavioral dependent variables (if any); and 0 parameters for the rest.
   The default is `useStdInits = FALSE`.

7.3.2 Convergence Check

When parameters have been estimated, first the convergence of the algorithm must be checked. This is done by looking at the *t-ratios for convergence* and the *overall maximum convergence ratio*. These are given in the output of the algorithm, presented above. This check considers the deviations between simulated values (in Phase 3, see below) of the statistics and their observed values (the latter are called the 'targets'). Ideally, these deviations should be 0. Because of the stochastic nature of the algorithm, when the process has properly converged the deviations are small but not exactly equal to 0. The program calculates the averages and standard deviations of the deviations and combines these in a t-ratio (in this case, average divided by standard deviation). The overall maximum convergence ratio is the maximum value of the ratio

$$\frac{\text{average deviation}}{\text{standard deviation}}$$

for any linear combination of the target values. A precise definition is given in `Siena_algorithms.pdf` which can be downloaded from the SIENA website. The overall maximum convergence ratio for sienaFit object `results1` is given by `results1$tconv.max`.

Convergence is excellent when the overall maximum convergence ratio is less than 0.2, and for all the individual parameters the t-ratios for convergence all are less than 0.1 in absolute value; convergence is reasonable when the former is less than 0.30. For published results, it is suggested that estimates presented come from runs in which the overall maximum convergence ratio is less than 0.25. (These bounds are indications only, and are not meant as severe limitations.)

In the example above, the largest absolute value of the t-ratios for convergence is equal to 0.0427, and the overall maximum convergence ratio is 0.1608; both are quite good values.

If convergence is not adequate, the best way to continue is by making another estimation run, now carrying on from the last obtained result. This is done by using this result in the prevAns ('previous answer') argument, while taking care that `useStdInits = FALSE` has been specified (but this is the default). An example is

```
results1 <- siena(data=mydata, effects=myeff, prevAns=results1)
```

In this case, this second estimation run produced good results, with a maximum absolute t-ratio for convergence equal to 0.0777. The output file gives more extensive results, viz., the averages and standard deviations of the deviations from targets and the resulting t-ratios:

```
End of stochastic approximation algorithm, phase 3.
---------------------------------------------------

Total of 1822 iterations.
Parameter estimates based on 822 iterations,
basic rate parameter as well as
convergence diagnostics, covariance and derivative matrices based on 1000 iterations.

Information for convergence diagnosis.
Averages, standard deviations, and t-ratios for deviations from targets:
  1.   0.2460  16.1494   0.0152
  2.   0.0560  14.3829   0.0039
  3.   0.9520  44.5338   0.0214
```

```
  4.   0.5380  14.5726   0.0369
  5.  -0.2080   4.8672  -0.0427

Good convergence is indicated by the t-ratios being close to zero.
Overall maximum convergence ratio = 0.1608.
```

For example, for the fourth parameter (3-cycles), the average deviation from the target value was 0.5380, and the standard deviation across the 1000 simulations in Phase 3 was 14.5726. This yields a t-ratio of $0.5380/14.5726 = 0.0369$. Large values of the averages and standard deviations are in themselves not at all a reason for concern; only the t-ratio is important.

7.3.3 Continued estimation to obtain convergence

Above, the prevAns argument was mentioned which will lead to using the result from a previous estimation as the initial value for the next estimation. If convergence is difficult to obtain, one may use other settings of the estimation algorithm, given as arguments in the set_algorithm_saom function, to try and improve convergence. The main arguments of set_algorithm_saom that can be used for this purpose are the following. They are briefly explained in the help file for this function. For the technical background, see `Siena_algorithms.pdf` which can be downloaded from the SIENA website.

- `n2start`
  This is the minimum length of Phase 2.1, i.e., the first subphase of phase 2. The default value is $2.52 \times (p + 7)$, where p is the number of estimated parameters, divided by `nbrNodes`, if multiple processes are used. The minimum lengths of the subsequent subphases[9] are $(2.52)^{k-1} \times$ `n2start` for subphase k.
  This implies the total duration of the algorithm will be roughly proportional to `n2start`. One may try using a value higher than the default.

- `nsub`
  This is the number of subphases in Phase 2 of the algorithm. Usually, the default value of 4 is large enough. Below it is mentioned that sometimes it can be good to use `nsub = 1` in conjunction with a large value of `n2start` to improve convergence after having found a reasonably good, but not totally satisfactory, estimate.

- `firstg`
  This determines the step sizes in the estimation algorithm. If the algorithm is unstable, use a smaller value (but greater than 0). The default value is 0.2. Sometimes for difficult data-model combinations, the algorithm diverges very quickly, and this may be countered by smaller values of `firstg`, e.g., 0.01 or 0.05.

- `n3`
  The number of steps in Phase 3 should be higher for complicated models. The estimation

[9] These values are meant to approximate $(7 + p) \times 2^{4(k+2)/3}$. With the stepwise gain parameter a_N being halved for each new subphase, these values imply that a_N is asymptotically of order $N^{-3/4}$, which is good for convergence of the algorithm.

of the overall maximum convergence t-ratio is biased upward, and the bias is reduced for larger values of `n3`.

- `diagonalize`
  This argument may range from 0 to 1, and determines the extent to which the matrix of derivatives of expected values with respect to parameters is diagonalized. The value 1 (total diagonalization) gives greatest stability; smaller values give greater efficiency. The default for MoM is 0.2.

- `doubleAveraging`
  This replaces the Robbins-Monro updating step by a double averaging step (Bather, 1989; Schwabe and Walk, 1996; Kushner and Yin, 2003) which can be more efficient. The default is `doubleAveraging=0`, which starts using this step from subphase 2.1. Mostly the default value is good and does not need to be changed.

If convergence is not very good even with repeated estimation with the prevAns option, sometimes it can be useful to try and use update_theta to copy the results from the earlier estimation rather than prevAns; this will use the same starting values but not skip Phase 1 of the estimation algorithm, and sometimes this turns out to lead to faster convergence.

Another approach that sometimes can be helpful to obtain convergence in difficult situations is to gradually build up the model, adding further effects while using the prevAns argument to use previous estimates as starting values for the next, extended model. This may be more successful than estimating a complicated model right from the start.

When there are difficulties in obtaining convergence, note that one also should reconsider the model specification; good specification of the model often considerably improves the converge of the parameter estimation. When reading the words here because of a concern about convergence, please read on at least until and including Section 7.5.

7.4 Use of the algorithm arguments

The arguments mentioned in the preceding section can be used in the following way to try and achieve convergence if one does not obtain the desired low level of tconv.max by successive estimation using the prevAns argument sequentially. The advice is to use (1.) or (2.), in both cases together with (3.). It is to be expected that (1.) will be computationally more efficient than (2.)

1. First do an estimation with the default arguments for the algorithm. If that has returned a somewhat reasonable provisional estimate (among other requirements, with a value of tconv.max that is not very high; and *not* with one or more parameter estimates being very high in absolute value), continue the estimation, using prevAns, with nsub=1; a high value for n3, e.g., 3000 or 5000; and a high value for n2start. What is 'a high value for n2start' depends on the data set - model combination, the number of parameters in the first place; and the number of processes used. The baseline value (see the preceding section[10]) would

[10]The value given here is the default number of simulation runs in the 4'th subphase.

be
$(p + 7) \times (2.52)^4$/nbrNodes
where p is the number of estimated parameters (excluding the rate parameters estimated by conditioning, if conditional estimation is used). For a 'high value', start with a value that is about twice as large, Note that computation time in Phase 2 will be proportional to n2start. E.g., with one processor and for a model with $p = 10$ parameters, you might use nsub=1, n2start=2000, firstg=0.01, n3=5000. For $p = 20$ parameters, you could use nsub=1, n2start=4000, firstg=0.01, n3=10000. With multiple processes the value of n2start could be reduced almost proportionately to 1/nbrNodes.

Continue estimation with these algorithmic arguments using prevAns, further perhaps increasing n2start if necessary.

All these choices clearly increase computing time; but less so, and with more likely success, than just carrying on very long with just the prevAns argument and further the default algorithmic arguments.

The following function is meant to iterate the execution of siena until it has converged. It can be modified to suit your further purposes. The argument `ans0` can be employed to use an earlier existing 'on track' estimation result, if available, as the initial value for the algorithm.

```
sienaToConvergence <- function(alg, dat, eff, ans0 = NULL, ...){
 numr <- 0
 ans <- siena(data=dat, effects=eff, prevAns=ans0)
 repeat {
   save(ans, file = paste("ans",numr,".RData",sep = "")) # to be safe
   numr <- numr + 1       # count number of repeated estimations
   tm <- ans$tconv.max    # convergence indicator
   cat(numr, tm,"\n")     # report how far we are
   if (tm < 0.25) {break} # success
   if (tm > 10) {break}   # divergence without much hope
                          #  of returning to good parameter values
   if (numr > 30) {break} # now it has lasted too long
   alg_alg <- set_algorithm_saom(nsub=1, n2start=2000, n3=3000, firstg=0.01)
   ans <- siena(data=dat, effects=eff, prevAns=ans, control_algo=alg_alg)
   }
 if (tm > 0.25)
 {
  cat("Warning: convergence inadequate.\n")
 }
 ans
}
```

7.5 What to do if there are convergence problems

If there are persisting convergence problems even after repeated estimations using the `prevAns` argument and trying out various settings for the algorithm as suggested in the preceding sections,

this may have several reasons.

- The data specification was incorrect (e.g., because the coding was not given properly).
- The starting values were poor. Try restarting from the standard initial values (a certain non-zero value for the density parameter, and zero values for the other parameters); or from values obtained as the estimates for a simpler model that gave no problems. The initial default parameter values can be obtained by choosing the algorithm option 'standard initial values".
- The model does not fit well in the sense that even with well-chosen parameters it will not give a good representation of the data.

  This can be the case, e.g., when there is a large heterogeneity between the actors which is not well represented by effects of covariates. The out-degrees and in-degrees are given in the begin of the RSiena output to be able to check whether there are outlying actors having very high in- or out-degrees, or a deviating dynamics in their degrees. Strong heterogeneity between the actors will have to be represented by suitable covariates; if these are not available, one may define one or a few dummy variables each representing an outlying actor, and give this dummy variable an ego effect in the case of deviant out-degrees, and an alter effect in the case of deviant in-degrees.

  Sometimes transitivity can better be modeled by the GWESP effects (search for this term in the manual) than by transitive triplets. This may help with convergence. For large degrees, the square root versions of degree effects may be better than the raw versions (e.g., `outActSqrt` instead of `outAct`; and for multivariate networks, `outActIntn` with internal effect parameter $p = 2$).

  For modeling large networks, it is important to represent meeting opportunities, e.g., by a suitable dyadic covariate, or by the `sameX` effect of a suitable categorical covariate.

  Sometimes there are important differences between the actors in how many changes they make in their outgoing ties. This should then be reflected by including one or more rate effects. The experience is that the main rate effect to include is the rate effect of the outdegrees (`outRate`, see Section 13.1.5).

  Another possibility is that there is time heterogeneity. Indications about this can be gathered also from the descriptives given in the start of the output file produced by write_report: the number of changes upward and downward, in the network and also – if any – in the dependent behavioral variable. If these do not show a smooth or similar pattern across the observations, then it may be useful to include actor variables representing time trends. These could be smooth – e.g., linear – but they also could be dummy variables representing one or more observational periods; these must be included as an ego effect to represent time trends in the tendency to make ties (or to display higher values of the behavior in question). Further see Section 6.13 for how to discover and handle time heterogeneity.
- Too many weak effects are included. Use a smaller number of effects, delete non-significant ones, and increase complexity step by step. Retain parameter estimates from the last (simpler) model as the initial values for the new estimation procedure, provided for this model

the algorithm converged without difficulties; here also `prevAns` may be used.
Effects that are left out of the estimation can still be used in the model by specifying them with `test=TRUE, fix=TRUE`; this will not burden the estimation process, and give information (with a score-type test, see Section 9.2) about the significance of this excluded effect.
Usually this will be applied with `initialValue=0`, the default. But sometimes it may be done with a plausible non-zero value for `initialValue`.

- Two or more effects are included that are almost collinear in the sense that they can both explain the same observed structures. This will be seen in high absolute values of correlations between parameter estimates, presented in the summary of the results object and also in the output file. In this case it may be better to exclude one of these effects from the model.

- An effect is included that is large but of which the precise value is not well-determined (see above: section on fixing parameters). This will be seen in estimates and standard errors both being large and often in divergence of the algorithm. Fix this parameter to some large value. (Note: large here means, e.g., more than 5 or less than –5; depending on the effect, of course.)

Another trick that may sometimes (rarely!) be tried is the following. Sometimes one (or some) of the rate parameters are especially the causes of difficulties of convergence. Then one may fix this parameter at a good value, and estimate the rest of the parameters. Suppose that this is feasible, i.e., good convergence can be obtained provided that this rate parameter is fixed. Then by trial and error one may find a fixed value for this rate parameter for which the t-ratio for convergence for this parameter also is acceptable (less than 0.2, preferably less than 0.1). Since normally, rate parameters are nuisance parameters (i.e., not of focal interest), this can be an acceptable way out.

7.6 Some important components of the sienaFit object

If a user would like to do further calculations, it can be useful to know about the following components of sienaFit objects. Suppose the object is called `ans`. Some of the components are the following. Further details are in the help file for siena.

`ans$theta`	parameter estimates (but not for the rate parameter used for conditioning; if time dummies were requested using sienaTimeFix, these are also in `theta`)
`ans$covtheta`	covariance matrix of the estimates
`ans$se`	standard errors of the estimates
`ans$pp`	number of parameters
`ans$targets`	targets (observed statistics) for Method of Moments estimation
`ans$nrunMH`	number of Metropolis-Hastings step per wave for Maximum Likelihood estimation
`ans$tconv`	t-ratios for convergence for each of the parameters
`ans$tmax`	maximum absolute value of these ratios for non-fixed parameters

`ans$tconv.max`	maximum t-ratio for convergence for any linear combination of the parameters, called the overall maximum convergence ratio
`ans$sf`	generated statistics in Phase 3 (targets subtracted)
`ans$msf`	covariance matrix of `ans$sf`
`ans$dfra`	estimated derivative of expected statistics w.r.t. parameters,
`ans$ac`	autocorrelations of generated statistics in Phase 3
`ans$sims`	simulated values of dependent variables in Phase 3 of the algorithm if returnDeps = TRUE in the call of siena; in case of Maximum Likelihood estimation (see Section 10.1), note that for observed tie variables the simulated values are equal to the observed; for missing tie variables, the simulated values can be regarded as model-based imputations
`ans$estMeans`	estimated expected values of the target statistics (if the Dolby option was chosen, this is not equal to the average of the simulations!)
`ans$requestedEffects`	the effects object with only the requested effects
`ans$effects`	the effects object including also the main effects for any requested interactions
`ans$x`	the algorithm object used
`ans$f`	everything needed for the calculations in C++; in particular, the data set is hidden in here, and can be reconstructed. Digging in will show as `mat1` the network data, as `mat2` the missings, and as `mat3` the structurally determined values. All these are stored as transposed edge lists. Programmers can consult function `initializeFRAN` for the creation and hence contents of this object.
`ans$version`	the RSiena version
`ans$startingDate`	date and time of the start of the estimation (since version 1.2-17).

Like for any R object, the internal structure of the sienaFit object can be requested by requesting

```
sink("ans.txt")
str(ans)
sink()
```

This writes the structure to the external file `ans.txt`, which may be better than printing it to the console, because it is a long story.
A limited representation of the structure of this object is obtained from

```
sink("ans.txt")
str(ans, 1)
sink()
```

To get some further understanding, one could investigate some of the components of this object as follows. Note that putting a statement between parentheses like in
`(A <- B)` is just a way for constructing the object `A` and showing it at the same time.

```
# Compute the covariance matrix of the generated statistics
print(covsf <- cov(ans$sf))
# This is the same as ans$msf, provided there are no fixed parameters.
# The means and standard deviations of the generated statistics minus targets
(v <- colMeans(ans$sf))
(s <- apply(ans$sf, 2, sd))
# This also allows to compute the convergence t-ratios
v / s
# To get the generated statistics without subtracting the targets,
# we have to add the targets.
# To do this, repeated transposition t can be used:
stats <- t(t(ans$sf) + ans$targets)
# or
stats <- ans$sf + rep(ans$targets, each = nrow(ans$sf))
```

The `tconv` components are used in the function sienaToConvergence presented above.

7.7 Algorithm

The estimation algorithm is an implementation of a procedure of which the original version was proposed by Robbins and Monro (1951). The algorithm is described in Snijders (2001, 2005) and in `Siena_algorithms.pdf` which can be downloaded from the SIENA website. It has three phases:

1. In Phase 1, the parameter vector is held constant at its initial value. This phase is for having a first rough estimate of the matrix of derivatives.

2. Phase 2 consists of several subphases. More subphases means a greater precision. The default number of subphases is 4.
   Each subphase consists of a large number of runs, usually hundreds of them. Each run consists of a simulation of the network dynamics over all periods, using trial parameter values. From this simulation, the estimation statistics are calculated; the devations between the generated statistics and the observed values, which are the target values, are used to update the trial parameter values. For statistics of which the outcome is too high the corresponding parameter will be decreased, if it is too low the parameter will be increased. These changes in the parameter values are smaller in the later subphases.
   If the option `dolby` is in force, the statistics are regressed on the score function of the augmented data. This reduces their variance without affecting their expected value, see Snijders (2013).
   The program searches for parameter values where these deviations average out to 0. This is reflected by what is called the 'quasi-autocorrelations' in the output screen. These are averages of products of successively generated deviations between generated and observed

statistics. It is a good sign for the convergence of the process when the quasi-autocorrelations are negative (or positive but close to 0), because this means the generated values are jumping around the observed values. When estimating by the Method of Moments, it is usual for the quasi-autocorrelations to become close to 0. For estimation by Maximum Likelihood, they will usually eventually tend to fluctuate about some positive values determined by the multiplication factor (see Section 7.11). Large quasi-autocorrelations (larger than .5), when using the Method of Maximum Likelihood, suggest that either the estimation process is still far from its eventual limit (the final estimate), or the multiplication factor may be too small. But in this case, the autocorrelations given in the output file are more important information than those given on the screen.

3. In Phase 3, the parameter vector is held constant again, now at its final value. This phase is for estimating the covariance matrix and the matrix of derivatives used for the computation of standard errors.
   The number of runs in Phase 3 is given by parameter `n3`, with the default value of 1000. For precise estimation of standard errors, higher values of `n3` are desirable. This requires a lot of computing time, but when the number of Phase 3 runs is too low, the standard errors computed are rather unreliable.

The number of subphases in Phase 2, and the number of runs in Phase 3, are determined by parameters nsub and n3 in the call of set_algorithm_saom.

During the estimation process, if the graphical user interface is used (the default `batch = FALSE` in the call of siena), the user can break in and modify the estimation process in two ways:

1. it is possible to terminate the estimation;
2. in Phase 2, it is possible to terminate Phase 2 and continue with Phase 3.

7.8 Output

Output can be obtained in several ways.

1. On the R console.
   When the sienaFit object produced by siena is called `ans`, requesting just `ans` or `print(ans)` produces output on the R console. The function `summary(ans)` produces more extensive output.
2. A table in latex or html format can be produced by the xtable.sienaFit method. For example,

```
xtable(ans, file = "ans1.htm", type = "html")
```

   produces in the working directory a html file with the `ans` results in tabular form. The xtable package has many further options.

3. The function siena_table is available which serves a similar purpose, but not using xtable. It has the option `sig=TRUE` to show asterisks indicating significance of the parameter estimates. This function writes the table to a file; the default file name of the table produced is the name of the sienaFit object. The choice between xtable.sienaFit and siena_table depends on the preference for the tables produced.

   For importing the results of xtable or siena_table into MS-Word, the following steps can be used.

   Request `siena_table(ans)` for some sienaFit object, called `ans` in this example. This will produce, under the default settings, the file `ans.htm` in the current working directory. Copy-paste this into the MS-Word file. In MS-Word then select this table, but only the lines of the header and the parameters, not any preceding blank line nor the footnote. In the MS-Word menu choose 'Insert — Convert text to table -– Autofit to contents'. This will produce the table in your MS-Word file. You can then further modify the table; e.g., change the double minus sign – to the MS-Word minus sign − (available under 'insert -– symbol') and replace the dots for the 't stat.' of the rate parameters by blanks.

4. The function siena writes an output file which is an ASCII ('text') file that can be read by any text editor. Its default name is the name of the data object to which `_out.txt` is appended. A different name can be specified in the call of set_output_saom.

   This output file is divided into sections indicated by a line `@1`, subsections indicated by a line `@2`, subsubsections indicated by `@3`, etc. For getting the main structure of the output, it is convenient to have a look at the `@1` marks first.

The primary information in the output of the estimation process consists of the following three parts.

7.8.1 Convergence check

This was discussed in Section 7.3.2 above.

7.8.2 Parameter values and standard errors

The next crucial part of the output is the list of estimates and standard errors. Suppose that the following result was obtained on the R console.

```
                              Estimate   Standard   Convergence
                                           Error      t-ratio
Rate parameters:
  0       Rate parameter       6.0742  ( 1.0134   )
  1. eval outdegree (density) -2.5341  ( 0.1445   )   0.0571
  2. eval reciprocity          2.1106  ( 0.2625   )   0.0710
  3. eval transitive triplets  0.5449  ( 0.1781   )   0.0584
  4. eval 3-cycles             0.0779  ( 0.3425   )   0.0777
  5. eval smoke1 similarity    0.4519  ( 0.2497   )   0.0400
```

The rate parameter is the parameter called ρ_m^{net} in Section 13.1.5 below (where $m = 1$ because there is only one period). The value 6.0742 indicates that the estimated number of opportunities for change per actor (note that each actor corresponds to a row in the adjacency matrix) between the two observations is 6.07 (rounded in view of the standard error 1.01). Note that this refers to unobserved changes, and that some opportunities for change lead to the decision 'no change', and moreover some of these changes may cancel (make a new choice and then withdraw it again), so the average observed number of differences per actor will be smaller than this estimated number of unobserved changes.

The other five parameters are the weights in the *evaluation function*. The terms in the evaluation function in this model specification are the out-degree effect defined as s_{i1} in Section 13.1.1, the reciprocity effect s_{i2}, transitive triplets effect s_{i3}, three-cycles effect s_{i5}, sex similarity effect s_{i96}. Therefore the estimated evaluation function here is

$$-2.53\, s_{i1}(x) \;+\; 2.11\, s_{i2}(x) \;+\; 0.54\, s_{i3}(x) \;+\; 0.08\, s_{i5}(x) \;+\; 0.45\, s_{i96}(x)$$

where again some rounding was applied in view of the standard errors. The parameter estimates can be combined with the standard errors to test the parameters. (Testing of parameters is discussed more extensively in Chapter 9.)

For the rate parameter, testing the hypothesis that it is 0 is meaningless because the fact that there are differences between the two observed networks implies that the rate of change must be positive. The weights in the evaluation function can be tested by t-statistics, defined as estimate divided by its standard error. (Do not confuse this t-test with the t-ratio for checking convergence; these are completely different although both are t ratios!) Here the t-values are, respectively, -2.5341/0.1445 = –17.54, 2.1106/0.2625 = 8.04, 0.5449/0.1781 = 3.06, 0.0779/0.3425 = 0.23, 0.4519/0.2497 = 1.81. Since the first three are larger than 2 in absolute value, they are significant at the 0.05 significance level. It follows that there is evidence that the actors have a 'preference' for reciprocal and transitive relations. For thee-cycles, the effect is not significant ($t = 0.23$), for smoking similarity it is significant at the 0.10 significance level. The value of the density parameter is not very important; it is important that this parameter is included to control for the density in the network, but as all other statistics are correlated with the density, the density is difficult to interpret by itself.

7.8.3 Collinearity check

In the output file, the covariance matrix of the estimates is presented. This can also be requested by `summary(ans)`. For conditional estimation, the rate parameters of the dependent variables used for conditioning are not included in this matrix. In this case the covariance matrix is as follows.

```
Covariance matrix of estimates (correlations below diagonal):
       0.021       -0.018       -0.010        0.006       -0.008
      -0.468        0.069        0.008       -0.034       -0.002
      -0.395        0.180        0.032       -0.049        0.003
       0.130       -0.378       -0.795        0.117       -0.001
      -0.223       -0.037        0.074       -0.007        0.062
```

The diagonal values are the variances, i.e., the squares of the standard errors (e.g., for the reciprocity effect, 0.069 is the square of 0.2625). Below the diagonal are the correlations. E.g., the correlation between the estimated outdegree effect and the estimated reciprocity effect is –0.468. These correlations can be used to see whether there is an important degree of collinearity between the effects. Collinearity means that several different combinations of parameter values could represent the same data pattern, in this case, the same values of the network statistics. When one or more of the correlations are very close to -1.0 or +1.0, this is a sign of near collinearity. This will also lead to large standard errors of those parameters. It may then be advisable to omit one of the corresponding effects from the model, because it may be redundant given the other (strongly correlated) effect; but see below. It is possible that the standard error of the retained effect becomes much smaller by omitting the other effect, which can also mean a change of the t-test from non-significance to significance.

The suggestion of omitting effects that lead to high parameter correlations with other effect does not directly apply to effects that should be included for other reasons, such as the density effect for network dynamics and the linear and quadratic shape effects for behavior dynamics.

However, correlations between parameter estimates close to -1.0 or +1.0 should not be used too soon in themselves as reasons to exclude effects from a model. This is for two reasons. In the first place, network statistics often are highly correlated (for example, total number of ties and number of transitive triplets) and these correlations just are one of the properties of networks. Second, near collinearity is not a problem in itself, but the problem (if any) arises when standard errors are high, which may occur because the value of the parameters of highly correlated variables is very hard to estimate with any precision. The problem resides in the large standard errors, not in itself in the strong correlation between the parameter estimates. If for both parameters the ratio of parameter estimate to standard error, i.e., the t-ratio, is larger than 2 in absolute value, in spite of the high correlations between the parameter estimates, then the significance of the t-test is evidence anyway that both effects merit to be included in the model. In other words, in terms of the 'signal-to-noise ratio': the random noise is high but the signal is strong enough that it overcomes the noise.

As a rule of thumb for parameter correlations, usually for correlations of estimated structural network effects there is no reason for concern even when these correlations are as strong as .9.

In the example above, the strongest correlation was found between the parameter estimates for transitive triplets and three-cycles. This is not surprising, because both are triadic effects. In this case, the three-cycle effect was not significant, and can be dropped for that reason.

7.9 Other estimation procedures

RSiena can estimate models by four estimation methods: the (unconditional or conditional) Method of Moments ('MoM', the default; Snijders, 2001; Snijders et al., 2007); the Generalized Method of Moments ('GMoM', see Amati et al., 2015, 2019); the Maximum Likelihood method ('ML', see Snijders et al., 2010a), and Bayesian methods (see Koskinen, 2004; Koskinen and Snijders, 2007; Schweinberger and Snijders, 2007a; Koskinen and Snijders, 2023).

Relative to the MoM, the GMoM is theoretically more efficient and somewhat more time-consuming than the regular MoM since it requires a large number of simulations and iterations of the model estimation (see Section 7.10.1). In models for one network, the MoM and the GMoM yield similar estimates. However, the GMoM estimator of the parameters of the effects related

to the new statistics is somewhat more efficient than the MoM estimator, especially for the co-evolution model. When the statistics are informative (i.e., when the number of simultaneous network and behavioral changes made by the same actor between any two observations is not negligible), and when the 'true model' has only evaluation effects for the GMoM statistics, the GMoM leads to more accurate estimates with smaller bias and standard errors than the regular MoM. This is documented in Amati et al. (2015, 2019).

Comparison among all the four methods indicates that, in models for one network, without co-evolution, in nice situations (data sets that are not too small, model specifications that do not request too much from the data, good fit between data and model specification), the four methods tend to agree and there seems to be no reason to use the more time-consuming ML or Bayesian methods. For co-evolution models and not-so-nice situations (very small network data sets, small network and behavior data sets in combination with complex models), however, GMoM, ML and Bayesian methods tend to produce more accurate results (i.e., smaller standard errors) than MoM. Statistical theory suggests that, if the model is well-specified, GMoM and ML are more efficient estimation methods than MoM in the sense of producing estimates with smaller standard errors. But in the 'nice situations' with only one dependent network variable the efficiency advantage of ML is very small. Bayesian estimation is based on a different statistical paradigm, and assumes and requires that the uncertainty about parameters is expressed itself in a probability distribution.

Estimation by the regular Method of Moments (MoM) and Generalized Method of Moments (GMoM) is implemented in the function siena. ML estimation is done by the function siena, using a set of options created by set_algorithm_saom with maxlike = TRUE. Bayesian estimation is done by the function multi_siena, but is as yet implemented only for multilevel network modeling (Section 12.4).

7.10 Generalized Method of Moments estimation

7.10.1 Using the function siena

Compared to the regular Method of Moments (MoM), the Generalized Method of Moments (GMoM) allows using more statistics than parameters, thereby exploiting more information in the observed data. The additional information derives from the use of supplementary statistics, referred to as GMoM statistics. The GMoM statistics are defined only for evaluation effects. Thus, they are used under the assumption that effects operate in the same strength for the creation and the dissolution of ties, and the increasing and decreasing of the behavior. In other words, the creation and the endowment effects corresponding to the GMoM statistics are assumed to be zero (see Section 6.1.3).

Estimation by the GMoM with the function siena is obtained by specifying the GMoM statistics in the effects object using function includeGMoMStatistics, and setting option `gmm=TRUE` in the algorithm object created by the function set_algorithm_saom.

Although the statistics are added to the effect object, one should keep in mind that the GMoM statistics are not effects in the stochastic model, i.e., they are not included as terms in the objective function. They are only used for parameter estimation to evaluate the moment conditions.

Several GMoM statistics have been implemented.

GMoM statistics for network evolution

The statistics for the network evolution (Amati et al., 2015) complement the information provided by the regular statistics by considering two consecutive observations of the network. This formulation allows distinguishing between different antecedents of a same effect. The implemented statistics are named `newrecip`, `persistrecip`, `realrecip`, `realtrans`, `agreetrans`.

GMoM statistics for co-evolution of networks and behavior

The GMoM statistics for the co-evolution of networks and behaviors (Amati et al., 2019) supplement the information of the cross-lagged statistics, depending jointly on network and behavior, by considering the network and behavior at the same time point. Thus, the GMoM statistics account for the information on the simultaneous observation of network and behavioral changes. The names of the implemented GMoM statistics are `avSim_gmm`, `avAlt_gmm`, `totAlt_gmm`, `maxAlt_gmm`, `minAlt_gmm`, `egoX_gmm`, `simX_gmm`.

For the GMoM, the additional statistics used for estimation are specified by the function includeGMoMStatistics. For example, the following lines

```
myeff <- includeGMoMStatistics(myeff, egoX_gmm, covar1 = "mybeh")
myeff <- includeGMoMStatistics(myeff, simX_gmm, covar1 = "mybeh")
```

specify the GMoM statistics for the covariate-related activity and similarity effects.

The distinction between model effects and GMoM statistics is recalled when printing the effect object in the R console. As illustrated by the example below, the output is divided into two parts. The first part lists the model effects. The second part reports the statistics used for the estimation: the regular statistics, included by default when an effect is specified in the probability model using set_effect, and the GMoM statistics, specified with the function includeGMoMStatistics. The comparison between the number of effects and that of the statistics illustrates that, for the GMoM, the one-to-one correspondence between effects operating for the evolution of networks and/or behaviors, and statistics operating for parameter estimation is lost.

```
  Effects and statistics for estimation by the Generalized Method of Moments

 Effects
   name  effectName                       include fix   test  initialValue parm type
1  mynet constant mynet rate (period 1) TRUE    FALSE FALSE    4.69604   0    rate
2  mynet constant mynet rate (period 2) TRUE    FALSE FALSE    4.32885   0    rate
3  mynet outdegree (density)             TRUE    FALSE FALSE   -1.46770   0    eval
4  mynet reciprocity                     TRUE    FALSE FALSE    0.00000   0    eval
5  mynet transitive triplets             TRUE    FALSE FALSE    0.00000   0    eval
6  mynet 3-cycles                        TRUE    FALSE FALSE    0.00000   0    eval
7  mynet mybeh ego                       TRUE    FALSE FALSE    0.00000   0    eval
8  mynet mybeh similarity                TRUE    FALSE FALSE    0.00000   0    eval
9  mybeh rate mybeh (period 1)           TRUE    FALSE FALSE    0.70571   0    rate
10 mybeh rate mybeh (period 2)           TRUE    FALSE FALSE    0.84939   0    rate
11 mybeh mybeh linear shape              TRUE    FALSE FALSE    0.32237   0    eval
12 mybeh mybeh quadratic shape           TRUE    FALSE FALSE    0.00000   0    eval
13 mybeh mybeh total similarity          TRUE    FALSE FALSE    0.00000   0    eval
```

```
 Regular and GMoM statistics
   name  effectName                      Statistic
1  mynet constant mynet rate (period 1) Regular
2  mynet constant mynet rate (period 2) Regular
3  mynet outdegree (density)            Regular
4  mynet reciprocity                    Regular
5  mynet transitive triplets            Regular
6  mynet 3-cycles                       Regular
7  mynet mybeh ego                      Regular
8  mynet mybeh similarity               Regular
9  mybeh rate mybeh (period 1)          Regular
10 mybeh rate mybeh (period 2)          Regular
11 mybeh mybeh linear shape             Regular
12 mybeh mybeh quadratic shape          Regular
13 mybeh mybeh total similarity         Regular
14 mynet mybeh ego                      GMoM
15 mynet mybeh similarity               GMoM
```

GMoM statistics for co-evolution of multiple networks

Similarly to the co-evolution of networks and behavior as treated in (Amati et al., 2019), GMoM statistics have been implemented for the co-evolution of multiple networks, using contemporaneous statistics that supplement the cross-lagged statistics of the MoM. supplement the information of the cross-lagged statistics, depending jointly on network and behavior, by considering the network and behavior at the same time point. The implemented GMoM statistics are `crprod_gmm`, `to_gmm`, and `from_gmm`, supplementing the effects with shortnames `crprod`, `to`, and `from`.

Operation of GMoM estimation

The computation of the GMoM estimates is more time consuming than that of the regular MoM. The GMoM estimator is computed by minimizing the distance between the expected value of the statistics and their sample counterpart. The definition of the distance demands the computation of a matrix of weights describing the importance of the statistics and their sensitivity to the parameters. To estimate the matrix of the GMoM weights, a larger number of simulations in phase 1 and phase 3 and (at least) one iteration of the function siena is used.

The following estimation steps are recommended to reach the convergence of the algorithm quickly.

1. Run the function siena with one or two sub-phases (i.e., `nsub = 1` or `nsub = 2`) in phase 2 and a large number of simulations in phase 3 (e.g., `n3 = 5000`) to get better initial estimates of the model parameters and a more stable estimation of the matrix of GMoM weights.

2. Re-estimate the model using the estimates produced in 1 by using four sub-phases in phase 2 (`nsub = 4`), the same number of simulations in phase 3 (e.g., `n3 = 5000`) and the argument `prevAns = TRUE`.

If convergence is not adequate, one runs siena repeatedly with `prevAns = TRUE`.

Difficulties in convergence might occur when the GMoM statistics are 'highly' correlated (> 0.9) with the regular statistics. Dropping the corresponding GMoM statistics would solve the problem. Intuitively, when the GMoM and the regular statistics are highly correlated, the GMoM statistics do not carry relevant information for the estimation of the parameters and can be discarded. Therefore, it is recommended to inspect the covariance matrix of the statistics to decide whether GMoM statistics should be used for the estimation.

We refer to the R script `GMoMscript.R` on the SIENA webpage for an illustration of the GMoM estimation, the steps described above and the inspection of the covariance matrix of the statistics.

Like the MoM, the GMoM allows conditional as well as unconditional estimation (see Section 7.12.1).

The specification of the effects object for GMoM estimation requires that in the effects object, apart from the basic rate effects, some of the effects were specified in set_effect with `type = "eval"` (the default, which means that this does not need to be stated) and the others with `type = "gmm"`. The first then are evaluation effects defining the model specification, the second are statistics used for estimation. The method requires that the number of statistics (`type = "gmm"`) is equal to or larger than the number of evaluation effects (`type = "eval"`). For example, the commands

```
net <- as_dependent_rsiena(array(c(s501,s502,s503), dim = c(50,50,3)))
dataset <- make_data_rsiena(net)
eff <- make_specification(dataset)
eff <- set_effect(eff, density)
eff <- set_effect(eff, density, type="gmm")
eff <- set_effect(eff, recip)
eff <- set_effect(eff, list(recip, realrecip, persistrecip), type="gmm")
eff <- set_effect(eff, transTrip)
eff <- set_effect(eff, list(transTrip, agreetrans, realtrans), type="gmm")
eff
```

will print the resulting effects object

```
  For estimation by the Generalized Method of Moments
  Effects
  effectName                    include fix   test  initialValue parm type
1 constant net rate (period 1) TRUE    FALSE FALSE    4.69604   0    rate
2 constant net rate (period 2) TRUE    FALSE FALSE    4.32885   0    rate
3 outdegree (density)          TRUE    FALSE FALSE   -1.46770   0    eval
4 reciprocity                  TRUE    FALSE FALSE    0.00000   0    eval
5 transitive triplets          TRUE    FALSE FALSE    0.00000   0    eval

 Statistics
  effectName          include type
1 outdegree (density) TRUE    gmm
```

```
2 reciprocity          TRUE    gmm
3 persistent recip.    TRUE    gmm
4 real recip.          TRUE    gmm
5 transitive triplets  TRUE    gmm
6 real trans. trip.    TRUE    gmm
7 agree trans. trip.   TRUE    gmm
```

with three evaluation effects and seven statistics. Like the MoM, the GMoM allows conditional as well as unconditional estimation (see Section 7.12.1). In this example, for conditional estimation the number of statistics will be nine, including the two statistics for the rate parameters, viz., Hamming distances between subsequent waves.

7.11 Maximum Likelihood and Bayesian estimation

An important difference between estimation by the Method of Moments (MoM) and estimation by the Method of Maximum Likelihood (ML) is that the MoM tries to find parameters that will produce simulations, given the beginning of the waves, for which on average the target statistics are equal to their observed values; whereas ML tries to find parameters for which the probability to arrive in the observed data for the end of the waves, given the beginning of the waves, is maximal. The MoM simulations take account only of the observations at the start of the wave; the ML simulations augment the data by connecting the observations at the start of each wave with the observation at the end of the wave. Such a connection may be called a *change path*. This is more complicated, and therefore more time-consuming. Technical accounts of the MoM are in Snijders (2001); Snijders et al. (2007), of the ML estimator in Snijders et al. (2010a). Both are presented in Snijders (2017).

ML estimation is done by the function siena, using a set of options created by set_algorithm_saom with `maxlike = TRUE`. Bayesian estimation is done by the function multi_siena. The following further information in this section is about ML estimation; Bayesian estimation is as yet implemented only for multilevel network modeling (see Section 12.4).

There are some restrictions to the application of ML estimation.

1. The method of joiners and leavers (Section5.3.3) is not available.
2. The following data configurations are not allowed:
   (a) tie variables in two consecutive waves changing from structural zero (code 10) to 1;
   (b) tie variables in two consecutive waves changing from structural one (code 11) to 0;
   (c) tie variables in three consecutive waves changing from structural zero (code 10) to `NA` to 1;
   (d) tie variables in three consecutive waves changing from structural one (code 11) to `NA` to 0;
   (e) and, for more than three consecutive waves, similar patterns with more NAs in between.

3. Using structural zeros to indicate intermittent absence of actors (i.e., structural zeros only at the beginning or only at the end of a period) is probably not a good idea. Above, some situations were mentioned that are logically impossible and therefore not allowed. In addition, the following is problematic:

    - tie variables that are 0 or 1 at the beginning of a wave and structurally zero (code 10) at the end of the wave, indicating that the actor has left the network.
      The reason is that this contains the information for RSiena that during this period, the tie variable has to change to the value 0; whereas probably you wish to convey the meaning that its value is unknown. At the end of a period, for a tie variable with the value 10, the contribution of this tie variable is not included in the computation of the log-likelihood, but through its dependence with other variables it still has consequences different from `NA`.

    A solution for the problem of structural zeros at the end of a period may be to replace them by `NA`.

    For data sets with multiple periods (i.e., three or more waves), the analysis using ML still is possible (as proposed by de la Haye et al., 2017) by arranging the data as a multi-group data set (Section 12.2). Unfortunately, this solution is not available for multi_siena.

The ML estimation algorithm uses Metropolis-Hastings steps to simulate the score function, which is the expected value of the augmented-data score function; see Snijders et al. (2010a). For ML estimation, an important argument for tuning the algorithm is the so-called multiplication factor, given in set_algorithm_saom as the argument `mult`. This determines the number of Metropolis-Hastings steps taken for simulating each new change path. The number of steps (sometimes called 'sampling frequency' in the literature) is the multiplication factor multiplied by the sum over dependent variables of the distances between successive waves. When this is too low, the sequentially simulated change paths are too similar, which will lead to high autocorrelation in the generated statistics. This leads to poor performance of the algorithm. These autocorrelations are given in the output file. When some autocorrelations are more than 0.4, it is good to increase the multiplication factor. When the multiplication factor is unnecessarily high, on the other hand, computing time will be unnecessarily high. The advice is to aim at values for the autocorrelations between 0.1 and 0.4.

The number of Metropolis-Hastings steps (sampling frequency) is potentially different for each wave, and for the answer object `ans` these numbers are given by `ans$nrunMH`.

A practical way to proceed is as follows. For initial tuning of the multiplication factor, use the model that is obtained as the default after creating the effects object, with very few effects included. The reason for using this model is the limited computation time and easy convergence. If the highest autocorrelation is more than 0.3, increase the multiplication factor (e.g., by making it twice as large; which will also lead to a twice as long computation time) and estimate the model again. If the highest autocorrelation is less than 0.1, then decrease the multiplication factor and estimate again. Tune the multiplication factor until the highest autocorrelation is between 0.1 and 0.3. Then start with estimating the models of interest. For other models the autocorrelations may change again, therefore it still can be important later on to adapt the multiplication factor to keep the highest autocorrelation less than 0.4.

Note that the multiplication factor can be given as one number, but also as a vector, with the number of elements equal to the number of basic rate parameters in the effects object; i.e., the number of periods times the number of waves. This permits setting a high value of the multiplication factor for only some period-wave combinations.

Some advice about how to set the value of the multiplication factor as a vector is in `https://www.stats.ox.ac.uk/~snijders/siena/sienaBayes_s.pdf`; this is formulated there for multi_siena, but applies equally well for ML estimation using siena.

The Metropolis Hastings algorithm uses a proposal distribution composed of steps; see Snijders et al. (2010a) and `Siena_algorithms.pdf`. Abbreviated names of these steps are 'InsDiag', 'CancDiag', 'Permute', 'InsPerm', 'DelPerm', 'InsMiss', 'DelMiss', 'InsMisdat', 'DelMisdat', and 'Move'. The 'Move' step was added in version 1.3.18, and is described in Greenan (2015b). The probabilities of the first 7 steps are determined by the argument `prML` in set_algorithm_saom. The probability of 'Move' is 1 minus the sum of these 7 probabilities; 'InsMisdat' and 'DelMisdat' are offspring of 'InsPerm" and 'DelPerm' and don't need their own proposal probability. 'InsMiss' and 'DelMiss' are superfluous when there are no missing data, but this is taken care of automatically and does not need to be specified in `prML`. Currently, 'Move' steps cannot be taken for data sets with more than one dependent variable.

Another argument of the algorithm that sometimes needs tuning (but less often than the multiplication factor) is the *Initial value of the gain parameter*. This determines the step sizes in the parameter updates in the iterative algorithm. It influences the stability and speed of moving of the algorithm. A too low value implies that it takes very long to attain a reasonable parameter estimate when starting from an initial parameter value that is far from the 'true' parameter estimate. A too high value implies that the algorithm will be unstable, and may be thrown off course into a region of unreasonable (e.g., hopelessly large) parameter values.
When using the Method of Moments (the default estimation procedure), it usually is unnecessary to change this. In the ML case, when the autocorrelations are smaller than 0.1 but the `t statistics for deviations from targets` are relatively small (less than, say, 0.3) but do not all become less than 0.1 in absolute value in repeated runs of the estimation algorithm, then it will be good to decrease the initial value of the gain parameter. Do this by dividing it by, e.g., a factor 2 or a factor 5, and then try again a few estimation runs.

7.11.1 Treatment of missing data

For missing data in the dependent variables (not in the covariates!), a different procedure is used than the one in the Methods of Moments estimation procedure. The missing data at the beginning of the wave are modeled as random variables with a simple prior distribution, given by the distribution of the non-missing entries for this wave. (For Maximum Likelihood estimation, this is a Bayesian incursion in an otherwise frequentist approach...) The missing data at the end of the wave are modeled as random variables with the distribution implied by the Markov chain model with the estimated parameters.

7.12 Other remarks about the estimation algorithm

7.12.1 Conditional and unconditional estimation

RSiena has two methods for MoM estimation and simulation: conditional and unconditional. They differ in the *stopping rule* for the simulations of the network evolution. In unconditional estimation, the simulations of the network evolution in each time period (and the co-evolution of the behavioral dimensions, if any are included) carry on until the predetermined time length (chosen as 1.0 for each time period between consecutive observation moments) has elapsed. This is explained in Snijders (2001, Section 4.2).

In conditional estimation, in each period the simulations run on until a stopping criterion is reached that is calculated from the observed data. Conditioning is possible for each of the dependent variables (network, or behavior), where 'conditional' means 'conditional on the observed number of changes on this dependent variable'. However, if there is more than one dependent variable (network and/or behavior), estimation can be conditional only on one of them.

Conditioning on the network variable means, for each period, running simulations until the number of entries that differ between the adjacency matrix of the initially observed network of this period and the simulated network is equal to the number of entries in the adjacency matrix that differ between the initially and the finally observed networks of this period.

Conditioning on a behavioral variable means, for each period, running simulations until the sum of absolute score differences on the behavioral variable between the initially observed behavior of this period and the simulated behavior is equal to the sum of absolute score differences between the initially and the finally observed behavior of this period.

Conditional estimation is slightly more stable and efficient, because the corresponding rate parameters are not estimated by the Robbins Monro algorithm, so this method decreases the number of parameters estimated by this algorithm.

The choice between unconditional and the different types of conditional estimation is made in the set_algorithm_saom function by setting the cond argument. For data specifications with multiple dependent variables, at most one dependent variable can be used for conditioning. This choice is made dependent on the condvarno and condname arguments in this function.

If there are changes in network composition (see Section 5.3.3), only the unconditional estimation procedure is available.

If there are a lot of structurally determined values (see Section 5.3.1) then unconditional estimation is preferable.

7.12.2 Fixing parameters

Sometimes an effect must be present in the model, but its precise numerical value is not well-determined. E.g., if the network at time t_2 would contain only reciprocated choices, then the model should contain a large positive reciprocity effect but whether it has the value 3 or 5 or 10 does not make a difference. This will be reflected in the estimation process by a large estimated value and a large standard error, a derivative which is close to 0, and sometimes also by lack of convergence of the algorithm. (This type of problem also occurs in maximum likelihood estimation for logistic regression and certain other generalized linear models; see Geyer and Thompson (1992, section

1.6), Albert and Anderson (1984); Hauck and Donner (1977).) In such cases this effect should be fixed to some large value and not left free to be estimated. This can be specified by using the set_effect function with the fix = TRUE option.

7.12.3 Automatic fixing of parameters

If the algorithm encounters computational problems, sometimes it tries to solve them automatically by fixing one (or more) of the parameters. This will be noticeable because a parameter is reported in the output as being fixed without your having requested this. This automatic fixing procedure is used, when in Phase 1 one of the generated statistics seems to be insensitive to changes in the corresponding parameter.

This is a sign that there is little information in the data about the precise value of this parameter, when considering the neighborhood of the initial parameter values. However, it is possible that the problem is not in the parameter that is being fixed, but is caused by an incorrect starting value of this parameter or one of the other parameters.

When the warning is given that the program automatically fixed one of the parameter, try to find out what is wrong.

In the first place, check that your data were entered correctly and the coding was given correctly, and then re-specify the model or restart the estimation with other (e.g., 0) parameter values. Sometimes starting from different parameter values (e.g., the default values implied by the algorithm option of 'standard initial values") will lead to a good result. Sometimes, however, it works better to delete this effect altogether from the model.

It is also possible that the parameter does need to be included in the model but its precise value is not well-determined. Then it is best to give the parameter a large (or strongly negative) value and indeed require it to be fixed (see Section 11.1).

7.12.4 Required changes from conditional to unconditional estimation

Even though conditional estimation is slightly more efficient than unconditional estimation, there is one kind of problem that sometimes occurs with conditional estimation and which is not encountered by unconditional estimation.

It is possible (but luckily rare) that the current parameter values are such that the conditional simulation does not succeed in ever attaining the condition required by its stopping rule (see Section 7.12.1). This will depend on the initial parameter values and the changes that occurred during the Robbins-Monro process. The solution is either to use different (perhaps standard) initial values or to go over to unconditional estimation.

7.13 Using multiple processes

1. If multiple processors are available, then using multiple processes can speed up the estimation in siena, and test_gof. The help pages for these functions show how to do this. You can find out the number of processes possible by

```
library(parallel)
detectCores()
```

It is not advisable to utilize all available cores, because other processes also have to run. If you are not the only user of the machine, then there are obvious issues to be dealt with.

There are some examples in the help page for siena.

2. For estimation by Method of Moments (MoM), in Phases 1 and 3 the simulations are performed in parallel. In Phase 2, multiple (viz., `nbrNodes` which is an argument given in the call of siena) simulations are done with the same parameters, and the resulting statistics are averaged for the updating step in the Robbins Monro algorithm. The gain parameter is increased and the number of iterations in Phase 2 reduced to take advantage of the increased accuracy of the update.
   When using the MoM, decrease in computation time will be somewhat less than proportional to the number of processes used, if this number is less than (say) 10; larger number of processes will have diminishing proportional returns.

3. For estimation by Maximum Likelihood (ML) and by multi_siena, parallelization goes by period; for multi-group data sets, period $\times$ group. Therefore, for this type of estimation, the maximum meaningful number of parallel processes is
   (number of waves – 1) $\times$ (number of groups).

4. The arguments required to run all processes on one computer are fairly simple: in your call to siena, set nbrNodes to the number of processes and useCluster and initC to TRUE.

5. To use more than one machine is more complicated, but it can be done by using, in addition, the clusterString argument. The computers need to be running incoming ssh.

6. For machines with exactly the same layout of R directories on each, simply set clusterString to a character vector of the names of the machines.

7. For other cases, e.g. using Macs alongside Linux, see the documentation for the package parallel.

8. RSiena uses sockets for inter-process communication.

9. On Windows, sub processes are always started using R scripts. On Linux and Mac there is an option available in R version 2.14.0 or later via the code interface to siena to ask for the sub processes to be formed by forking. See the help page for details.

10. Each process needs a copy of the data in memory. If there is insufficient memory available there will be no speed gain as too much time will be spent paging.

11. In each iteration the main process waits until all the other processes have finished. The overall speed is therefore that of the slowest process, and there should be enough processors to allow them all to run at speed.

8 Standard errors

The estimation of standard errors of the MoM estimates requires the estimation of derivatives, which indicate how sensitive the expected values of the statistics (see Section 7.7) are with respect to the parameters. The derivatives can be estimated by two methods:

- finite differences method with common random numbers,
- score function method.

The finite difference method is explained (briefly) in Snijders (2001), the score function method was developed in Schweinberger and Snijders (2007b) (where also the finite difference method is explained). The score function method is preferable, because it is unbiased and demands less computation time than finite differences, although it requires more iterations in Phase 3 of the estimation algorithm (see Section 7.7). It is recommended to use the score function method with at least 1000 iterations (default) in Phase 3. For published results, it is recommended to have at least[11] 5000 iterations in Phase 3.

In some cases, with high correlations between parameters, the estimation of standard errors is unstable and even 5000 may be not enough. This can be checked this by running siena several times, with different random number seeds (or the default `seed = NULL` which also gives different seeds every run).

A more extensive way of checking this is presented in the scripts
http://www.stats.ox.ac.uk/~snijders/siena/SE_checks.R
and
http://www.stats.ox.ac.uk/~snijders/siena/RscriptStandardErrors.R

To calculate the standard errors for a given set of estimates, when these estimates are good and do not need to be changed, algorithm settings can be used with `nsub = 0` in set_algorithm_saom, so phases 1 and 2 are skipped and the whole operation of siena takes considerably less time.

The method for estimating derivatives is set by the findiff parameter and the number of iterations in Phase 3 by the n3 parameter, both in function set_algorithm_saom that creates the object with specifications for the algorithm.

8.1 Multicollinearity

Multicollinearity means that the matrix that is inverted to give the correlation matrix is ill-conditioned. Correlations between parameter estimates close to ± 1 are the most usual signs of this.

If the parameter estimates are perfectly collinear (standard errors of some parameters, or linear combinations of parameters, being infinitely large), standard errors are reported[12] as `NA` (the R term for "not available", missing). This can happen depending on the data-model combination (e.g., including the covariate-ego effect for a covariate with variance 0; or including some effects that are collinear for any data set, such as the combination of outdegree, transitive triplets, outdegree activity, and balance effects — see Snijders, 2005), or on the combination of data, model and

[11] Earlier recommendations in this manual were lower, '2000 to 4000'. This is adequate in many cases; the increase to 5000 has the purpose to be on the safe side.

[12] Since version 1.1-285.

parameters (when a parameter value was given or was reached where the model is not sensitive to some combination of parameters). The remedy here usually is to drop some of the effects.

When siena finds near-perfect collinearity at the end of the estimation algorithm, it will give a warning message of the following kind.

```
*** Standard errors not reliable ***
The following is approximately a linear combination
for which the data carries no information:
0.5 * beta[8] + -0.41 * beta[9] + 1 * beta[14]
It is advisable to drop one or more of these effects.
```

This is based on automatic detection of linear dependencies between the terms in the objective function (or linear predictor) (6). The coefficients are rounded. This warning message signals that, very probably, the combination of the effects mentioned causes the multicollinearity. This multicollinearity is signaled for the parameter values as obtained at the end of the algorithm.

In cases with strong but not complete multicollinearity, i.e., correlations between some parameter estimates (or some of their linear combinations) being close but not equal to –1 or +1, the estimated standard errors are less reliable. Estimates for these correlations are given under the heading `Covariance matrix of estimates (correlations below diagonal)` in the output file, and in the summary(...) of the estimation results (see Section 7.8.3). Strong collinearity may in practice lead to large differences between the estimated standard errors, and also to considerable differences between the parameter estimates, when comparing the results produced by different estimation runs. The remedy is to reduce the model to a more parsimonious one by excluding non-significant effects of which the parameter estimates are highly correlated with others.

Provisionally (until further experience has been collected), the following may be a reasonable guideline. High parameter correlations with the outdegree effect are not a reason for worry, but high parameter correlations with other effects are a reason for checking the stability of the estimated standard errors. The threshold for finding a parameter correlation 'too high' in this respect can be quite high, such as 0.95 (or 0.90). In cases of high parameter correlations, estimating the model twice (or more) and considering the stability of the standard errors will be a good way for seeing whether there are reasons for special caution. If the standard errors are stable, then parameter correlations above 0.90 still can be acceptable – in particular, when they are obtained for parameters that are significantly different from 0 and of which estimates as well as standard errors are stable across repeated runs of the estimation algorithm. Also see the preceding section about the stability of standard errors.

8.2 Precision of the finite differences method

The implementation of the finite differences method is not scale-invariant[13], with the result that the standard errors produced by this method are not very reliable if they are of the order of 0.02 or less.

[13]The scales are determined by the variable z$scale in function robmon. A better procedure would be to set the scale adaptively, but the finite differences method is hardly ever used any more, having been superseded by the score function method, and therefore this improvement has not been effectuated.

9 Tests

Estimated parameters in RSiena can be of three kinds: basic rate parameters, indicated by the letter ρ (rho), and specific for a dependent variable – period combination; other parameters for the rate function (which may be absent from the model), indicated by the letter α (alpha); and parameters that are weights for the objective function, indicated by β (beta). The basic rate parameters are necessarily positive, so testing the null hypothesis that they are equal to 0 makes no sense. For a combined notation, we denote the combination of α and β parameters by the letter θ (theta); for non-conditional estimation, θ will also include the basic rate parameters ρ.

For these parameters, two types of test are available in SIENA.

1. t-type tests of single parameters can be carried out by dividing the parameter estimate by its standard error. Under the null hypothesis that the parameter is 0, these tests have approximately a standard normal distribution. These may also be called Wald-type tests. Section 9.1 indicates how to construct multi-parameter tests from the same principle.

2. Score-type tests of single and multiple parameters are described in Section 9.2.

In addition, there are procedures for assessing goodness of fit as explained in Section 6.15.

9.1 Wald-type tests

'Wald test'[14] is the fancy name for the usual t-test when based on Maximum Likelihood estimates; or F-test for multivariate parameters. For the Stochastic Actor-Oriented Model when the Method of Moments is used, we use the term 'Wald-type tests'. These are based on the parameter estimates and their covariance matrix. Recall that the variances of the parameter estimates are on the diagonal of this covariance matrix, and the standard errors are the square roots of these diagonal elements.

In Section 7.6 we saw that, for a sienaFit object `ans`, the estimates are given in `ans$theta`, the covariance matrix in `ans$covtheta`, and the standard errors in `ans$se`. For testing the null hypothesis that component k of the parameter vector is 0,

$$H_0 : \theta_k = 0,$$

the t-test is based on

$$\frac{\hat{\theta}_k}{\text{s.e.}(\hat{\theta}_k)} = \texttt{ans\$theta[k] / ans\$se[k]} \quad . \tag{9}$$

This can be easily calculated by hand from the RSiena results. It can also be obtained, e.g., from

```
test_parameter(ans, tested=k)
```

where `k` is the sequence number of the effect. The function test_parameter can be used for testing multiple parameters simultaneously but, like in this example, can be used also for a single parameter. The well-known representation of p-values by asterisks is available from

[14] Abraham Wald (1902–1950)

```
siena_table(ans, type="html", sig=TRUE)
```

In some cases, however, the t-statistic (9) does not have an approximate standard normal distribution under the null hypothesis, so that this test is not appropriate. This is the so-called Donner-Hauck phenomenon, named after Hauck and Donner (1977), who first drew attention to this phenomenon in the case of logistic regression. It is discussed also in Geyer and Thompson (1992, section 1.6) and Albert and Anderson (1984). For logistic regression this phenomenon occurs when there is complete, or almost complete, separation of the set of observed values for the vector of predictor variables such that this set consists of values in two half-spaces, where for one of these half-spaces the dependent variable always is 0, and for the other half-space always 1. The data then indicates that the parameter should be very large in absolute value, but not how large; mathematically, the estimated value may be infinite. The parameter as estimated by a practical algorithm then will be not infinite but large, and the standard error also will be large; but the point is that ratio (9) does not need to be large, and the Wald test may be non-significant while the data flatly contradicts a hypothetical parameter value of 0. In Section 7.12.2 we proposed that in such cases, it may be helpful to fix the parameter at some large value, without estimating it. Or, when it is being estimated and the overall maximum convergence ratio is small so convergence is judged as good, this estimate can be used, but in these cases, not its standard error. One possibility that then is available to test the significance is to make a second estimation run in which the parameter is fixed at the value 0, corresponding to the null hypothesis, and test this null value using the score-type test of Section 9.2. See the script `RscriptSienaMultiple.R` on the SIENA webpage for an example.

9.1.1 Multi-parameter Wald tests

For testing the null hypothesis that several parameters are 0, the Wald-type test can also be applied. The function test_parameter produces the chi-squared value of the test statistic, the number of degrees of freedom, and the p-value.

As an example, for the klas12b data the following results were obtained, collected in the sienaFit object `ans`.

```
                                      Estimate   Standard   Convergence
                                                   Error      t-ratio
Rate parameters:
  0.1      Rate parameter period 1      8.7995  ( 1.7061   )
  0.2      Rate parameter period 2      7.6486  ( 1.2754   )
  0.3      Rate parameter period 3      8.2209  ( 1.3090   )

Other parameters:
  1.  eval outdegree (density)         -1.9823  ( 0.1374   )   -0.0200
  2.  eval reciprocity                  1.6659  ( 0.2548   )    0.0153
  3.  eval transitive triplets          0.3948  ( 0.0504   )   -0.0225
  4.  eval transitive recipr. triplets -0.4540  ( 0.0962   )   -0.0276
  5.  eval primary                      0.6062  ( 0.1372   )   -0.0646
  6.  eval sex alter                    0.0732  ( 0.1331   )    0.0069
  7.  eval sex ego                      0.3323  ( 0.1347   )   -0.0526
```

```
  8.  eval sex similarity               0.8828  ( 0.1348   )    0.0525

Overall maximum convergence ratio:    0.1366
```

The output of `summary(ans)` also contains the covariance matrix of the estimates:

```
Covariance matrix of estimates (correlations below diagonal)
   0.019 -0.023 -0.006  0.008 -0.002  0.001 -0.001 -0.004
  -0.666  0.065  0.009 -0.020 -0.001 -0.001  0.002 -0.003
  -0.817  0.664  0.003 -0.004  0.000  0.001  0.000  0.000
   0.620 -0.795 -0.861  0.009 -0.001 -0.001  0.000  0.000
  -0.083 -0.023  0.063 -0.104  0.019  0.000  0.000  0.002
   0.045 -0.019  0.107 -0.097  0.020  0.018 -0.009 -0.001
  -0.035  0.059 -0.073  0.012 -0.021 -0.481  0.018  0.005
  -0.220 -0.099  0.054 -0.012  0.101 -0.049  0.255  0.018
```

You probably do not need this as a user, but this covariance matrix is used by RSiena to carry out the tests. To test the null hypothesis, e.g., that the three sex effects (ego, alter, similarity) are zero, the command

```
test_parameter(ans, tested=6:8)
```

produces the following information:

```
Tested effects:
 friends: sex alter
 friends: sex ego
 friends: sex similarity
chi-squared = 45.66, d.f. = 3;  p < 0.001.
```

The value $p < 0.001$ expresses strong evidence that the network dynamics depends on sex.

If only one parameter is tested by this function, also the one-sided statistic is given. For example, the effect of knowing each other from primary school can be tested by

```
test_parameter(ans, tested=5)
```

with the result

```
Tested effects:
 friends: primary
chi-squared = 19.52, d.f. = 1; one-sided Z = 4.42;  p < 0.001.
```

The Z-value is just the square root of the chi-squared, but with the sign of the parameter attached.

(Here we use the term "Z-value"; elsewhere we have used the term "t-ratio"; for practical purposes these terms are equivalent here, because we do as if the number of degrees of freedom is infinite, so that the standard normal distribution and the t-distribution are the same.)

9.1.2 Tests of linear combinations

The principle of the Wald-type test is more general. It can be used to test null hypotheses that can be represented as a linear constraint on θ:

$$H_0 : A\,\theta = 0,$$

where A is a $r \times p$ matrix, the dimension of θ being p. This can be requested by

```
test_parameter(ans, tested=A)
```

Consider the example that an evaluation and an endowment/maintenance function are included in the model. This situation was explained in Section 2.1.2 and further elaborated in Section 13.1.4. Equations (31) and (32) imply that, if there is no creation function, the split between creation and maintenance can be made as follows:

- creation of ties: evaluation parameter;
- maintenance of ties: evaluation parameter + maintenance parameter.

In an example using the Glasgow data (160 students, with joiners and leavers), we obtained the following results (the LaTeX table was made by the function siena_table). The name of the `sienaFit` object was `G160_net_4`.

Effect	par.	(s.e.)
constant friendship rate (period 1)	11.158	(1.213)
constant friendship rate (period 2)	8.932	(0.898)
outdegree (density)	–2.984***	(0.093)
reciprocity	4.338***	(0.433)
reciprocity maintenance	–1.941**	(0.700)
GWESP I $\rightarrow$ K $\rightarrow$ J (69)	3.428***	(0.311)
GWESP I $\rightarrow$ K $\rightarrow$ J (69) maintenance	–3.537***	(0.681)
rec.degree - activity	–0.262***	(0.028)
sex alter	–0.110	(0.082)
sex ego	0.062	(0.088)
same sex	0.480***	(0.079)
reciprocity x GWESP I $\rightarrow$ K $\rightarrow$ J (69)	0.198	(0.313)

† $p < 0.1$; * $p < 0.05$; ** $p < 0.01$; *** $p < 0.001$;
convergence t ratios all < 0.04.
Overall maximum convergence ratio 0.06.

For creation of new ties, in this model only the evaluation parameters should be considered. For maintenance of existing ties, equation (32) says that the evaluation parameter and the maintenance parameter should be added. This leads to a contribution of $4.338 - 1.941 = 2.397$ for reciprocity and $3.428 - 3.537 = -0.109$ for transitivity (gwespFF). Are these significant?

This can be answered by using the method explained above with a one-dimensional 1×12 matrix

$$A = (0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0)$$

for reciprocity. We can use commands

```
A <- matrix(0, 1, 12)
A[1,4] <- 1
A[1,5] <- 1
test_parameter(G160_net_4, tested=A)
```

The result is that the standard error of the sum of the two estimated parameters is 0.37 and the Z value (standard normal test statistic) is $Z = 6.45$, highly significant.

For the transitivity effect on tie maintenance the matrix

$$A = (0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0)$$

should be used and the results are 0.39 and $Z = -0.28$.

The mathematical background is the following. If a is a vector with coefficients of the linear combination $a' \theta$, in which we are interested, then

```
sum(a*ans$theta)
```

is the corresponding estimate,

```
a %*% ans$covtheta %*% t(a)
```

is the variance, and

```
sqrt(a %*% ans$covtheta %*% t(a))
```

is the standard error.

In this example, a further question is whether the effect of reciprocation on creation of ties is equal to its effect on maintenance of ties. This is simply tested by the parameter θ_5, which —as is implied by the above— is just the difference between the effects of reciprocation on maintenance and on creation of ties. In this case, the difference is –1.941, with t-ratio $t = -1.941/0.700 = -2.77$, $p < 0.005$. We may conclude that, in this model, the effect of reciprocation on creation of ties is significantly (two-sided $p < 0.005$) larger than on maintenance of ties; but the latter, on itself, still is significant.

9.1.3 Tests of equality of parameters

The hypothesis that two parameters in one model have the same value can also be tested using the principle of the multi-parameter Wald test. This is implemented for this particular case by the function test_parameter with argument `method="same"`. E.g., if `ans` is the estimated `sienaFit` object and we wish to test the null hypothesis that parameters with index numbers 9 and 10 are equal, we use

```
test_parameter(ans, method="same", tested=9, tested2=10)
```

9.2 Score-type tests and one-step estimates

The Wald test is based on the estimate for the parameter, and thereby integrates estimation and testing. Sometimes, however, it can be helpful to separate these two types of statistical evaluation. This is the case notably when estimation is instable, e.g., when a model is considered with rather many parameters given the information available in the data set, or when the precise value of the estimate is not determined very well as happens under the Donner-Hauck phenomenon treated in the previous section. The score-type test gives the possibility of testing a parameter without estimating it.

This is done using the generalized Neyman-Rao [15] score test that is implemented for the Method of Moments estimation method in SIENA as a part of function siena, following the methods of Schweinberger (2012). For the ML estimation method, following the same steps produces the Rao (1947) efficient score test. Since the name of 'score test' is associated with likelihood-based analysis as in Rao (1947), the test of Schweinberger (2012) that is associated with the Method of Moments is called 'score-type test'.

When using the score-type test, some model is specified in which one or more parameters are restricted to some constant, in most cases 0 – these constant values define the null hypothesis being tested. This can be obtained in RSiena by appropriate choices in the effects dataframe. Parameters can be restricted by putting `TRUE` in the fix and test columns, and the tested value in the initialValue column. The function set_effect is available to do this. For example, a score test for the evaluation effect of transitive ties in a network can be requested as follows.

```
myeff <- set_effect(myeff, transTies, initialValue=0,
                    fix=TRUE, test=TRUE)
ans <- siena(data=mydata, effects=myeff)
```

Suppose that the effect of transitive ties has number 10 in the output; then

```
st <- test_parameter(ans, method="score", tested=10)
```

will produce the score-type test.

The score-type test proceeds by simply estimating the restricted model (not the unrestricted model, with unrestricted parameters) by the standard SIENA estimation algorithm. The t-ratios for convergence can be disregarded for the parameters that are fixed and estimated by the score test, as convergence is not an issue for these parameters.

It should be noted that using the prevAns option in siena overrides the initial values in the effects object, so that using siena for an effects object with `fix = TRUE` for some of the effects, jointly with the prevAns option, will lead to score-type tests of hypothesized values as given by the prevAns option. Since the presentation of the results includes the hypothesized value, there is no reason for doubt as to what has been done. However, mostly this is not what is desired, and therefore it often will be preferable to proceed as follows. First update the initial values using update_theta, then set the hypothesized value by set_effect, and then carry out the estimation and score-type test by siena. The following is an example, assuming that an earlier reasonable estimate was found in sienaFit object `myans0`, and the user wishes to use this as starting values.

[15] Jerzy Neyman, 1894-1981; Calyampudi Radhakrishna Rao, 1920-2023

```
myeff  <- update_theta(myeff, myans0)
myeff  <- set_effect(myeff, transTies, initialValue=0,
                     fix=TRUE, test=TRUE)
myans1 <- siena(data=mydata, effects=myeff)
summary(myans1)
```

9.2.1 One-step estimates

Related to the score-type test are the so-called *one-step estimates*. These are 'quick and easy' estimates which are not reliable, but which can be used as approximations of the unrestricted estimates (that is, the estimates that would be obtained if the model were estimated once again, but without restricting the parameter by `fix = TRUE`). They are explained mathematically in Schweinberger (2012). Their computational advantage is based on the fact that they require simulations only for parameter vectors where this parameter is fixed. Often, when parameters are fixed, they are fixed to 0; but other values could be used, and this would be preferable if they are closer to an appropriate value for this parameter.

The one-step estimates are more reliable according as the significance of the fixed-and-tested effects is smaller and there is less collinearity between the estimated and the fixed parameters.

Continuing the above example, the one-step estimates can be obtained from

```
test_parameter(ans, method="score", tested=NULL, onestep=TRUE)
```

Sometimes, e.g., if there are strong collinearities between the fixed parameters, some or many of the one-step estimates are large (larger than 10 in absolute value). Then they are useless – although they may be useful to indicate which parameters are collinear with each other.

If the one-step estimates have reasonable values, an alternative to the procedure mentioned above (leading to `myans1`) is to put the one-step estimates in the effects object. E.g., if answer object `ans` was obtained using effects object `myeff` and there was only one fixed-and-tested effect, with number 10, the commands that could be given are

```
myeff2  <- update_theta(myeff, ans, onestep=TRUE, freed=10)
(myans2 <- siena(data=mydata, effects=myeff2))
```

9.3 Alternative application: convergence problems

An alternative use of the score test statistic is as follows. When convergence of the estimation algorithm is doubtful, it is sensible to restrict the model to be estimated. Either 'problematic' or 'non-problematic' parameters can be kept constant at preliminary estimates (estimated parameters values). Though such strategies may be doubtful in at least some cases, it may be, in other cases, the only viable option besides simply abandoning 'problematic' models. The test statistic can be exploited as a guide in the process of restricting and estimating models, as small values of the test statistic indicate that the imposed restriction on the parameters is not problematic. As mentioned above, the one-step estimates can be a guide to indicate which parameters are collinear.

9.4 Testing differences between independent groups or periods

Sometimes it is interesting to test differences between parameters estimated for independent groups. For example, for work-related support networks analyzed in two different firms, one might wish to test whether the tendency to reciprocation of work-related support, as reflected by the reciprocity parameter, is equally strong in both firms. Such a comparison is meaningful especially if the total model is the same in both groups, as control for different other effects would compromise the basis of comparison of the parameters.

If the parameter estimates in the two networks are $\hat{\beta}_a$ and $\hat{\beta}_b$, with standard errors $s.e._a$ and $s.e._b$, respectively, then the difference can be tested with the test statistic

$$\frac{\hat{\beta}_a - \hat{\beta}_b}{\sqrt{(s.e_a)^2 + (s.e_b)^2}}\,, \tag{10}$$

which under the null hypothesis of equal parameters, $\beta_a = \beta_b$, has an approximating standard normal distribution.

The same method can be used for comparing estimates obtained from separate analyses for different periods from a data set with three or more waves. For example, if there are three waves and separate analyses were done for period 1 (wave 1 – wave 2) and period 2 (wave 2 – wave 3), then $\hat{\beta}_a$ could be an estimate obtained for period 1 while $\hat{\beta}_b$ would be the estimate for the same parameter for period 2. This is allowed because the analysis of period 2 conditions on wave 2.

9.5 Testing time heterogeneity in parameters

The model assumes that the parameter values β_k of the objective function are constant over the periods from 1 to $M-1$. This can be tested by the test_time function, as explained in this section.

We initially assume that β does not vary over time, yielding a *restricted model*. Our data contains $|\mathcal{M}|$ observations, and we estimate the restricted model the method of moments. We wish to test whether the *restricted model* is misspecified with respect to time heterogeneity. Formally, define a vector of time dummy terms $\mathbf{h}$:

$$h_k^{(m)} = \begin{cases} 1 & \{m : w_m \in \mathcal{W}, m \neq 1\} \\ 0 & \text{elsewhere} \end{cases}, \tag{11}$$

where k corresponds to an effect included in the model.[16] The explanation here is formulated for the network evaluation function, but the principle can be applied more generally. An *unrestricted model* which allows for time heterogeneity in all of the effects is considered as a modification of (15):

$$f_{ij}^{(m)}(\mathbf{x}) = \sum_k \left(\beta_k + \delta_k^{(m)} h_k^{(m)}\right) s_{ik}\big(\mathbf{x}(i \rightsquigarrow j)\big) \tag{12}$$

where $\delta_k^{(m)}$ are parameters for interactions of the effects with time dummies. One way to formulate the testing problem of assessing time heterogeneity is the following:

$$\begin{aligned} H_0 &: \delta_k^{(m)} = 0 \text{ for all } k, m \\ H_1 &: \delta_k^{(m)} \neq 0 \text{ for some } k, m. \end{aligned} \tag{13}$$

This testing problem can be addressed by the score test in a way that no extra estimation is necessary. This method was elaborated and proposed by Lospinoso et al. (2011) and is implemented in RSiena. To apply the test to your dataset, run an estimation in the usual way, e.g. as follows (we specify `nsub = 2, n3 = 100` just to have an example that runs very quickly):

```
myalgorithm <- set_algorithm_saom(nsub = 2, n3 = 100)
mynet1      <- as_dependent_rsiena(array(c(s501, s502, s503),
                                         dim = c(50, 50, 3)))
mydata      <- make_data_rsiena(mynet1)
myeff       <- make_specification(mydata)
myeff       <- set_effect(myeff, list(transTrip, outAct))
ans2        <- siena(data=mydata, effects=myeff,
                     control_algo=myalgorithm)
```

and conduct the timetest through

```
## Conduct the score type test to assess whether heterogeneity is present.
tt2 <- test_time(ans2)
plot(tt2, effects = 1:2)
```

[16] The dummy $\delta_k^{(1)}$ is always zero so that period w_1 is (arbitrarily) considered the reference period.

When time heterogeneity is found, there are several options for how to continue. One option is to break apart the set of periods, and do the analysis separately for smaller subsets of waves. The extreme of this is to do a wave-by-wave analysis, i.e., an analysis for each pair of consecutive waves. Then there is no issue of heterogeneity, and the results will show in some cases that the heterogeneity affects only one or a few parameters. Depending on the non-constant pattern that is shown by some of the parameters, a new time-varying actor variable, or several, can be defined to capture this heterogeneity, and the analysis may be done for the set of all periods, for a model including interactions of this time varying covariate with the effects for which heterogeneity was found. This is the second option. The time-varying actor variable could be a linear function of time, a time dummy, or whatever. The summary of test_time gives information that can be used to see which waves, and which effects, are mainly responsible for the time heterogeneity. A wave-by-wave analysis may give this information more clearly, but is more work.

One possibility for time-varying covariates is a time dummy variable. This is available in an automated way in the function includeTimeDummy. An example is

```
myeff <- includeTimeDummy(myeff, recip, outAct, timeDummy = "2")
ans3   <- siena(data=mydata, effects=myeff)
```

after which you can test the new model again,

```
tt3 <- test_time(ans3)
```

and so on.

At the SIENA website there is a script `RscriptSienaTimeTest.r` with an extensive example.

10 Simulation

The simulation option simulates the network evolution for fixed parameter values. This is meaningful, e.g., for theoretical exploration of the model, for goodness of fit assessment, and for studying the sensitivity of the model to parameters. Simulations are produced by the siena function also used for parameter estimation, but by calling it in such a way that only Phase 3 is carried out (see section 7.7). This is done by requesting `nsub = 0` in the model specification in function set_algorithm_saom. By also requesting `simOnly = TRUE` the calculation of standard errors, which usually is not meaningful when simulating without estimating, is suppressed. Mostly it is more meaningful to do this for non-conditional simulation (hence `cond = FALSE`), and a two-wave data set, so that the simulations are totally determined by the parameters and the first observation. The second wave then must be present in the data set only because RSiena requires it for estimation, and here we are using a function that is originally meant for estimation. The principle is to use an 'estimation' algorithm

```
alg_alg  <-  set_algorithm_saom(cond=FALSE,
                                nsub=0, n3=1000, simOnly=TRUE)
```

(where a different value than 1000 for the number of runs may be used); but this has to be further specified.

There are three ways to determine the parameter values that will be used in the simulations. One is to use the parameter values that are present in the effects object. Then we use

```
alg_alg  <-  set_algorithm_saom(cond=FALSE,
               useStdInits=FALSE, nsub=0, n3=1000, simOnly=TRUE)
sim_ans  <- siena(data=mydata, effects=myeff, control_algo=alg_alg)
```

Parameter values are obtained from the effects object, because of the option `useStdInits = FALSE`. If the name of the effects object is `myeff`, the current parameter values are obtained from requesting

```
myeff
```

and different values can be specified by the function set_effect.

The second way to determine the parameter values for simulation is that these are the parameters in an available estimated model `ans`. This can be done using

```
alg_alg  <-  set_algorithm_saom(cond=FALSE,
               nsub=0, n3=1000, simOnly=TRUE)
sim_ans  <- siena(data=mydata, effects=myeff,
               control_algo=alg_alg, prevAns=ans)
```

The value of `useStdInits` does not matter if the argument `prevAns` is used for siena.

A third possibility is, for co-evolution models, to use different estimated models for the various dependent variables (networks, behavior). This can be achieved by using function updateTheta to get the estimated parameters in the effects object and then work with `useStdInits = FALSE`. See the help page for updateTheta.

When artificial data sets are generated that have a close link to observed data, the restriction that simulations follow the monotonicity patterns that might be present in the data (see Section 5.2.4) can be undesirable. This restriction can be lifted by using `allowOnly = FALSE` in the call of as_dependent_rsiena (see the help file for this function). This parameter will then set any `uponly` and `downonly` flags to `FALSE`, precluding monotonicity constraints.

The statistics generated, which are the statistics corresponding to the effects in the model, can be accessed from the sienaFit object produced by siena. Denoting the name of this object by `sim_ans`, its component `sim_ans$sf` contains the generated deviations from targets. As discussed also in Section 7.6, the statistics can be recovered from the deviations and the targets as follows.

```
# To get the generated statistics without subtracting the targets,
# we have to add the targets to the deviations.
# To do this, repeated transposition t can be used:
stats       <-  t(t(sim_ans$sf) + sim_ans$targets)
# Calculate means and covariance matrix:
v           <-  apply(stats,2,mean)
covsf       <-  cov(stats)
# covsf is the same as sim_ans$msf
```

Of course, any other distributional properties of the generated statistics can also be obtained by the appropriate calculations and graphical representations in R.

10.1 Accessing the generated networks

If one is interested in the networks generated, not only in the statistics internally calculated, then the entire networks can be accessed. This is done by using the returnDeps option, as follows.

```
sim_ans  <- siena(data=mydata, effects=myeff, returnDeps=TRUE)
```

The `returnDeps = TRUE` option attaches a list `sim_ans$sims` containing all simulated networks as edge lists to the `sim_ans` object. This uses rather a lot of memory. Since here the default `n3 = 1000` was used, `sim_ans` will be a list of 1000 elements; e.g., if there are three waves, the 568'th network generated for wave 2 is given by

```
sim_ans$sims[[568]][[1]][[1]][[1]]
```

The numbering is as follows: first the number of the simulation run (here, arbitrarily, 568); then the number of the group as defined in Section 12.2 (1 in the usual case of single-group data structures); then the number of the dependent variable (here 1, because it is supposed that there only is a dependent network); then the period number (i.e., wave number minus 1). This type of information can be found out by requesting

```
str(sim_ans$sims[[568]])
```

In the case of maximum likelihood estimation, if there is only one group and two waves, the list structure is simpler. Then the two superfluous list levels, group and wave/period, are omitted. The list then has three levels: dependent variable within group within simulation run.

The help page for siena contains an example for how to access the generated networks in the case of ML estimation. Note that this is meaningful only in view of the missing values of dependent variables for waves 2 and further, for which the simulations in Phase 3 can be regarded as a kind of model-based imputation.

Section 5.1.2 explains how such an edgelist can be transformed to an adjacency matrix:

```
# Determine number of actors (normally the user will know this)
n <- length(mydata$nodeSets[[1]])
# create empty adjacency matrix
adj <- matrix(0, n, n)
# Make shorter notation for edge list
edges <- sim_ans$sims[[856]][[1]][[1]][[1]]
# put edge values in desired places
adj[edges[, 1:2]] <- edges[, 3]
```

As an example, the following commands turn this list into a list of edgelists according to the format of the sna package (Butts, 2008), and then calculate the maximum k-core numbers in the networks. This assumes that a one-mode network is being analyzed.

```
# First define a function that extracts the desired component
# from the list element,
# gives the column names required for sna edgelists,
# and adds the attribute defining the number of nodes in the graph,
# as required by sna.
make.edgelist.sna <- function(x, n)
                     {
                         x <- x[[1]][[1]][[1]]
                         colnames(x) <- c("snd", "rec", "val")
                         attr(x, "n") <- n
                         x
                     }
# Note: if there is only one group and two waves,
# x[[1]][[1]][[1]] should be replaced by x[[1]]
# Apply this function to the list of simulated networks
simusnas <- lapply(sim_ans$sims, make.edgelist.sna, 50)
# Define a function that calculates the largest k-core number in the graph
library(sna)
max.kcores <- function(x)max(kcores(x))
# Apply this function and make a histogram
mkc <- sapply(simusnas, max.kcores)
hist(mkc)
```

Another possibility is to use the extractor functions, sparseMatrixExtraction, networkExtraction, or behaviorExtraction that are also used for test_gof.

10.2 Conditional and unconditional simulation

The distinction between conditional and unconditional simulation is the same for the simulation as for the estimation option of RSiena, described in Section 7.12.1. The choice between conditional and unconditional simulation is made in the set_algorithm_saom function by setting the cond parameter, possibly also the condvarno and condname parameters.

If the conditional option is chosen, then the simulations carry on until the desired distance is achieved on the dependent variable used for conditioning. For networks, the distance is the number of differences in the tie variables; for behavioral variables, the sum across actors of the absolute differences. This is determined as the distance between the consecutive networks (or, behaviors, if such a variable is used for conditioning) given in the call of make_data_rsiena. The rate parameter for this dependent variable then has no effect.

If the conditional simulation option was chosen (which is the default) and the simulations do not succeed in achieving the condition required by its stopping rule (see Section 7.12.1), then the simulation is terminated with an error message, saying *Unlikely to terminate this epoch*. In this case, you are advised to change to unconditional simulation.

10.3 Simulating with variable parameter values

The procedures described until now in this chapter conduct the simulations in Phase 3 with a constant parameter vector. If `nsub = 0` this parameter value is given in the effects object, and returned for the resulting sienaFit object `ans` as `ans$theta`. It is also possible to use a different parameter vector in every run of Phase 3. This is done by specifying a matrix of parameter values (runs in the rows, parameters in the columns) as the argument `thetaValues` for siena. This requires specifying `simOnly = TRUE`. The parameter values actually used are then returned in `ans$thetaUsed`.

If parallel processes are used (`useCluster = TRUE`) and the number of processes is `nbrNodes` > 1, the parameters used will be constant in each set of `nbrNodes` consecutively stored simulations. If the number of rows of `thetaValues` is not large enough given the values of `n3` and `nbrNodes`, the last row of `thetaValues` will continue to be used. All this will be reflected by the value of `ans$thetaUsed`.

11 Getting started

Note that there is a section 'Getting started' in the Minimal Introduction: Section 2. It may be best to go through that section first.

The best way to get started is to download the R scripts from the SIENA website
https://www.stats.ox.ac.uk/~snijders/siena/siena_scripts.htm
and start reading and playing with them. There is one script, basicRSiena.r, which is a very short general introduction. Then there is a collection of five scripts:

1. Rscript01DataFormat.R
2. RSienaSNADescriptives.R
3. Rscript02SienaVariableFormat.R
4. Rscript03SienaRunModel.R
5. Rscript04SienaBehaviour.R

which together are meant as an introduction to get started. As the names suggest, the second of these does not use RSiena, but package sna.

As a more general manner to find your own way, you can do the following. For carrying on and getting a first acquaintance with your own running of the model, the data set collected by Gerhard van de Bunt is useful; this data is discussed extensively in van de Bunt (1999); van de Bunt et al. (1999), and used as example also in Snijders (2001) and Snijders (2005). The data files are provided with the program and at the SIENA website. The digraph data files used are the two networks vrnd32t2.dat, vrnd32t4.dat. The networks are coded as 0 = unknown, 1 = best friend, 2 = friend, 3 = friendly relation, 4 = neutral, 5 = troubled relation, 6 = item non-response, 9 = actor non-response. Recode the network so that values 1, 2, and 3 are interpreted as ties for the first as well as the second network, and values 6 and 9 are missing data codes (NA).

The actor attributes are in the file vars.dat. Variables are, respectively, gender (1 = F, 2 = M), program, and smoking (1 = yes, 2 = no). See the Data sets tab at the SIENA website, and the references mentioned above for further information about this network and the actor attributes.

Create the various required objects, using functions make_data_rsiena, make_specification, and set_algorithm_saom, as indicated in Chapters 5 and 7. At first, leave the whole model specification as it is by default (see Section 6): a constant rate function, the out-degree effect, and the reciprocity effect.

Then let the program estimate the parameters, using function siena. If you are using Windows, you will see a screen with intermediate results: current parameter values, the differences ('deviation values') between simulated and observed statistics (these should average out to 0 if the current parameters are close to the correct estimated value), and the quasi-autocorrelations discussed in Section 7. (If you are running RSiena from RStudio, this screen is visible when you click on the little feather that appears in the taskbar when the estimation is running.) It is possible to intervene in the algorithm by clicking on the appropriate buttons: the algorithm may be restarted or terminated. In most cases this is not necessary.

A little bit of patience is needed to let the machine complete its three phases. When the algorithm has finished, look at the results in the output file or by the print or summary function of the resulting sienaFit object. Check that the overall maximum convergence ratio is small enough (ideally less than .25). If not, continue estimation with the prevAns option as discussed in Section 7.3.2. When satisfactory convergence has been obtained, make sense of the results: for example, is the reciprocity parameter significant?

As further steps, include some extra effects. First candidates are the transitive triplets effect or the gwesp (shortName `gwespFF`) effect, and any effects that may follow from your theories about the investigated network (see, e.g., Section 6); you can find their shortName, needed to specify them, in Chapter 13, where also the mathematical specifications are given. When these new effects have been added, follow the same steps: estimate, check convergence, if this is not yet satisfactory estimate again with the new initial values, and interpret the results when converged has been obtained.

Note that in Chapter 13 all shortNames are given within parentheses, so that if you are looking for the definition of a particular shortName, e.g., `gwespFF`, you can search in the manual for the string `(gwespFF)`. By the way, it is much more convenient and energy-friendly to download the manual on your own computer and read it there, then to keep it for reading and searching at the internet.

To continue, non-significant effects may be excluded (but it is advised always to retain the out-degree and the reciprocity effects) and other effects may be included, as suggested in Section 6.

11.1 Model choice

For the selection of an appropriate model for a given data set it is best to start with a simple model (including, e.g., 2 or 3 effects), delete non-significant effects, and add further effects in groups of 1 to 3 effects. Like in regression analysis, it is possible that an effect that is non-significant in a given model may become significant when other effects are added or deleted!

When you start working with a new data set, it is often helpful first to investigate the main endogenous network effects (reciprocity, transitivity, etc.) to get an impression of what the network dynamics looks like, and later add effects of covariates. The most important effects are discussed in Section 6; the effects are defined mathematically in Chapter 13.

Approaches to model specification are presented in Chapter 6, in Snijders et al. (2010b), and in
`https://www.stats.ox.ac.uk/~snijders/siena/Siena_ModelSpec_s.pdf`.

When the distribution of the out-degrees is fitted poorly (which can be inspected using the test_gof function of Section 6.15), an improvement usually is possible either by including non-linear effects of the out-degrees in the evaluation function, or by other improvements of the model. This totally depends on the data set at hand.

12 Multilevel network analysis

For combining RSiena results of several independent networks, there are four options. ('Independent' networks here means that the sets of actors are disjoint, and it may be assumed that there are no direct influences from one network to another.) The first two options assume that the parameters of the actor-based models for the different networks are the same – except for the basic rate parameters and for those differences that are explicitly modeled by interactions with dummy variables indicating the different networks. All but the second option require that the number of observations is the same for the different networks. These methods can be applied for two or more networks.

In the following discussion, the terms 'networks' and 'groups' are used interchangeably.
The four options are:

1. Combining the different networks in one large network, indicating by structural zeros that ties between the networks are not permitted. This is explained in Section 5.3.1.
   The special effort to be made here is the construction of the data files for the large (combined) network. This is superfluous; it is better to use option 2.

2. Combining different groups into one *multi-group* data set, and analyzing this by siena. The 'groups' are the same as the 'different networks' mentioned here. This is explained in Section 12.2.
   A difference between options 1 and 2 is that the use of structural zeros (option 1) will lead to a default specification where the rate parameters are equal across networks (this can be changed by making the rate dependent upon dummy actor variables that indicate the different networks) whereas the multi-group option yields rate parameters that are distinct across different networks.
   In this option (like in option 1), the assumption is made that all parameters are the same for the various networks, except for the basic rate parameters; and except for explicitly specified interaction effects between variables depending on the group, and other effects. This is a strong assumption, and usually not very desirable if options 3-4 are possible.
   This option can be applied, e.g., if there are important changes of composition between waves. If there are three waves, and the sets of actors at these waves are N_1, N_2, and N_3, respectively, then the intersections of the node sets $N_{12} = N_1 \cap N_2$ and $N_{23} = N_2 \cap N_3$ can be utilized to form two longitudinal data sets each with two waves. These can then be combined in a multi-group data set and analyzed by the multi-group option. This can be straightforwardly extended to more than three waves.
   This is the "inclusive" analytic strategy proposed by de la Haye et al. (2017).

3. Analyzing the different networks separately, without any assumption that parameters are the same but using the same model specification, and post-processing the output files by a meta-analysis using meta_siena. This is explained in Section 12.3.

4. Combining different groups into one *multi-group* data set as in option (2), but analyzing this by multi_siena. This is explained in Section 12.4.

Here the assumption for the parameters is that all basic rate parameters may differ arbitrarily between the groups; for the other parameters, some are identical and others vary randomly across groups according to a multivariate normal distribution. The distinction between 'some' and 'others' here is made by the parameter `random` in function set_effect.

The first and second options will yield nearly the same results, with the differences depending on the basic rate (and perhaps other) parameters that are allowed to differ between the different networks, and of course also depending on the randomness of the estimation algorithm. The second option is more 'natural' given the design of RSiena and will normally run faster than the first. Therefore the second option seems preferable to the first.

The third option makes much less assumptions because parameters are not constrained at all across the different networks. This is preferable to the assumption of homogeneous parameters made in options 1 and 2. The fourth option is a middle ground between the first two and the third. The arguments usual in statistical modeling apply: as far as assumptions is concerned, options (3) and (4) are safer; but if the assumptions are satisfied (or if they are a good approximation), then options (1) and (2) have higher power and are simpler. Option (3) requires that each of the different network data sets is informative enough to lead to well-converged estimates; this will not always be the case for small data sets, and then option (4) may be preferable.

One difference between option 3 and the others is the centering of covariates; if this is used (which is the default!). For actor covariates, the multi-group data sets will center actor covariates by subtracting the grand mean; in the individual data sets analyzed in option 3, groupwise means are subtracted, This will affect the results. In some cases, for a multi-group analysis by option 2 or 4, it will be desirable not to center and/or to also use the groupwise means of covariates as an additional covariate.

When the data sets for the different networks are not too small individually, then a middle ground might be found in the following way. Start with option (3). This will show for which parameters there are important differences between the networks. Next follow option (2), with interactions between the group dummies and those parameters for which there were important between-network differences; or option (4), where the randomness of the effects is determined by these differences.

When the data sets for the different networks are quite small, then one might start by option (2), and use test_time to test for which of the effects especially there is a large variation in parameter values across the groups; next one could follow approach (4), determining the randomness of the effects by the results about this variability.

In all cases, it is probably best to use an identical model specification for the various groups. A problem that may occur especially if the groups are small is that in some of the groups the change of the dependent variable (network or behavior) may be upward only or downward only, which by default then will be regarded by RSiena as a constraint for the simulations, as mentioned in Section 5.2.4. This leads to model differences that in most cases will be undesirable. Therefore it is advisable in the original construction of the datasets to use `allowOnly = FALSE` in the call of as_dependent_rsiena.

12.1 Function make_group_rsiena

The function make_group_rsiena operates on a list of siena data sets created by make_data_rsiena, containing variables with the same names and of the same types. It creates a so-called sienaGroup object, which is a list of siena data objects with some additional information contained in attributes. This is used for options 2 and 4. For option 2 the numbers of waves may differ between the groups, for option 4 the groups all should have the same number of waves.

For centered actor covariates, function make_group_rsiena centers the original values again, now around the overall mean. This allows the use of centered actor covariates that are constant within groups. The `egoX` effect of a centered covariate, e.g., will be the sender effect of the original variable with grand-mean centering. For the `altX` effect the centering makes no difference.

For actor covariates, function make_group_rsiena recalculates attributes `range`, `range2`, and `simMean`. Since version 1.3.4, the attributes `range` and `range2` for each group are set at the range of this covariate in all groups. The `simMean` attribute of a group is set at the average of the similarity values for this group using the overall range. This implies that the parameters of the `simX` effect refer to the similarity transform (1) normalized with respect to the same range r_V for all groups; and the centering of this transform, which affects the parameter of the `outdegree (density)` effect, is group-specific.

12.1.1 Group-level variables

Group-level covariates for actors as well as for dyads can be used without any problem. They can be defined for each group separately, with the same name across groups, and with a constant value within the group. To avoid the warning that would result, the option `warn=FALSE` can be used for the functions as_covariate_rsiena etc.

Group level variables can be used with ego effects (`egoX`) and interactions of ego effects with other effects.

A specific useful type of group-level variable is a set of group dummy variables where each dummy variable has value 1 with a given group and 0 for all other groups. Such dummies, and their interactions, can be used to represent time heterogeneity.

12.2 Multi-group Siena analysis

The multi-group option 'glues' several Siena data sets (further referred to as *groups*) one after the other into one larger multi-group data set. These groups must have the same sets of variables of all kinds: that is, the list of dependent networks, dependent behavioral variables, actor covariates, and dyadic covariates must be the same for the various groups. Also their names must be the same. The number of actors and the number of waves can be different, however.

Gluing the groups together means that the observations (waves) of the groups are arranged after each other, so the total number of observations in the multi-group data set is the sum over all groups of the number of observations per group. In this way the groups then are combined into one Siena data set where the number of actors is the largest of the number of actors of the groups, and the number of observations is the sum of the observations of the groups. Constant covariates (whether monadic or dyadic) are transformed to changing covariates, because the groups now are regarded as multiple observations.

This is done by the function make_group_rsiena which creates a so-called `sienaGroup` object, which is a list of `siena` objects with some additional information.

As an example, suppose that three Siena data sets with names `sub1`, `sub2`, and `sub3` are combined. Suppose `sub1` has 21 actors and 2 observations, `sub2` has 35 actors and 4 observations, and `sub3` has 24 actors with 5 observations. Then the combined multi-group group has 35 actors and 11 observations. The step from observation 2 to 3 switches from group `sub1` to group `sub2`, while the step from observation 6 to 7 switches from group `sub2` to `sub3`. These switching steps do not correspond to simulations of the actor-based model, because that would not be meaningful.

The different groups are considered to be unrelated except that they have the same model specification, the same variable names, and the same parameter values. It is important to check that this is a reasonable assumption. One aspect of this is by looking at the descriptives for change produced by write_report, and checking that the tendencies in the dependent variable or variables, upward/stable/downward, are not too different between the groups. The test_time function can be used for formally testing this assumption. Moderate violations (p-values larger than 0.01) will probably be acceptable in the sense that the combined results still are a meaningful aggregate, strong violations are not acceptable and should be remedied by dropping some of the groups or by including an interaction term.

Given the potentially large number of periods that can be implied by the multi-group option, it probably is advisable, when using Method of Moments estimation, to use the conditional estimation option.

In multi-group groups, individual covariates are centered by subtracting the overall mean (across all groups), but dyadic covariates are centered by subtracting the within-group means.

12.3 Meta-analysis of Siena results

The function meta_siena is a meta-analysis method for SIENA. It combines estimates for a common model estimated for several data sets, that must have been obtained earlier. This function combines the estimates in a meta-analysis or multilevel analysis according to the methods of Snijders and Baerveldt (2003), and according to a Fisher-type combination of one-sided p-values.

The function meta_siena takes as input the sienaFit objects produced by separate runs of siena. These sienaFit objects must have exactly the same model specification and the same names of all variables; but it is allowed that there are differences with respect to parameters being fixed and perhaps tested. To get the same names of variables, the variables must be renamed in the call of make_data_rsiena; an example is in script RscriptMultipleGroups.R at the StOCNET website. If in some but not all groups a dependent variable has only upward or only downward changes, the automatic restriction to follow this pattern also in the simulations (see Section 5.2.4) must be lifted, because this would make the model specifications different. This must be done already in the original construction of the datasets that then later are combined by meta_siena, by using `allowOnly = FALSE` in the call of as_dependent_rsiena, as mentioned in Section 5.2.4.

If there are some parameters that cannot be estimated for some of the data sets (e.g., the effect of sex in a one-gender school; or because of near-multicollinearity), these parameters must still be included in the model for those data sets, but the parameters can be fixed to 0 (and perhaps tested by a score-type test).

Each parameter in the model is treated separately in the meta-analysis, without taking account of the dependencies between the parameters and their estimates. Denote the number of combined data sets by N. If we denote a given parameter (e.g., the coefficient of the reciprocity effect) by θ, then the *true parameter values* for the N data sets are denoted $\theta_1, \theta_2, \ldots, \theta_N$, while their *estimates* are denoted $\hat{\theta}_1, \hat{\theta}_2, \ldots, \hat{\theta}_N$.

The package metafor can also be used for meta-analysis. This package is extensively documented in Viechtbauer (2010). In terms of Viechtbauer (2010), meta_siena follows a random effects approach and presents the Hedges estimator which is the procedure of Snijders and Baerveldt (2003), proposed by Cochran (1954); it also presents the maximum likelihood estimator. In Viechtbauer (2005), an extensive study is made comparing the various approaches, and it turns out that the comparison is not unequivocal. His recommendation, however, is to use the restricted maximum likelihood estimator. Since this is not implemented in meta_siena, this recommendation suggests that one should rather use metafor, with the option `method = "REML"`.

Still another possibility is offered by the package mvmeta (Gasparrini et al., 2012). This was used in conjunction with RSiena by An (2015), who showed how to use this for incorporating group-level explanatory variables, using a fixed effects as well as a random effects approach, the latter with restricted maximum likelihood.

12.3.1 Meta-analysis directed at the mean and variance of the parameters

In this meta-analysis it is assumed that the data sets can be regarded as a sample from a population – i.e., a population of dynamic networks – and accordingly the true parameters θ_j are a random sample from a population. If the number of data sets is small, e.g., less than 20, and especially if this number is less than 10, this assumption is not very attractive from a practical point of view, because the sample then would be quite small so the information obtained about the population is very limited.

The mean and variance in this population of parameters are denoted

$$\mu_\theta = \mathcal{E}\,\theta_j\ ,$$
$$\sigma^2_\theta = \mathrm{var}\,\theta_j\ .$$

Each of these parameters must have been estimated in a run of siena, yielding the estimate $\hat{\theta}_j$, which is the true parameter plus a statistical error E_j:

$$\hat{\theta}_j\ =\ \theta_j + E_j\ .$$

The standard error of this estimate is denoted by s_j.

For each of the parameters θ, the function meta_siena estimates the mean μ_θ and the variance σ^2_θ of the distribution of θ, and tests several hypotheses concerning these 'meta-parameters':

1. Test $H_0^{(0)}:\ \mu_\theta = \sigma^2_\theta = 0$ (all $\theta_j = 0$), i.e., effect θ is nil altogether.
   This is done by means of a chi-squared test statistic T^2 with N d.f.

2. Estimate μ_θ.

3. Test $H_0^{(1)}$: $\mu_\theta = 0$.
   This is done by means of a standard normal test statistic t_{μ_θ}, being the ratio of the estimate for μ_θ to its standard error.

4. Test $H_0^{(2)}$: $\sigma_\theta^2 = 0$, i.e., $\theta_j = \mu_\theta$ for all j.
   This is done by means of a chi-squared test statistic Q with $N - 1$ d.f.

5. Estimate σ_θ^2 .

Two approaches are followed and presented in the output. The first is an iterative weighted least squares method based on Cochran (1954) and Snijders and Baerveldt (2003). The second is a likelihood-based method under the assumption of normal distributions: the estimators are maximum likelihood estimators; the associated confidence intervals are based on profile likelihoods, and therefore will be asymmetric. The reported p-values for the population mean (hypothesis $H_0^{(1)}$) are based on the t distribution with $N - 1$ d.f. In all cases, it is possible that some of the data sets j are dropped for some of the parameters because the standard error s_j is too large (see below); in that case, the number N used here is the number of data sets actually used for the parameter under consideration.

For both of these two approaches, it is assumed that the true deviations $\theta_j - \theta$ and the random errors E_j are uncorrelated. This is not always a plausible assumption; Fisher's combination, mentioned below, does not make this assumption. The plots of estimates versus standard errors, produced by using meta_siena and following it up by plot.sienaMeta, can be used as information about the plausibility of this assumption.

For testing the hypotheses mentioned here, it is also assumed that, given the true parameter values θ_j, the estimates $\hat{\theta}_j$ are approximately normally distributed with mean θ_j and variance s_j^2. This is often a reasonable assumption.

The likelihood-based methods also assume that the true values θ_j are normally distributed in the population. If this is a reasonable approach, the likelihood-based methods are preferable. A disadvantage of the iterative weighted least squares method is that results are possible where the outcome of the test of $H_0^{(2)}$ is significant at a usual level of significance, i.e., σ_θ^2 is thought to be positive, whereas the estimate is $\hat{\sigma}_\theta^2 = 0$. This potential inconsistency is possible because the test and the estimator in this approach are not directly related (cf. Snijders and Baerveldt, 2003). The likelihood-based method does not suffer from this problem because the maximum likelihood estimate always is contained in the confidence interval based on the profile likelihood.

There may be reasons to distrust the estimates which are large with also a large standard error. (This is known as the Donner-Hauck phenomenon in logistic regression, discussed in Section 7.12.2.) Unfortunately, it is impossible to say in general what is to be regarded as a large standard error. A threshold of 4 or 5 for the standard error often is reasonable for most effects; if a tested parameter has a standard error larger than 4, then it is advisable to redo the analysis in a specification where this parameter only is fixed to 0 and a score test is carried out for this parameter. However, for some effects, in any case for the 'average similarity' effect for behavior dynamics, parameters and standard errors tend to be larger, and a larger threshold (e.g. 10) is appropriate. The same holds for effects of covariates with small variances (less than .1).

An alternative, probably better, for the estimation of standard errors is by using a non-parametric bootstrap confidence interval. For example, the adjusted percentile (BC_a) method (Efron, 1987; Davison and Hinkley, 1997, Chapter 5) which is available in function boot.ci in R package boot.

12.3.2 Meta-analysis directed at testing the parameters

Another method for combining the various data sets, which does not make the assumption that the parameters are a sample from a population and also makes no assumptions of absence of correlation[17] between the true deviations $\theta_j - \theta$ and the random errors E_j, is based on Fisher's method for combining independent p-values; the principle of this combination method of Fisher (1932) is described in Hedges and Olkin (1985) and (briefly) in Snijders and Bosker (2012, Chapter 3).

This principle here is applied in a double test:

1. for detecting if there are any networks with a positive parameter value, the null hypothesis tested is
   H_0 : For all networks, the value of this parameter is zero or less than zero;
   with the alternative hypothesis;
   H_1 : For at least one network, the value of this parameter is greater than zero;

2. for detecting if there are any networks with a negative parameter value, the null hypothesis tested is
   H_0 : For all networks, the value of this parameter is zero or greater than zero;
   with the alternative hypothesis
   H_1 : For at least one network, the value of this parameter is less than zero.

For each of these combined tests, the p-value is given. In the output these are denoted, respectively, as 'combination of right one-sided p-values' and 'combination of left one-sided p-values'.

It is advisable to use for each the significance level of $\alpha/2$ (e.g., 0.025 if $\alpha = 0.05$) which yields an overall combined test at significance level α. Note that four different overall results are possible. Indicating the right-sided and the left-sided p-values by p_r and p_l, respectively, these possible results are:

(a) $p_r > \alpha/2,\ p_l > \alpha/2$:
No evidence for any nonzero parameter values;

(b) $p_r \leq \alpha/2,\ p_l > \alpha/2$:
Evidence that some networks have a positive parameter value, no evidence for any negative parameter values;

(c) $p_r > \alpha/2,\ p_l \leq \alpha/2$:
Evidence that some networks have a negative parameter value, no evidence for any positive parameter values;

[17] This correlation is defined for the population of networks, and if the population does not exist then also the correlation is not defined.

(*d*) $p_r \leq \alpha/2,\ p_l \leq \alpha/2$:
Evidence that some networks have a negative parameter value, and some others have a positive parameter value.

If all networks have a zero true parameter value, i.e., under the combined null hypothesis that $\theta_j = 0$ for all j, the probability of result (1) is less than or equal to α; this is the way in which this combined test respects the overall probability of an error of the first kind.

12.3.3 Contrast between the two kinds of meta-analysis

To understand the contrast between the method following the Cochran approach for inference about a population of networks, and the Fisher approach for combining independent tests, the following may be helpful. Inferring about a population always adds some uncertainty; this is more serious when the sample size (here: number of combined networks) is smaller. In the extreme case, consider the combination of $N = 2$ networks, with estimates $\hat{\theta}_1 = 1$, standard error $s_1 = 0.1$, and $\hat{\theta}_2 = 5, s_2 = 0.1$. Then for both of the groups the t-statistic $\hat{\theta}_j/s_j$ is very large, leading to the conclusion that parameters θ_1 and θ_2 are very likely to be positive. This will lead to a significant result for Fisher's combination of tests. On the other hand, the mean in the population of networks, given that there is available a sample of size as low as $N = 2$, cannot be determined with any degree of precision, so the confidence interval for this mean μ_θ will be huge, and the result for testing the null hypothesis $H_0^{(1)}$ will not be significant. However, the results for testing $H_0^{(0)}$ and $H_0^{(2)}$ will be significant.

12.4 Random coefficient multilevel Siena analysis

The function multi_siena is for Bayesian estimation of one group or of multiple groups all having the same number of waves and the same model specification. It is available in package multiSiena (see the SIENA website). The methodology is explained in Koskinen and Snijders (2023). The parameters – excepting the basic rate parameters – can be either randomly varying between groups according to a multivariate normal distribution, or non-varying and constant across groups. (Note that the term 'non-varying' here does not mean that parameters are fixed, only that they do not vary between groups.) The difference is made by setting the argument `random` in the function set_effect. The default is that only the out-degree (density) effect is randomly varying, but it is advisable to specify this for a larger set of effects. Specifying it for too many effects may, however, lead to unstable estimation.

The analysis is done by a Bayesian estimation method. This contrasts with the other functions in RSiena, which are based on the frequentist approach to statistical inference. There are many introductions to Bayesian inference and textbooks; see e.g., Kaplan and Depaoli (2013); Gill (2015); Donovan and Mickey (2019).

The operation of multi_siena is explained in this section, and also in https://www.stats.ox.ac.uk/~snijders/siena/sienaBayes_s.pdf.

For the groupwise parameters normal distributions are assumed with conjugate priors. The prior distribution for the basic rate parameters is determined in a data-dependent way. For the non-varying parameters, the default is to assume a flat prior.

The procedure consists of three parts: initialization, warming, main phase.

1. In the initialization phase, initial parameter values and the proposal covariance matrix for Metropolis-Hastings ('MH') steps for groupwise parameters are obtained. This initialization has four parts.
   First, Method of Moments estimation of a parameter vector assumed to be the same across the groups (in a multi-group estimation), with step size `initgainGlobal`. This can be replaced by a run of siena outside of multi_siena, and giving the results of this run to multi_siena as the `prevAns` parameter.
   The second part, if `initgainGroupwise` is greater than 0, is one subphase of the Robbins-Monro algorithm for Method of Moments estimates for each group separately, with step size `initgainGroupwise`. Special precautions are taken here for dealing with group-level monadic variables, to prevent the issues that arise with such variables in normal use of siena.
   Third, `nprewarm` pre-warming steps are taken. These are MH steps where the proposal covariance matrices are used directly (i.e., without scaling) for random walk proposals. Pre-warming is stopped when too few acceptances are observed.
   The fourth part of the initialization scales the proposal covariance matrices, in the function improveMH, to achieve about 25 out of 100 acceptances of Bayes proposals after single MH steps.

2. After initialization and scaling of the proposal covariance matrices, a warming phase is done of nwarm Bayesian proposals each with a number of MH steps, followed again by the function improveMH.

3. Finally `nmain` repeats of $\Big\{$`nrunMHBatches` repeats of a number of MH steps sampling chains (the 'inner loop'), plus `nSampVarying` MH steps sampling the varying parameters (θ_j) plus `nSampConst` MH steps sampling the non-varying parameters (η) plus one Gibbs step sampling the global mean and covariance matrix of the varying parameters (μ and Σ)$\Big\}$ are performed. In the warming as well as the final phase, the number of MH steps in the inner loop is determined by parameter `mult` ('multiplication factor') in the call of set_algorithm_saom that created the algorithm object.

The function multi_siena is time-consuming. When starting to use it, it is advisable to start with low values of `nmain` to explore computing time. When the procedure seems to diverge, and for very small groups, it is advisable to use smaller values of the parameters `initgainGlobal` and `initgainGroupwise`; and perhaps `reductionFactor`.

12.4.1 Treatment of missing data

For missing data in the dependent variables (not in the covariates!), a different procedure is used than the one in the Methods of Moments estimation procedure. The missing data at the beginning of the wave are modeled as random variables with a simple prior distribution, given by the distribution of the non-missing entries for this wave. The missing data at the end of the wave are modeled as random variables with the distribution implied by the Markov chain model with the estimated parameters.

12.4.2 Which data sets to use for multi_siena

multi_siena uses as data set a sienaGroup object with 2 or more groups. The number of waves should be the same for all groups.

The following data configurations are not allowed for groups included in multi_siena estimation:

1. tie variables in two consecutive waves changing from structural zero (code 10) to 1;
2. tie variables in two consecutive waves changing from structural one (code 11) to 0;
3. tie variables in three consecutive waves changing from structural zero (code 10) to NA to 1;
4. tie variables in three consecutive waves changing from structural one (code 11) to NA to 0;
5. and, for more than three consecutive waves, similar patterns with more NAs in between.

To use data sets including such patterns for multi_siena, you will have to make minimal changes to the data set so as to avoid these patterns. For example, replace sequences 10-1 by NA-1, and 10-NA-1 by NA-NA-1. An example script to make such replacements is at http://www.stats.ox.ac.uk/~snijders/siena/changeForbiddenChanges.R.

In addition to what is or is not allowed as data sets for multi_siena, it is important to mention that using structural zeros to indicate intermittent absence of actors (i.e., structural zeros only at the beginning or only at the end of a period) is not a good idea for multi_siena. Tie variables that are 0 or 1 at the beginning of a wave and structurally zero (code 10) at the end of the wave, have to assume the value 0 at the end of the period; and this is used as information for the estimation in multi_siena of the parameters. Probably by such a structural zero you wish to convey the meaning that the value of this tie variable is unknown. If there are two waves of data, the simple solution is to replace all structural zeros for wave 2 by missing values (i.e., NA). There is no straightforward solution if there are three or more waves.

multi_siena should be possible for groups as small as 5 actors. A restriction (maybe to be lifted later) is that the networks must not be empty at any wave; and consecutive waves of networks must not be identical (in other words, all Jaccard indices should be strictly less than 1).

Be prepared for long computation times. The reason is that likelihood-based computations are used (as distinct from the Method of Moments approach). If all individual groups have enough information for good estimation by the Method of Moments according to the intended model, the use of siena with the (default) Method of Moments, followed by a meta-analysis by means of meta_siena or metafor, may be preferable.

It is advisable to first do a multi-group analysis of the same model, followed by a test_time, to get an initial understanding of where problems might occur. You may then later use the result for the prevAns parameter of multi_siena.

12.4.3 Model specification

Non-constant rate functions currently are not supported in multi_siena.

The extra part of model specification for multi_siena, compared to siena, is that it is required to specify which parameters are randomly varying from group to group, and which are constant

across groups. The specification of constant vs. randomly varying for the other parameters is done in the function set_effect, by its parameter `random`. To be able to see which of the effects are specified in this way, please have a look at the help page for print.sienaEffects:

```
?print.sienaEffects
```

and note the parameters `includeRandoms` and `dropRates`, which are helpful especially for random coefficient multilevel modeling. Requesting `includeRandoms = TRUE` will show the column indicating which effects are random; `dropRates = TRUE` omits the rows with rate effects, which can be a lot of superfluous information that perhaps you do not want to see.

The exception to the rate parameters being always randomly varying is that they may be fixed — in which case all of them need to be fixed, and input parameter `priorRatesFromData` needs to be set to –1.

There currently is little advice about which effects to specify as randomly varying. The basic issues are the following.

interest: is variability of the effect across groups a primary part of the research question?
(Usually not; such research questions about variability are of a quite secondary nature.)

knowledge: is there prior knowledge about whether effects differ between groups?
(Usually not. It is possible to first do a multi-group analysis of the same model, followed by a test_time, and specify those effects as randomly varying for which the time heterogeneity is largest according to the groupwise results.)

misspecification: if it would be erroneously assumed that the effect is constant across groups, would this affect the parameter estimates of primary interest?
(About this we have little general knowledge; it can be tried out by running estimations for different specifications of this part of the model.)

convergence: specifying effects as random will influence convergence. Up to a point, randomly varying parameters will have better convergence properties than constant parameters; but if their number is too large, given the number of groups, convergence can deteriorate again. This may then lead to outliers in the posterior means, which can be diagnosed by function plotPostMeansMDS.

amount of information: which specification will use the information in the data most efficiently?
(Here finally we do have a provisional answer. Assuming that an effect is constant across groups will give a smaller uncertainty —posterior standard deviation, interpreted as standard error— in the estimated parameter than assuming it varies randomly; cf. Section 12.4.9. This will be the more so as the number of groups is smaller. Therefore, for the coefficients for which there is no strong prior knowledge that they are variable across groups, and which are tested as a primary issue for answering the research question, from the point of view of statistical power it is advisable to specify that they are constant.)

12.4.4 How to enter your data in multi_siena

See the example at the bottom of the multi_siena help page for how a sienaGroup object can be created and used. If some but not all of the Siena data objects combined in the sienaGroup have periods where changes are only upward or only downward, it will be necessary to use `allowOnly = FALSE` in the call of as_dependent_rsiena; see the help page for sienaDependent. The scripts `RscriptListGroups.R` and `RscriptMultipleGroups.R` give further examples and explanation for creating sienaGroup objects.

If you have a large number of groups (more than 30), try first with a smaller number of groups (10-20). If you need or wish to make a selection of groups, select the one with few or no missing data and with Jaccard coefficients at least 0.30.

12.4.5 How to choose the parameter settings for multi_siena

It is good to have an initial estimation as a multi-group estimation using the Method of Moments, i.e., using siena, with an algorithm having `nsub = 2` and a reduced value of `firstg`, e.g., 0.05 or less, and look at these results – they will give you a rough idea even if they are nowhere near convergence. If some of the estimated non-rate parameters are very large in absolute value, use a smaller value of `firstg`, such as 0.01 or 0.001. This result of siena can then be given as the `prevAns` parameter to multi_siena (see the help page). Note that the parameter `firstg` corresponds to `initgainGlobal` of multi_siena, if no `prevAns` is used.

It may be good to have an initial try run with
`nwarm = 5, nmain = 10, nrunMHBatches = 5, nImproveMH = 20` (for speed)
and
`silentstart = FALSE` (for information about the initialization phase),
and then printthe result. This will give information about the results of the initialization phase and about computing time. If some of the groups have some very high estimated rate parameters, you should either drop those groups or decrease the value of `initgainGroupwise`. The new value could be, e.g., 0.005 or 0.001 or 0.0. With the lower value of `initgainGroupwise`, dropping the groups concerned may be unnecessary, so don't drop groups too soon.

For normal use, `nwarm = 500, nmain = 1000, nrunMHBatches = 20, nImproveMH = 100` may be reasonable. Computing time is roughly proportional to `nmain` $\times$ `nrunMHBatches`. We still have to develop guidelines about how to choose the number of iterations. If the tracelines show that the process is still quite unstable even in the later part of the runs, possibilities are to increase `nrunMHBatches` but also to increase the `mult` parameter in set_algorithm_saom. Increasing these will for both of them lead to a proportional increase in computing time.

If multiple processors are available (which most computers have nowadays), you can make more speed by setting the `nbrNodes` parameter to a value larger than 1. Since parallelization goes by period $\times$ group, it is nice, but not necessary, to have a value for `nbrNodes` that is a divisor of (the number of periods multiplied by the number of groups); higher values are meaningless. Do not use such a high value for `nbrNodes` that your computer gets too hot or overworked. For Windows machines this can be monitored by opening the Task Manager (you will find how to do this by right clicking on the bottom toolbar).

12.4.6 Prior distributions

The prior mean and prior variance need to be given for the vector of parameters that are randomly varying between the groups. These are the rate parameters and the parameters specified with

```
random = TRUE
```

in the call of `set_effect`. For an effects object called `myeff`, you can see which parameters are specified as randomly varying by

```
print(myeff, includeRandoms = TRUE)
```

This will also tell you the number of random effects, which are the dimensions for the parameters `priorMu` and `priorSigma` used to specify the prior distribution. The rate parameters can be dropped from the `print` result by requesting

```
print(myeff, includeRandoms = TRUE, dropRates = TRUE)
```

The list and order of only the randomly varying effects can be shown by requesting

```
myeff[(myeff$randomEffects | (myeff$basicRate & (myeff$group==1) )) & myeff$include,
```

The list and order of only the non-varying effects can be shown by requesting

```
myeff[myeff$include & (!myeff$basicRate) & (!myeff$randomEffects),]
```

Non-varying parameters

For the non-varying parameters (η, the greek letter 'eta'), by default an improper flat prior is used, and nothing needs to be specified by the user.

However, there is the possibility to specify normal priors with mean 0 and a given variance. This can be useful for effects of group-level variables. The prior variances of the non-varying parameters are given by the parameter `priorSigEta`. This needs to be a vector with length equal to the number of non-varying parameters. Values `NA` indicate that a flat prior should be used (which can be interpreted as an infinite variance).

Varying parameters

The population distribution of the randomly varying parameters is assumed to be multivariate normal $\mathcal{N}(\mu, \Sigma)$. The conjugate prior for the multivariate normal distribution is used to reflect the prior uncertainty in the parameters μ and Σ; cf. Gelman et al. (2014, Section 3.6) and O'Hagan and Forster (2004, Chapter 14). This conjugate prior is the inverse Wishart distribution for Σ; and, conditional on Σ, for μ a multivariate normal distribution, specified as follows:

$$\Sigma \sim \text{InvWishart}_{p_1}(\Lambda_0^{-1}, \nu_0)\,, \text{ and conditionally on } \Sigma \tag{14a}$$

$$\mu \mid \Sigma \sim \mathcal{N}_{p_1}(\mu_0, \Sigma/\kappa_0)\,. \tag{14b}$$

Thus, the parameters of the prior are ν_0, Λ_0, μ_0, and κ_0. These are specified for multi_siena in RSiena in the following way:

- ν_0 = `priorDf`;

- $\Lambda_0 =$ `priorDf` $\times$ `priorSigma`;
- $\mu_0 =$ `priorMu`;
- $\kappa_0 =$ `priorKappa`.

This means the following for the prior parameters as specified for multi_siena:

- `priorDf` can be regarded as the effective sample size that has led to the prior information. For mathematical reasons, this must be at least $p + 2$, where p is the number of varying parameters.
- `priorSigma` is your prior guess for the population covariance matrix of the varying parameters; in other words, this reflects the differences that may exist between the groups.
- `priorMu` is your prior guess for the population mean μ of the varying parameters.
- `priorSigma / priorKappa` reflects your uncertainty about your guess for μ.

It is quite a restriction that the covariance matrix expressing the prior uncertainty about μ should be proportional to the population (i.e., 'true between-group') covariance matrix of the varying parameters. This is a mathematical consequence of the use of the conjugate prior. Often, the prior uncertainty about μ will be greater than the assumed between-group differences, leading to a choice of `priorKappa` less than 1, e.g., 0.01; note that `priorKappa` is a ratio of variances, and the ratio of uncertainties will be better expressed by its square root.

More research is needed for advice about prior distributions. Especially for small numbers of groups, the priors may have a strong influence.

The prior mean of the varying parameters, `priorMu`, is a vector of length equal to the number of varying parameters. In the object produced by multi_siena, let us call it `ans`, this length is stored as `ans$p1`. For example, in a model with 4 waves, one dependent network variable, and varying parameters specified for outdegree, reciprocity, transitive ties, and similarity for some covariate V, the length would be 3+4=7 (3 for three periods + 4 for four varying parameters). The prior covariance matrix of the varying parameters, `priorSigma`, is a symmetric square matrix of dimension equal to the length of `priorMu`.

The treatment of these prior distributions should distinguish between the rate parameters and the parameters for the objective function.

Rate parameters

If rate parameters are fixed (an exceptional case), they are not included in the prior.

In other cases, special attention must be given to the rate parameters. Sometimes, for small groups and complicated models, some of the rate parameters may be estimated in the multi-group option by very high numbers. This may be the case especially for groups with low Jaccard coefficients, or for groups that deviate strongly from the other groups. It may be advisable to take out the groups with extremely high rate parameters (e.g., larger than 50 or 80). To make it feasible to try and include some groups with high estimated rate parameters, a stronger prior distribution for these parameters may be employed.

By default, the prior for the basic rate parameters is data-dependent, with a mean and covariance matrix that are robust estimates based on the estimated rate parameters from the initialization phase. This mostly will be adequate.

To have explicit control of the prior for the basic rate parameters, the data dependence can be turned off by using `priorRatesFromData = 0`. Then the choice for `priorMu` and `priorSigma` is going to matter. Consider the rate parameters as estimated in the multi-group analysis of the same data set. Suppose the total number of groups is N. For each separate rate parameter (for a given wave and for a given dependent variable) there then are N estimates, some of which may be too high; denote the average and the variance of the subset of not-too-high values by m and s^2, respectively. These are specific for the given wave and the given dependent variable. These values for mean and variance should represent what one might find plausible values for this rate parameter.

The corresponding elements of `priorMu` and diagonal elements of `priorSigma` should be then set, respectively, to values close to these m and s^2.

As an example, suppose there are $N = 20$ groups, 3 waves and one dependent variable (a network), and 5 varying parameters in addition to the rate parameters. Then `p1` = 2+5 = 7. Suppose that for the rate parameters the default data-dependent prior is used (`priorRatesFromData = 2`). Then in the parameters `priorMu` and `priorSigma` it still is necessary to include the prior rate parameters, to have the correct dimensionality of this vector and matrix, but their values do not matter. Suppose further that for the parameters of the objective function the prior mean for the density parameter is specified as -1, for the reciprocity parameter as $+1$, all other prior means as 0, and all prior between-group variances as 1. Further assume that the prior uncertainty about μ is thought to be 10 times larger, on the variance scale, than the prior between-group differences. Assuming that `mydaya` and `myeff` are, respectively, a multigroup data set and an effects object, one then could use the following piece of code.

```
m7 <- c(5, 5, -1, 1, 0, 0, 0)
S7 <- matrix(0, 7, 7)
diag(S7) <- 0.01
ans <- multi\_siena(mydata, myeff, priorMu = m7, priorSigma = S7, priorDf = 22, prior
                   priorRatesFromData = 2, ....)
```

`priorDf` is the prior degrees of freedom for the covariance matrix. It can be interpreted as the sample size on which, hypothetically, the prior knowledge about the covariance matrix of the parameters is based. The influence of the prior on the results is larger when `priorDf` is larger.

Parameters of the objective function

For parameters of the objective function, it will be usually be possible to use some prior knowledge, together with neutrality with respect to the sign of tested parameters (in order not to unduly bias results.) In most cases the outdegree parameter is expected to be negative and the reciprocity parameter positive. The researcher should consider earlier studies of similar network dynamics; reasonable values for the prior mean for the outdegree parameter might be –2 or –1, and for the reciprocity parameter +1.5 or +2. For homophily parameters on important attributes expressed by the `simX` effect (which is standardized), as long as these are regarded as control effects, one might specify the prior mean conservatively as 0.3 or 0.5.

Continuing the example above, if the 5 varying non-rate parameters would start with an out-degree and a reciprocity parameter, the prior means could be modified, e.g., to

```
m7 <- c(5, 5, -2, 1.5, 0, 0, 0)
```

(if for the similarity parameters also the value of 0 is used...).

12.4.7 Operation of multi_siena

In Section 12.4.5 it was already suggested to start with a run with very low values of the sample size settings for the MCMC procedure.

It is advisable to use multiple processes ('parallel computing') for multi_siena; see Section 7.13 where it is explained that parallelization is different for multi_siena than for MoM estimation by siena.

When the procedure has made a good start but the MCMC sample seems too short, you can make a prolonged analysis using the `prevBayes` option, and then combine the earlier with the later results using multi_glue. This is illustrated in the example on the help page.

During operation of multi_siena, partial results of the function are now and then stored as objects named `z` in files with the name `PartialBayesResult.RData`; see the help page. This is for the case that the computer or R stops inadvertently during the long computations. These are multi_sienaFit objects, and therefore can be used in the printand summary functions; they also can be used for the `prevBayes` option to continue estimation.

12.4.8 Assessing convergence

You can visually inspect convergence by looking at the tracelines of the various parameters. These can be plotted by the functions in `BayesPlots.r` (available from the Siena website). In many cases, the tracelines for the rate parameters already tell the story about convergence.

The file `BayesPlots.r` contains a variety of plotting functions that can be used to obtain trace plots and posterior density plots.

The parameters `nwarm` and `nmain` in the call of multi_siena only imply that an extra improveMH step is made between the warming and the main iterations; there are no other differences between the warming and main iterations. It is possible that convergence has set in only later; depending on the case, the traceplots may give information about this. If you conclude that convergence has occurred later, then use this to define the `nfirst` parameter in the printand summary functions for multi_sienaFit object (see `?print.sienaBayesFit`).

When the procedure seems to have diverged and this occurs right from the start, it is advisable to use smaller values of the parameters `initgainGlobal` and `initgainGroupwise`. If divergence sets in later and is most pronounced for the rate parameters, it may be advisable to use a smaller value of `reductionFactor`, e.g., 0.1. If generally the tracelines are irregular, it may be good to increase `nrunMHBatches` but another possibility is to increase the `mult` parameter set in set_algorithm_saom.

Using other packages for convergence assessment

The advice of literature such as Gelman et al. (2014) is to use multiple sequences produced independently, preferably from overdispersed starting points, for assessing convergence. For example,

one may use 3 to 5 such sequences. Function extract.multi_siena can be used to extract from these sequences the draws from the posterior distributions of the parameters of interest. This function produces a three-dimensional array of iterations by chains by parameters, which then can be used, e.g., in function monitor of package rstan or with the help of package coda.

Currently there is no good way in multi_siena to use overdispersed starting points. (The difficulty is to get overdispersion while still retaining a reasonable convergence for each sequence.) The best option currently is to use independent restarts of the whole algorithm; or to use one MoM estimation run as the basic starting point, and use that as `prevAns` parameter in several independent restarts, and perhaps continuations after this using `prevBayes`. Note that 'independence' is obtained by using different random number seeds in the `sienaAlgorithm`; and that this is implied by its default value `NULL`.

Function monitor gives information about the potential scale reduction $\hat{R}$ of the posterior distribution if simulations were continued indefinitely, and the effective sample size n_{eff} (i.e., the estimated equivalent sample size under independent sampling). Rules of thumb given in Gelman et al. (2014, p. 287) are that, for all parameters of interest, $\hat{R} \leq 1.1$ and $n_{\text{eff}} \geq 5m$, where m is the number of chains.

Note that what is given by monitor as the '`se_mean`' is the standard error of the mean of the posterior distribution as an estimator for the global mean μ_i: i.e., this expresses the uncertainty due to the finite length of the MCMC chain, it is not a measure of spread of the posterior distribution itself.

12.4.9 Interpreting results of multi_siena

The printand summary functions give posterior means, posterior standard deviations, 95% credibility intervals, and one-sided posterior p-values for testing whether the parameter is positive or negative. These are the Bayesian versions of estimates, standard errors, confidence intervals, and p-values. These functions can be used with a parameter `nfirst`, indicating the first iterations from which convergence is assumed. The default value is the number of warming iterations + 1, but this may be inadequate. From the trace plots obtained by functions in script `BayesPlots.r` is may be possible to make an improved guess for this iteration number, and use this for `nfirst`.

The function siena_table can be used also for making `html` or LaTeX tables of the results for the global parameters; here again, the parameter `nfirst` may be used.

The functions test_parameter and test_parameter are available for testing parameters; see the help page for these functions. To compute further properties of the sample of the posterior, the components `ThinParameters, ThinPosteriorMu, ThinPosteriorEta`, and `ThinPosteriorSigma` of the multi_sienaFit object, as mentioned in the help page, may be useful.

The file `BayesPlots.r`, mentioned above, contains some plotting functions that are useful for making posterior density plots.

The function extract_posteriorMeans can be used to obtain posterior means and standard deviations of the groupwise estimated parameters. These are like the '*empirical Bayes*' estimates known also from other statistical models. A multi-dimensional scaling ('MDS') plot of the posterior means can be obtained from the function
plotPostMeansMDS. This is useful especially for detecting outliers. If strong outliers are found,

this may have two different reasons: truly deviating groups; or instability of the estimation of the posterior means, related perhaps to having too many randomly varying parameters. In the latter case, it may be quite arbitrary (in the sense of 'depending on the whim of the random numbers during the operation of multi_siena') which groups are indicated in the plot as being the outliers.

It should be noted that due to the high correlations between parameters, the posterior means may be useful mainly for model specifications with a limited number of randomly varying parameters. If one is interested, for example, in assessing differences between groups with respect to reciprocation by comparing posterior means for the reciprocity parameter, it should be avoided to specify other parameters as randomly varying that are highly correlated with the reciprocity parameter.

It should be noted that the `.txt` files produced by multi_siena are produced somewhere in the initial phase of the project and not meant to be informative for final results. They contain the results of the initial estimate by the multiple groups method and may be disregarded.

When comparing results for specifications that differ with respect to specifying the effects as constant or varying across groups, it will be noted that posterior standard deviations for the means are larger when specifying the effects as randomly varying, as compared to specifying them as constant. This is natural, and it is associated with a difference in interpretation. Specifying the effect as randomly varying implies that there also is an important step of generalization from the observed groups to the population of groups. The between-group variance then is a priori unknown and one is estimating a mean parameter from a sample of N groups; usually N is not very large and the uncertainty about the between-group differences will contribute considerably to the uncertainty of the population mean.

13 Mathematical definition of effects

The list of all effects available for any data set is obtained by the command

```
effectsDocumentation()
```

which produces a html file. For a given effects object, say with the name `myeff`, the command

```
effectsDocumentation(effects = myeff)
```

will give a file with all effects implemented for this effects object. See

```
?effectsDocumentation
```

for further options.

This chapter present the mathematical formulae for the definition of the effects. Further background to these formulae can be found for network dynamics in Snijders (2001, 2005); Snijders et al. (2010b); for network and behavior dynamics in the last reference and Steglich et al. (2010); and for co-evolution of multiple networks, including two-mode networks, in Snijders et al. (2013). The effects are grouped into effects for modelling network evolution and effects for modelling behavioral evolution (i.e., the dynamics of dependent actor variables). Within each group of effects, the effects are listed in the order in which they appear in SIENA. The short name of the effect (`shortName`), as it is specified in RSiena is specified in brackets.

For two-mode (bipartite) networks, only a subset of the effects is meaningful, since the first node set has only outgoing ties and the second only incoming; for example, the reciprocity effect is meaningless because there cannot be any reciprocal ties; the out-degree popularity effect is meaningless because it refers to incoming ties of actors with high out-degrees; and there are no direct similarity effects of actor covariates. However, distance-two effects such as `simEgoInDist2`, and the interaction of effects like `altInDist2` with `egoX` can be used to represent homophily-like tendencies. There are some specific effects for two-mode networks, e.g., the four-cycle effect.

13.0.1 Internal effect parameters

Some of the effects contain a number which is denoted in this section by p, and called in this manual an *internal effect parameter*. (These are totally different from the statistical parameters which are the weights of the effects in the objective function.)

The internal effect parameter is used to define several specifications of effects; they are fixed parameters which cannot be estimated. These are set or modified by the `set_effect` function, e.g.,

```
myeffects <- set_effect(myeffects, gwespFF, parameter=69)
```

The value of the internal effect parameter is mirrored in the effect name shown in RSiena results; e.g., the effect mentioned here will be represented as
`GWESP I->K->J (69) ;`
as another example, when the internal effect parameter refers to taking a square root, this will be reflected by `sqrt` in the effect name.

13.1 Network evolution

The model of network evolution consists of the model of actors' decisions to establish new ties or dissolve existing ties (according to *evaluation*, *creation*, and *endowment functions*) and the model of the timing of these decisions (according to the *rate function*). The model, and the roles played by these three functions, were briefly explained in Section 6.1.

For some effects the creation and endowment functions are implemented not for estimation by the Method of Moments but only by the Maximum Likelihood or Bayesian method; this is indicated below by 'endowment effect only likelihood-based'.

(It may be noted that the network evaluation function was called objective function, and the creation and endowment functions were called gratification function, in Snijders (2001).)

13.1.1 Network evaluation function

The network evaluation function for actor i is defined as

$$f_i^{\mathrm{net}}(x) \;=\; \sum_k \beta_k^{\mathrm{net}} s_{ik}^{\mathrm{net}}(x) \tag{15}$$

where β_k^{net} are parameters and $s_{ik}^{\mathrm{net}}(x)$ are effects as defined below. If the model also contains some elementary effects (see Section 6.1.1), the objective function is the sum of this and

$$f_i^{\mathrm{el}}(x) \;=\; \sum_k \beta_k^{\mathrm{el}} s_{ik}^{\mathrm{el}}(x)\;, \tag{16}$$

see Section 6.1.3. Elementary effects are of the type $s_{ijk}^{\mathrm{el}}(x) \;=\; x_{ij}\, s_{ijk}^{\mathrm{el0}}(x)$, where $s_{ijk}^{\mathrm{el0}}(x)$ does not depend on x_{ij}.

The potential effects in the network evaluation function are the following. Note that in all effects where a constants p occurs, this constant can be chosen and changed by the user; this is the internal effect parameter mentioned above, which can be modified by the function `set_effect(..., parameter=..., ...)`. For non-directed networks, the same formulae are used, unless a different formula is given explicitly. Some of the effects are dropped for non-directed networks, because they are not meaningful; and some of the names differ in the non-directed case. Section 13.1.2 gives some effects that are specific to non-directed networks.

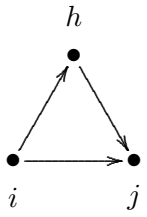
13.1.1.1 Structural effects

Structural effects are the effects depending on the network only. The following list also contains some elementary effects (see Section 6.1.1). The type of the elementary effects in RSiena still is `eval`, indicating that its parameter is applied both for creating new ties and for maintaining existing ties.

1. *out-degree effect* or *density effect* (`density`), defined by the out-degree
   $s_{i1}^{\mathrm{net}}(x) = x_{i+} = \sum_j x_{ij}$,
   where $x_{ij} = 1$ indicates presence of a tie from i to j while $x_{ij} = 0$ indicates absence of this tie;

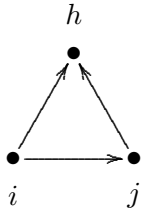
2. *reciprocity effect* (`recip`), defined by the number of reciprocated ties $s_{i2}^{\text{net}}(x) = \sum_j x_{ij}\, x_{ji}$;

3. *transitive triplets effect* (`transTrip`), defined by the number of transitive patterns in i's relations (ordered pairs of actors (j, h) to both of whom i is tied, while also h is tied to j),
$s_{i3}^{\text{net}}(x) = \sum_{j,h} x_{ij}\, x_{ih}\, x_{hj}$;
(there was an error here until version 3.313, which amounted to combining the transitive triplets and transitive mediated triplets effects;)

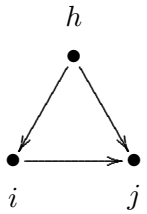


4. *transitive triplets effect type 1* (`transTrip1`), may also be called *transitive closure effect*; the elementary effect corresponding to creating or maintaining the tie $i \to j$ in the figure above; this is transitive closure in the strict sense of the term. The effect is
$s_{i4}^{\text{el}}(x) = x_{ij} \sum_h x_{ih}\, x_{hj}$;

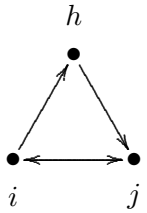
5. *transitive triplets effect type 2* (`transTrip2`), may also be called *two-out-star closure effect*; the elementary effect corresponding to creating or maintaining the tie $i \to j$ in the figure here; this could be called structural equivalence for outgoing ties
(but note that there is also the `balance` effect which is another implementation of structural equivalence equivalence for outgoing ties).
The effect is
$s_{i5}^{\text{el}}(x) = x_{ij} \sum_h x_{ih}\, x_{jh}$;



6. *transitive mediated triplets effect* (`transMedTrip`), defined by the number of transitive patterns in i's relations where i has the mediating position (ordered pairs of actors (j, h) for which h is tied to i and i to j, while also h is tied to j), which is different from the transitive triplets effect only for directed networks,
$s_{i6}^{\text{net}}(x) = \sum_{j,h} x_{ij}\, x_{hi}\, x_{hj}$;
this cannot be used together with the transitive triplets effect in Method of Moments estimation, because of perfect collinearity of the fit statistics;
in the saying 'friends of my friends are my friends', the actor i here is the second 'friend';



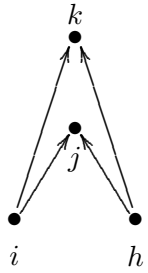
7. *transitive reciprocated triplets effect* (`transRecTrip`), which can be regarded as an interaction between the transitive triplets effect and reciprocity, where the reciprocated tie is the tie $i \leftrightarrow j$ that closes the two-path $i \rightarrow h \rightarrow j$,
$s_{i7}^{\text{net}}(x) = \sum_{j,h} x_{ij}\, x_{ji}\, x_{ih}\, x_{hj}$;



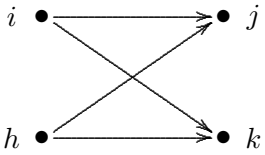
8. *transitive reciprocated triplets effect (type 2)* (`transRecTrip2`), another interaction between the transitive triplets effect and reciprocity, where the reciprocated tie is the tie $h \leftrightarrow j$ in the closed the two-path $\{i \rightarrow h \rightarrow j, i \rightarrow j\}$,
$s_{i8}^{\text{net}}(x) = \sum_{j,h} x_{ij}\, x_{ih}\, x_{hj}\, x_{jh}$;
this represents the tendency to send ties simultaneously to pairs of actors who are mutually linked; but when outdegree-activity is also included in the model, it represents as well the tendency to send ties simultaneously to pairs of actors who are not linked to each other;

9. *number of three-cycles* (`cycle3`),
$s_{i9}^{\text{net}}(x) = \sum_{j,h} x_{ij}\, x_{jh}\, x_{hi}$;

10. *shared popularity* `sharedPop`,
$s_{i10}^{\text{net}}(x) = \frac{1}{4} \sum_{j,k,h;\text{all different}} x_{ij}\, x_{hj}\, x_{ik}\, x_{hk}$;
this is like a 4-cycle but in a special orientation, like the 2PU ('two-paths up') effect (for $k = 2$) for directed ERGMs proposed in Robins et al. (2009). Therefore the statistic is called the 'number of 2-2PU' configurations;



11. for two-mode networks and for non-directed networks:
the *number of four-cycles* with shortName (`cycle4`);
(the shortName (`cycle4ND`) for the non-directed case was replaced by (`cycle4`) in RSiena version 1.4.22)
$s_{i11}^{\text{net}}(x) = \frac{1}{4} \sum_{j,k,h;\text{all different}} x_{ij}\, x_{ik}\, x_{hj}\, x_{hk}$;
for parameter $p = 2$ the square root is taken:
$s_{i11}^{\text{net}}(x) = \frac{1}{2} \sum_{j} x_{ij} \sqrt{\sum_{k,h;\text{all different}} x_{ik}\, x_{hj}\, x_{hk}}$;
note that this is like the `sharedPop` effect above, but for the two-mode and non-directed cases the directionality plays no role;



12. *transitive ties effect* (`transTies`) (earlier called *(direct and indirect ties) effect*), defined by the number of actors to whom i is directly as well as indirectly tied,
$s_{i12}^{\text{net}}(x) = \sum_{j} x_{ij}\ \max_h(x_{ih}\, x_{hj})$;

13. *betweenness count* (`between`),
$s_{i13}^{\text{net}}(x) = \sum_{j,h} x_{hi}\, x_{ij}\, (1 - x_{hj})$;
note that this is one of the rare effects that depends on non-existence of a tie (here, the tie $h \to j$); this makes it less attractive, use it only if you know what you are doing!

14. *balance* (`balance`), defined by the similarity between the outgoing ties of actor i and the outgoing ties of the other actors j to whom i is tied,

$$s_{i14}^{\text{net}}(x) = \sum_{j=1}^{n} x_{ij} \sum_{\substack{h=1 \\ h \neq i,j}}^{n} (b_0 - \mid x_{ih} - x_{jh} \mid) \ ,$$

where b_0 is a constant included to reduce the correlation between this effect and the density effect, defined by

$$b_0 = \frac{1}{(M-1)n(n-1)(n-2)} \sum_{m=1}^{M-1} \sum_{i,j=1}^{n} \sum_{\substack{h=1 \\ h \neq i,j}}^{n} \mid x_{ih}(t_m) - x_{jh}(t_m) \mid \ .$$

This may also be regarded as *structural equivalence with respect to outgoing ties*. (In RSiena versions before 3.324, this was divided by $n-2$, which for larger networks tended to lead to quite large estimates and standard errors. Therefore in version 3.324, the division by $n-2$ – which had not always been there – was dropped.)

15. *structural equivalence effect with respect to incoming ties* (`inStructEq`), which is an analogue to the balance effect but now considering similarity with respect to incoming ties,

$$s_{i15}^{\text{net}}(x) = \sum x_{ij} d_{ij} \tag{17a}$$

with

$$d_{ij} = \sum_{\substack{h=1 \\ h \neq i,j}}^{n} \left(b_0 - \mid x_{hi} - x_{hj} \mid\right) . \tag{17b}$$

This effect is not quite finalized yet, because provisionally $b_0 = 0$ instead of a mean of the subtracted values like in the balance effect. Subtraction of the mean will lead to better convergence properties.

16. *Jaccard similarity for outgoing ties effect* (`Jout`), an elementary effect defined by the Jaccard similarity with respect to outgoing ties,
$s_{i16}^{\text{net}}(x) = \sum_j x_{ij}\, J_{\text{out}}(i,j)$, where

$$J_{\text{out}}(i,j) \ = \ \frac{\sum_h x_{ih}\, x_{jh}}{x_{i+} + x_{j+} - \sum_h x_{ih}\, x_{jh}}$$

(where 0/0 is taken as 0).

Since the Jaccard measure has smaller variability than a lot of other effects, the parameter estimates of this will often be larger, with correspondingly larger standard errors, than many other parameter estimates. The same holds for the other Jaccard similarity effects.

17. *Jaccard similarity for incoming ties effect* `(Jin)`, an elementary effect defined by the Jaccard similarity with respect to incoming ties,
$s_{i17}^{\text{net}}(x) = \sum_j x_{ij}\, J_{\text{in}}(i,j)$, where

$$J_{\text{in}}(i,j) \;=\; \frac{\sum_h x_{hi}\, x_{hj}}{x_{+i} + x_{+j} - \sum_h x_{hi}\, x_{hj}}$$

(where again 0/0 is taken as 0).

18. *number of distances two effect* `(nbrDist2)`, defined by the number of actors to whom i is indirectly tied (through at least one intermediary, i.e., at sociometric distance 2),
$s_{i18}^{\text{net}}(x) = \#\{j \mid x_{ij} = 0,\, \max_h(x_{ih}\, x_{hj}) > 0\}$;
endowment effect only likelihood-based because the Method of Moments estimators for endowment effects are based on the 'loss' associated with terminated ties, and this cannot be straightforwardly applied for the number of distances two effect.
This effect is difficult to interpret: actor i can increase this effect by making a new tie to an actor j at distance 2 but also by dropping a direct tie to an actor j to whom i is also connected via a two-path $i \to h \to j$.
When using this effect to represent a particular theoretical mechanism, it may be advisable to use it as separate creation and endowment effects (using `type=creation` and `type=endow`), to avoid this ambiguity. For representing network closure it may be preferable to use one of the many other effects representing it (`transTrip`, `transTies`, the various gwesp effects, the various Jaccard effects, etc.).

19. *number of doubly achieved distances two effect* `(nbrDist2twice)`, defined by the number of actors to whom i is not directly tied, and tied through twopaths via at least two intermediaries,
$s_{i19}^{\text{net}}(x) = \#\{j \mid x_{ij} = 0,\, \sum_h(x_{ih}\, x_{hj}) \geq 2\}$;
evidently, this is even more difficult to interpret than `(nbrDist2)`; both effects should be used if there are very specific and convincing reasons to include them in the model;
endowment effect only likelihood-based;

20. *number of dense triads* `(denseTriads)`, defined as triads containing at least p ties,
$s_{i20}^{\text{net}}(x) = \sum_{j,h} x_{ij}\, I\{x_{ij} + x_{ji} + x_{ih} + x_{hi} + x_{jh} + x_{hj}) \geq p\}$,
where the 'indicator function' $I\{A\}$ is 1 if the condition A is fulfilled and 0 otherwise, and where p is either 5 or 6;
(this effect is superfluous and undefined for symmetric networks);

21. five variations of the *GWESP (geometrically weighted edgewise shared partners)* effects: `gwespFF`, `gwespBB`, `gwespFB`, `gwespBF`, `gwespRR`, and for non-directed W networks the sixth version `gwesp`.
Note that there is a difference since version 1.1-251; see at the end of this item.

These are effects like those developed for exponential random graph models ('ERGMs') by Snijders et al. (2006), in the parametrisation of Hunter (2007). The five versions relate to

the various directionalities of the ties, just like the proposals for directed networks in Robins et al. (2009).

The `gwespFF` effect is an alternative expression for transitivity. This concept here is specified in an actor-based way, by counting configurations in the local neighbourhood of a given actor, rather than in the tie-oriented way of the models in the ERGM family, for which the GWESP statistic was first developed. The actor-based `gwespFF` effect is defined, in direct analogy to the corresponding global statistic of Hunter (2007), by

$$\text{GWESPFF}(i,\alpha) \;=\; \sum_{k=1}^{n-2} e^{\alpha}\left\{1 \;-\; \left(1-e^{-\alpha}\right)^{k}\right\} \text{EPFF}_{ik}\;, \tag{18a}$$

where EPFF_{ik} (for 'edgewise partners') is the number of nodes j such that $i \to j$ and there are exactly k other nodes h for which there is the two-path $i \to h \to j$.

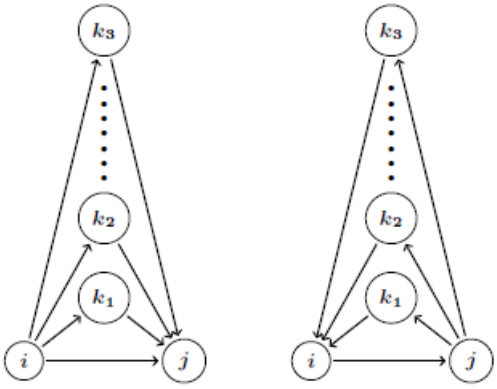


Figure 3: Geometrically weighted edgewise shared partners; left, transitive ('gwespFF'); right, cyclical ('gwespBB'); for F='forward', B='backward'.

An equivalent way of writing this is

$$\text{GWESPFF}(i,\alpha) \;=\; \sum_{j=1}^{n} x_{ij}\, e^{\alpha}\left\{1 \;-\; \left(1-e^{-\alpha}\right)^{\sum_{h=1}^{n} x_{ih}x_{hj}}\right\}\;, \tag{18b}$$

where the convention is used that $x_{jj} = 0$ for all j.

The parameter α is a tuning parameter that may range from 0 to ∞. The internal effect parameter is defined as $100\times\alpha$. For all α, it holds that $\text{GWESP}(0,\alpha) = 0$, $\text{GWESP}(1,\alpha) = 1$, and $\text{GWESP}(k,\alpha)$ increases with k to a maximum slightly less than e^{α}. For $\alpha = 0$ the coefficients $e^{\alpha}\left\{1 \;-\; \left(1-e^{-\alpha}\right)^{k}\right\}$ are equal to 1 for all $k \geq 1$, and for $\alpha \to \infty$ they tend to k. Since we can write

$$\sum_{j,h} x_{ih}x_{hj}x_{ij} \;=\; \sum_{k=1}^{n-2} k\,\text{EP}_{ik}\;,$$

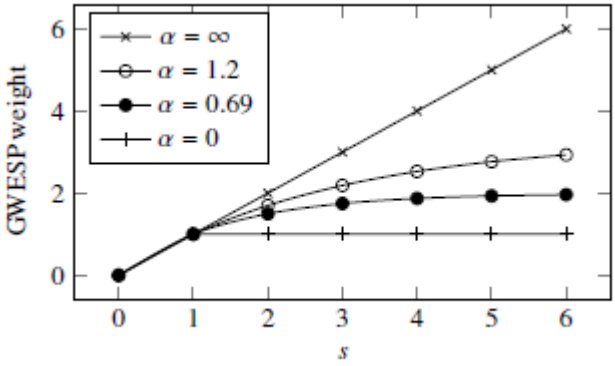


Figure 4: GWESP weight for a tie closing s two-paths for $\alpha = \infty, 1.2, 0.69, 0$.

this implies that for $\alpha \to \infty$ the regular number of transitive triplets is approached, while for smaller α the extra contribution of a high number of intermediaries h is downweighted. An often used value is $\alpha = \log(2) = 0.69$ (Snijders et al., 2006), corresponding to an internal effect parameter of 69,

```
myeffects <- set_effect(myeffects, gwespFF, parameter=69)
```

but it is worthwhile to try out different values of α to see which one gives the best fit.

Although the fit statistic of the GWESP effect is identical to that for transitive ties for $\alpha = 0$ and approximates the fit statistic for transitive triplets for large α, the estimates are not the same because some other calculations are done differently. The issue is that the GWESP effects are not implemented as an evaluation effect, but as an elementary effect, where for the change statistic only changes of the tie $i \to j$ in (18b) are considered, and not of the tie $i \to h$.

Thus, the GWESP effects are defined in RSiena as the elementary effects

$$s_{i21}^{\mathrm{el}}(x) \;=\; x_{ij}\, e^{\alpha} \left\{ 1 \;-\; \left(1 - e^{-\alpha}\right)^{\sum_{h=1}^{n} x_{ih} x_{hj}} \right\} . \tag{19}$$

It should be noted that although the GWESP statistic is not triadic but depends on higher-order configurations, still it is actor-oriented in the sense that only those configurations are considered that are part of the personal network, i.e., the set of actors immediately connected to the focal actor i.

The other types of GWESP effect are analogous, with different tie orientations. They are defined as follows:
`gwespBB`: uses EPBB_{ik}, counting the number of nodes j with $i \to j$ and there are exactly k other nodes h for which there is the two-path $i \leftarrow h \leftarrow j$;
`gwespFB`: uses EPFB_{ik}, counting the number of nodes j with $i \to j$ and there are exactly

k other nodes h for which there is the two-out-star $i \to h \leftarrow j$;
`gwespBF`: uses EPBF$_{ik}$, counting the number of nodes j with $i \to j$ and there are exactly k other nodes h for which there is the two-in-star $i \leftarrow h \to j$;
(the last two were incorrect in the manual before 23/06/2017)
`gwespRR`: uses EPRR$_{ik}$, counting the number of nodes j with $i \to j$ and there are exactly k other nodes h for which there are the reciprocal ties $i \rightleftarrows h \rightleftarrows j$.

For further interpretations and distinctions between the different directionalities in these effects, see Robins et al. (2009).

In version 1-1.251 the implementation was changed (thanks to Nynke Niezink), because earlier versions were not quite according to what is described above. The effect was until version 1-1.250 implemented as
$c_1 + c_2 \times \text{GWESPFF}(i, \alpha')$
for values $c_1(\alpha)$ and $c_2(\alpha)$ not dependent on x, and with positive parameters α, α' related according to $\exp(-\alpha) + \exp(-\alpha') = 1$. Note that for the default value $\alpha = \log(2)$ corresponding to the effect parameter 69 (see above), $\alpha = \alpha'$.

22. two variations of the *GWDSP (geometrically weighted dyadwise shared partners)* effects: `gwdspFF` and `gwdspFB`. These are defined, respectively, by

$$\text{GWDSPFF}(i, \alpha) \;=\; \sum_{h=1;\, h\neq i}^{n} e^{\alpha} \left\{ 1 \;-\; \left(1 - e^{-\alpha}\right)^{\sum_{j=1}^{n} x_{ij} x_{jh}} \right\} , \tag{20}$$

and

$$\text{GWDSPFB}(i, \alpha) \;=\; \sum_{h=1;\, h\neq i}^{n} e^{\alpha} \left\{ 1 \;-\; \left(1 - e^{-\alpha}\right)^{\sum_{j=1}^{n} x_{ij} x_{hj}} \right\} . \tag{21}$$

It may be noted these are of a quite different nature than most other effects, with the outgoing tie variables from i tucked far away in the formula. These effects are somewhat similar with respect to what they mean for the networks as `outPop` and `inPop`, respectively.
Like the gwesp effects, these are similar to the effects developed for exponential random graph models ('ERGMs') by Snijders et al. (2006), in the parametrisation of Hunter (2007). They are motivated because they are natural companions to `gwespFF` and `gwespFB`, as the conditions for the transitive closure expressed by the latter two effects.
These effects are implemented as regular effects, not as elementary effects.

23. *number of (unilateral) peripheral relations to dense triads*,
$s_{i23}^{\text{net}}(x) = \sum_{j,h,k} x_{ij}(1-x_{ji})(1-x_{hi})(1-x_{ki})I\{(x_{jh}+x_{hj}+x_{jk}+x_{kj}+x_{hk}+x_{kh}) \geq p\}$,
where p is the same constant as in the *dense triads* effect;
for symmetric networks, the 'unilateral' condition is dropped, and the definition is
$s_{i23}^{\text{net}}(x) = \sum_{j,h,k} x_{ij}(1 - x_{hi})(1 - x_{ki})I\{(x_{jh} + x_{hj} + x_{jk} + x_{kj} + x_{hk} + x_{kh}) \geq p\}$;

24. *in-degree related popularity effect* (`inPop`) (earlier called *popularity* or *popularity of alter effect*), defined by the sum of the in-degrees of the others to whom i is tied,

$s^{\text{net}}_{i24}(x) = \sum_j x_{ij}\, x_{+j} = \sum_j x_{ij} \sum_h x_{hj} = \sum_j x_{ij}(\sum_{h \neq i} x_{hj} + 1)$;
in RSiena 3 until version 3.313, this effect was multiplied by a factor $1/n$;
in RSiena this effect has had a bug until version 1.1-219;
in RSiena the target statistic for this effect was multiplied by a factor n until version 1.1-241;

25. *in-degree related popularity effect (centered)* (`inPop.c`), defined by the sum of the centered in-degrees of the others to whom i is tied,
$s^{\text{net}}_{i25}(x) = \sum_j x_{ij} \big(x_{+j} - d\big)$,
where d is the average in-degree across all waves;

26. *in-degree related popularity (sqrt) effect* (`inPopSqrt`) (earlier called *popularity of alter (sqrt measure) effect*), defined by the sum of the square roots of the in-degrees of the others to whom i is tied,
$s^{\text{net}}_{i26}(x) = \sum_j x_{ij} \sqrt{x_{+j}} = \sum_j x_{ij} \sqrt{\sum_h x_{hj}}$;
this often works better in practice than the raw popularity effect; also it is often reasonable to assume that differences between high in-degrees are relatively less important than the same differences between low in-degrees;

27. *in-degree related popularity dyadic effect* (`inPop_dya`), defined as the elementary effect version of the half-centered squared in-degree of the other to whom i is tied,
$f^{\text{net, el}}_{i27}(x) = \big(x_{+j} - \bar{x}_{+.}\big)$,
where
$\bar{x}_{+.} = \frac{1}{MK} \sum_{m=1}^{M} \sum_{i,j} x_{ij}$,
with K the number of j (for one-mode networks this is n, for two-mode networks this is the number of nodes in the second mode);
for internal effect parameter $p = 2$, it is defined by
$f^{\text{net, el}}_{i27}(x) = \sqrt{x_{+j}} - \sqrt{\bar{x}_{+.}}$;
since this is an elementary effect, it can be used in interactions and in endowment and creation functions;

28. *out-degree related popularity effect* (`outPop`), defined by the sum of the out-degrees of the others to whom i is tied,
$s^{\text{net}}_{i28}(x) = \sum_j x_{ij}\, x_{j+} = \sum_j x_{ij} \sum_h x_{jh}$;
until version 3.313, this effect was multiplied by a factor $1/n$;
this is a regular evaluation effect with standard MoM estimation for internal effect parameter 1 and higher; if the internal effect parameter is equal to 0, the estimation statistic uses the outdegrees x_{j+} of the start of the period; if the internal effect parameter is equal to –1, the estimation statistic uses the sum of the current outdegrees and those at the start of the period;

29. *out-degree related popularity effect (centered)* (`outPop.c`), defined by the sum of the centered out-degrees of the others to whom i is tied,
$s^{\text{net}}_{i29}(x) = \sum_j x_{ij} \big(x_{j+} - d\big)$,
where d is the average out-degree across all waves;

30. *out-degree related popularity (sqrt) effect* (`outPopSqrt`), defined by the sum of the square roots of the out-degrees of the others to whom i is tied,
$s_{i30}^{\text{net}}(x) = \sum_j x_{ij} \sqrt{x_{j+}} = \sum_j x_{ij} \sqrt{\sum_h x_{jh}}$;
this often works better in practice than the raw activity effect for the same reasons as mentioned above for the sqrt measure of the popularity effect;

31. *out-degree related popularity more than p effect* (`outPopMore`), defined by the sum of the difference between the out-degrees of the others to whom i is tied, and the internal effect parameter p, truncated to positive values:
$s_{i31}^{\text{net}}(x) = \sum_j x_{ij} \max\{x_{j+} - p, 0\}$;
internal effect parameter p should be at least 1;

32. *out-degree related popularity (sqrt) more than p effect* (`outPopSqrtMore`), defined by the sum of the difference between the square roots of the out-degrees of the others to whom i is tied, and the internal effect parameter p, truncated to positive values:
$s_{i32}^{\text{net}}(x) = \sum_j x_{ij} \max\{\sqrt{x_{j+}} - \sqrt{p}, 0\}$;
internal effect parameter p should be at least 1;

33. *out-degree related popularity more than p effect* (`outPopThreshold`), defined by the number of others to whom i is tied who have outdegree larger than the internal effect parameter p:
$s_{i33}^{\text{net}}(x) = \sum_j x_{ij} I\{x_{j+} > p\}$;
internal effect parameter p should be at least 1;

34. *reciprocal degree-related popularity effect* (`reciPop`) defined by the sum of the reciprocal degrees of the others to whom i is tied,
$s_{i34}^{\text{net}}(x) = \sum_j x_{ij} x_j^{(r)}$,
where the reciprocal degree is defined by
$x_j^{(r)} = \sum_h x_{jh} x_{hj}$.

35. *reciprocal degree-related popularity (sqrt) effect* (`reciPopSqrt`) defined by the sum of the square roots of the reciprocal degrees of the others to whom i is tied,
$s_{i35}^{\text{net}}(x) = \sum_j x_{ij} \sqrt{x_j^{(r)}}$,
where the reciprocal degree is defined as above;

⊙ for non-directed networks, the popularity and activity effects are taken together as 'degree effects', since in-degrees and out-degrees are the same in this case;

36. *in-degree related activity effect*, (`inAct`) defined as the cross-product of the actor's in- and out-degrees,
$s_{i36}^{\text{net}}(x) = \sum_j x_{ij} x_{+i} = x_{i+} x_{+i}$;
this is a regular evaluation effect with standard MoM estimation for internal effect parameter 1 and higher; if the internal effect parameter is equal to 0, the estimation statistic uses the indegrees x_{+i} of the start of the period; if the internal effect parameter is equal to –1, the estimation statistic uses the sum of the current indegrees and those at the start of the period;

37. *in-degree related activity effect (centered)* (`inAct.c`), defined by the sum of the cross-product of the actor's out- and centered in-degrees,
$s^{\text{net}}_{i37}(x) = x_{i+}\left(x_{+i} - d\right)$,
where d is the average in-degree across all waves;

38. *in-degree related activity (sqrt) effect*, (`inActSqrt`) defined by
$s^{\text{net}}_{i38}(x) = x_{i+}\sqrt{x_{+i}}$;
this is a regular evaluation effect with standard MoM estimation for internal effect parameter 1 and higher; if the internal effect parameter is equal to 0, the estimation statistic uses the square roots of the indegrees x_{+i} of the start of the period; if the internal effect parameter is equal to –1, the estimation statistic uses the square roots of the sum of the current indegrees and those at the start of the period;

39. *in-isolate Outdegree effect*, (`inIsDegree`), the (additional) out-degree (or activity) effect for actors with in-degree zero, defined as the out-degree but only if the actor has in-degree zero,
$s^{\text{net}}_{i39}(x, z) = I\{x_{+i} = 0\}\sum_j x_{ij}$;

40. *out-degree related activity effect* (`outAct`), defined as the squared out-degree of the actor,
$s^{\text{net}}_{i40}(x) = x^2_{i+}$;
endowment effect only likelihood-based;

41. *out-degree related activity effect (centered)* (`outAct.c`), defined by the out-degree times the centered out-degree of the actor,
$s^{\text{net}}_{i41}(x) = x_{i+}\left(x_{i+} - d\right)$,
where d is the average out-degree across all waves;

42. *out-degree related activity (sqrt) effect* (`outActSqrt`) (earlier called *out-degree^(1.5)*), defined by
$s^{\text{net}}_{i42}(x) = x^{1.5}_{i+} = x_{i+}\sqrt{x_{i+}}$
endowment effect only likelihood-based;

⊙ The following effects indicated by the suffix `_ego` are meant mainly for interactions. Recall that for interaction effects, the internal effect parameters should match; see Section 6.12.1. They are elementary effects (see Section 6.1.1) and have `interactionType="ego"`, which means they can be used in interactions and in endowment and creation functions.

43. *in-degree related activity ego effect* (`inAct_ego`), defined as the elementary effect version of the half-centered cross-product of the actor's in- and out-degrees,
$f^{\text{net, el}}_{i43}(x) = \left(x_{+i} - \bar{x}_{+.}\right)$,
where
$\bar{x}_{+.} = \frac{1}{Mn}\sum_{m=1}^{M}\sum_{i,j} x_{ij}$; for internal effect parameter $p = 2$, it is defined by
$f^{\text{net, el}}_{i43}(x) = \sqrt{x_{+i}} - \sqrt{\bar{x}_{+.}}$;

44. *out-degree related activity ego effect* (`outAct_ego`), defined as the elementary effect version of the half-centered squared out-degree of the actor,
$f_{i44}^{\text{net, el}}(x) = \left(x_{i+} - \bar{x}_{.+}\right)$,
where
$\bar{x}_{.+} = \frac{1}{Mn} \sum_{m=1}^{M} \sum_{i,j} x_{ij}$;
for internal effect parameter $p = 2$, it is defined by
$f_{i44}^{\text{net, el}}(x) = \sqrt{x_{i+}} - \sqrt{\bar{x}_{.+}}$;

45. *1 divided by in-degree activity effect* (`divIn_ego`), the elementary effect defined by 1 divided by the actor's in-degree ,
$f_{i45}^{\text{net, el}}(x) = 1/x_{+i}$,
where $1/0$ is computed as 0;
for internal effect parameter $p = 2$ the square root is taken;

46. *1 divided by out-degree activity effect* (`divOut_ego`), the elementary effect defined by 1 divided by the actor's out-degree ,
$f_{i46}^{\text{net, el}}(x) = 1/x_{i+}$,
where $1/0$ is computed as 0;
for internal effect parameter $p = 2$ the square root is taken;

47. *out-degree related activity more than p ego effect* (`outActMore_ego`), defined as the elementary effect of outdegree activity for the positive difference of outdegrees and the internal effect parameter p:
$f_{i47}^{\text{net, el}}(x) = \max\{x_{i+} - p,\, 0\}$;

48. *out-degree related activity more than p (sqrt) ego effect* (`outActSqrtMore_ego`), defined as the elementary effect of outdegree activity for the positive difference of the square roots of the outdegrees and the internal effect parameter p:
$f_{i48}^{\text{net, el}}(x) = \max\{\sqrt{x_{i+}} - \sqrt{p},\, 0\}$;

49. *out-degree related activity with threshold p ego effect* (`outMore_ego`), defined as the elementary effect of outdegree activity for outdegrees larger than the internal effect parameter p:
$f_{i49}^{\text{net, el}}(x) = I\{x_{i+} > p\}$;
this is the elementary-effect version of `outMore`;

50. *reciprocal degree-related activity effect* (`reciAct`) defined by the degree of i multiplied by i's reciprocal degree,
$s_{i50}^{\text{net}}(x) = x_{i+}\, x_i^{(r)}$;
where the reciprocal degree is defined as above;
for internal effect parameter $p = 2$, it is defined by
$s_{i50}^{\text{net}}(x) = x_{i+} \sqrt{x_i^{(r)}}$;

51. *reciprocal degree related activity ego effect* (`reciAct_ego`), defined as the elementary effect version of the degree of i multiplied by i's centered reciprocal degree,
$f_{i51}^{\text{net, el}}(x) = \big(x_i^{(r)} - \bar{x}^{(r)}\big)$,
where
$\bar{x}^{(r)} = \frac{1}{Mn} \sum_{m=1}^{M} \sum_{i,j} x_{ij}\, x_{ji}$;
for internal effect parameter $p = 2$, it is defined by
$f_{i51}^{\text{net, el}}(x) = \sqrt{x_i^{(r)}} - \sqrt{\bar{x}^{(r)}}$;
since this is an elementary effect, it can be used in interactions and in endowment and creation functions;

52. *out-degree up to* p (*truncated out-degree*) (`outTrunc`), where p is some constant (internal effect parameter, see above), there are two implementations here: `outTrunc` and `outTrunc2`, to allow the simultaneous use of this effect with 2 different internal effect parameters; the effect is defined by
$s_{i52}^{\text{net}}(x) = \min(x_{i+}\,,\, p)$;
note that for $p = 1$ this represents –inversely– the tendency to be an isolate with respect to outgoing ties, i.e., have out-degree equal to 0: $\min(x_{i+}\,,\, 1) = 0$ if $x_{i+} = 0$, and $\min(x_{i+}\,,\, 1) = 1$ if $x_{i+} \geq 1$.
Since the representation is inverse, a result like a negative coefficient -1.2 for `outTrunc` (internal effect parameter $p = 1$) is interpreted as a positive tendency $+1.2$ toward outdegrees equal to 0; the standard error is unchanged. In the case of $p = 1$, an alternative and more directly comprehensible name is *outdegree at least 1* effect.
Note that the `outIso` effect (see below) is equivalent to `outTrunc` for $p = 1$, but with the opposite sign.

Compare what is written below about the in-isolate effects: the isolation effects are difficult to interpret in words or pictures, because they are not based on subgraph counts. This holds for the `outTrunc`($p = 1$) effect but, in modified form, also for the `outTrunc` effect generally, The interpretation can best be understood from the change statistics. The `outTrunc` effect has a change statistic of +1 when actor i creates a link, if the previous outdegree of i is less than p. For $p = 1$, this means a change statistic of +1 for the creation of the first outgoing tie of i, and of –1 for the termination of the last outgoing tie.

53. *out-isolate* (`outIso`), defined by
$s_{i53}^{\text{net}}(x) = I\{x_{i+} = 0\}$;
this is equivalent to `outTrunc` for $p = 1$, but with a sign reversal for the parameter;

54. *out-degree more than* p (`outMore`) where p is some constant (internal effect parameter, see above), there are three implementations here: (`outMore`), (`outMore2`), and (`outMore3`), to allow the simultaneous use of this effect with three different internal effect parameters; they are defined by
$s_{i54}^{\text{net}}(x) = \max(x_{i+} - p,\, 0)$;
this can be regarded as the mirror image of `outTrunc`;

55. *out-degree larger than* p (`outThreshold`) (out-threshold(p)) where p is some constant (internal effect parameter, see above), there are two implementations here: (`outThreshold`)

and (outThreshold2), to allow the simultaneous use of this effect with two different internal effect parameters; they are defined by
$s_{i55}^{\text{net}}(x) = I(x_{i+} > p)$;

56. *square root out-degree*, defined by
$s_{i56}^{\text{net}}(x) = \sqrt{x_{i+}}$;
this was left out in later versions of SIENA;

57. *squared (out-degree – p)*, where p is some constant, defined by
$s_{i57}^{\text{net}}(x) = (x_{i+} - p)^2$,
where p is chosen to diminish the collinearity between this and the density effect;
this is left out in later versions of SIENA;

58. *sum of (1/(out-degree + p)* (outInv), where p is some constant, $p \geq 1$, defined by
$s_{i58}^{\text{net}}(x) = 1/(x_{i+} + p)$;
endowment effect only likelihood-based;
also see divOut_ego, which is similar but suitable for interacting with other effects;

59. *sum of (1/(out-degree + p)(out-degree + p + 1))* (outSqInv), where p is some constant, defined by
$s_{i59}^{\text{net}}(x) = 1/(x_{i+} + p)(x_{i+} + p + 1)$;
endowment effect only likelihood-based.

60. *average out-degree* (X: av. degree) (avDeg)
defined by the out-degree of i multiplied by the average outdegree centered by the internal effect parameter p
$s_{i60}^{\text{net}}(x) = \sum_j x_{ij} \left(\frac{1}{n} \sum_h x_{h+} - p \right) = x_{i+} \left(\frac{1}{n} \sum_h x_{h+} - p \right)$;
this effect is a network-by-period level effect and therefore is useful only for multi-group data sets or for data sets with many periods;
it may be beneficial for convergence to use a value of p that is close to the average degree (but constant, while this effect is for the dynamic average outdegree);

⊙ the following assortativity effects are, for $p = 1$, various orientations of three-paths (plus a bit of two-paths):

61. *out-out degree^(1/p) assortativity* (outOutAss), which represents the differential tendency for actors with high out-degrees to be tied to other actors who likewise have high out-degrees,
$s_{i61}^{\text{net}}(x) = \sum_j x_{ij}\, x_{i+}^{1/p} x_{j+}^{1/p}$;
p can be 1 or 2 (the latter value is the default);

62. *out-in degree^(1/p) assortativity* (outInAss), which represents the differential tendency for actors with high out-degrees to be tied to other actors who have high in-degrees,
$s_{i62}^{\text{net}}(x) = \sum_j x_{ij}\, x_{i+}^{1/p} x_{+j}^{1/p}$;
p can be 1 or 2 (the latter value is the default);

63. *in-out degree^(1/p) assortativity* (`inOutAss`), which represents the differential tendency for actors with high in-degrees to be tied to other actors who have high out-degrees,
$s^{\text{net}}_{i63}(x) = \sum_j x_{ij}\, x_{+i}^{1/p} x_{j+}^{1/p}$;
p can be 1 or 2 (the latter value is the default);

64. *in-in degree^(1/p) assortativity* (`inInAss`), which represents the differential tendency for actors with high in-degrees to be tied to other actors who likewise have high in-degrees,
$s^{\text{net}}_{i64}(x) = \sum_j x_{ij}\, x_{+i}^{1/p} x_{+j}^{1/p}$;
p can be 1 or 2 (the latter value is the default);

65. *network-isolate effect*, (`isolateNet`), the effect of ego having in-degree as well as out-degree zero, i.e., being a total isolate,
$s^{\text{net}}_{i65}(x, z) = I\{x_{+i} = x_{i+} = 0\}$;

66. *anti isolates effect*, (`antiIso`), the effect of wishing to connect to others who otherwise would be a total isolate, i.e., have no incoming or outgoing ties, and wishing not to sever connections to others who thereby would become a total isolate,
$s^{\text{net}}_{i66}(x) = \sum_j I\{x_{+j} \geq 1, x_{j+} = 0\}$;

67. *anti in-isolates effect*, (`antiInIso`), the effect of wishing to connect to others who otherwise would have no incoming ties, and wishing not to sever connections to others who thereby would lose their last incoming connection:
$s^{\text{net}}_{i67}(x) = \sum_j I\{x_{+j} \geq 1\}$;
the isolation effects are difficult to interpret in words or pictures, because they are not based on subgraph counts, and the in-isolation effects are especially difficult. The interpretation can best be understood from the change statistics. The anti in-isolates effect has a change statistic of +1 when actor i creates a link to some other actor who previously had indegree 0; and (by implication) a change statistic of –1 when the actor was the only one to link to some other actor and now drops the link;

68. *anti in-near-isolates effect = indegree at least 2 effect*, (`antiInIso2 = in2Plus`), the effect of wishing to make a new connection to others who currently have an indegree equal to 1, and wishing not to sever connections to others who currently have an indegree equal to 2:
$s^{\text{net}}_{i68}(x) = \sum_j I\{x_{+j} \geq 2\}$;
the anti in-near-isolates effect has a contribution of +1 when actor i creates a link to some actor who previously had indegree 1, and (by implication) –1 when the actor drops a link to some other actor who before this had indegree +2;

69. *indegree at least 3 effect*, (`in3Plus`), the effect of wishing to make a new connection to others who currently have an indegree equal to 2, and wishing not to sever connections to others who currently have an indegree equal to 3:
$s^{\text{net}}_{i69}(x) = \sum_j I\{x_{+j} \geq 3\}$;

70. *isolate popularity effect*, (`isolatePop`), the effect of being tied to actors who further are isolates (the fact that such a tie does exist will give the other actor an in-degree of 1),
$s_{i70}^{\text{net}}(x) = \sum_j x_{ij}\, I\{x_{+j} = 1, x_{j+} = 0\}$.
Note that perhaps this effect is of limited use, as other (third) actors might increase the indegree of j to more than 1, and then the ex-isolate does not contribute any more to i's evaluation of the network; the three effects above ('anti isolates', 'anti in-isolates', and 'anti in-near-isolates') may be more useful instead.

Note that the network-isolate effect expresses the tendency for ego to be an isolate (not sending ties if ego has indegree 0), whereas the in-isolate and isolate popularity effect express the tendency for ego to connect to others who, without this connection, would have an indegree of 0, or be total isolates, respectively. Thus, in modeling the number of isolates, for the network-isolate effect the agency is in the isolate (see the i in the formula), whereas for the various anti isolates and the isolate popularity effects the agency is in others connecting (or not) to the isolate (see the j in the formulae).

71. only in RSienaTest: *Simmelian outdegree effect*, (`simmelian`), the effect of the Simmelian outdegree. The Simmelian transformation of the network is the network composed of all ties embedded in at least one complete triad:

$$x_{ij}^{\text{simm}} = \begin{cases} 1 & x_{ij} = x_{ji} = 1 \text{ and} \\ & \quad \text{there is at least one } h \text{ for which } x_{hi} = x_{ih} = x_{hj} = x_{jh} = 1 \\ 0 & \text{else.} \end{cases}$$

In other words, these ties must be reciprocal, and there must be at least one third actor to whom both have a mutual tie. The Simmelian outdegree of i is
$s_{i71}^{\text{net}}(x) = \sum_j x_{ij}^{\text{simm}}$.

13.1.1.2 Dyadic covariate effects

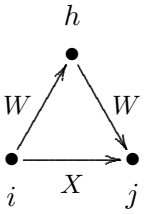
The effects for a dyadic covariate w_{ij} are

72. *covariate (centered) main effect* (`X`),
$s_{i72}^{\text{net}}(x) = \sum_j x_{ij}\,(w_{ij} - \bar{w})$
where $\bar{w}$ is the mean value of w_{ij} ;

73. *covariate (centered)* $\times$ *reciprocity* (`XRecip`),
$s_{i73}^{\text{net}}(x) = \sum_j x_{ij}x_{ji}\,(w_{ij} - \bar{w})$.

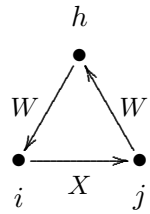
⊙ Various different ways can be modeled in which a triadic combination can be made between the dyadic covariate and the network. In the explanation, the dyadic covariate is regarded as a weighted network (which will be reduced to a non-weighted network if w_{ij} only assumes the values 0 and 1). By way of exception, the dyadic covariate is not centered in these effects (to make it better interpretable as a network). In the text and the pictures, an arrow with the letter W represents a tie according to the weighted network W.

74. WW => X *closure of covariate* (WWX),
$s^{\text{net}}_{i74}(x) = \sum_{j,h} x_{ij}\, w_{ih}\, w_{hj}$;
this refers to the closure of $W - W$ two-paths; each $W - W$ two-path $i \overset{W}{\to} h \overset{W}{\to} j$ is weighted by the product $w_{ih}w_{hj}$ and the sum of these product weights measures the strength of the tendency toward closure of these $W - W$ twopaths by a tie.

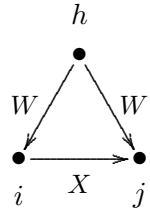


Since the dyadic covariates are represented by square arrays and not by edgelists, this and the following effects will be relatively time-consuming if the number of nodes is large.

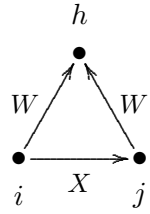
75. *mixed cyclic* WW => X *closure*, (X: cyclic closure of W) (cyWWX)
$s^{\text{net}}_{i75}(x) = \sum_{j,h} x_{ij}\, w_{jh}\, w_{hi}$;
this refers to the cyclic closure of $W - W$ two-paths (weighted if the dyadic covariate W does not only have 0 and 1 values); the contribution of the tie $i \overset{X}{\to} j$ is proportional to the number of product of weights of $W - W$ two-paths $j \overset{W_{jh}}{\longrightarrow} h \overset{W_{hi}}{\longrightarrow} i$;



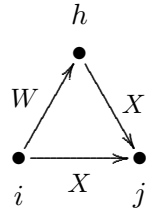
76. *incoming shared* WWX, (X: incoming shared W) (InWWX)
$s^{\text{net}}_{i76}(x) = \sum_{j,h} x_{ij}\, w_{hi}\, w_{hj}$;
this refers to shared incoming W ties contributing to the tie $i \overset{X}{\to} j$;



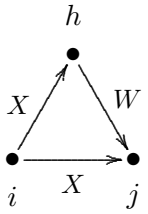
77. *incoming shared* WWX, (X: incoming shared W) (OutWWX)
$s^{\text{net}}_{i77}(x) = \sum_{j,h} x_{ij}\, w_{ih}\, w_{jh}$;
this refers to shared outgoing W ties contributing to the tie $i \overset{X}{\to} j$;



78. WX => X *closure of covariate* (WXX),
$s^{\text{net}}_{i78}(x) = \sum_{j,h} x_{ij}\, w_{ih}\, x_{hj}$;
this refers to the closure of mixed $W - X$ two-paths; each $W - X$ two-path $i \overset{W}{\to} h \to j$ is weighted by w_{ih} and the sum of these weights measures the strength of the tendency toward closure of these mixed $W - X$ twopaths by a tie;
(this effect is implemented as an elementary effect; it may be noted that the corresponding evaluation effect is the (WXX) effect for $W + t(W)$, where $t(W)$ is the transpose of W);



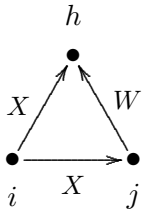
79. *$XW =>$ X closure of covariate* `(XWX)`,
$s^{\text{net}}_{i79}(x) = \sum_{j,h} x_{ij}\, x_{ih}\, w_{hj}$;
this refers to the closure of mixed $X - W$ two-paths; each $X - W$ two-path $i \to h \overset{W}{\to} j$ is weighted by w_{hj} and the sum of these weights measures the strength of the tendency toward closure of these mixed $X - W$ twopaths by a tie.
The covariate W here always is used non-centered.



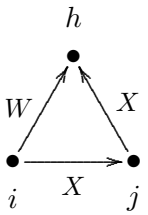
There are two partial variants of this effect; they can be distinguished not by the Method of Moments, but only by Maximum Likelihood and Bayesian estimation.

80. *$XW =>$ X closure-1 of covariate* `(XWX1)`,
This is an elementary effect, not an evaluation effect, comprising of the '$XW =>$ X closure of covariate' effect only the contribution of the the number of weighted $X - W$ two-paths $i \overset{X}{\to} h \overset{W}{\to} j$. In other words, only the $i \to j$ tie in the figure above is the dependent variable here. The effect is defined as
$s^{\text{el}}_{i80}(x) = x_{ij} \sum_{h;h\neq j} x_{ih}\, w_{hj}$;
The covariate W here always is used non-centered.

81. *$XW =>$ X closure-2 of covariate* `(XWX2)`,
This is an elementary effect, not an evaluation effect, comprising of the '$XW =>$ X closure of covariate' effect only the contribution of the number of weighted $X - W$ two-in-stars $i \overset{X}{\to} h$, $j \overset{W}{\to} h$. In other words, only the $i \to j$ tie in the figure here to the right is the dependent variable. The effect is defined as
$s^{\text{el}}_{i81}(x) = x_{ij} \sum_{h;h\neq j} x_{ih}\, w_{jh}$.
The covariate W here always is used non-centered.



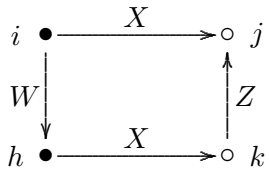
82. *covariate-weighted twopaths* `(XXW)`,
$s^{\text{net}}_{i82}(x) = \sum_{j,h} x_{ij}\, x_{jh}\, w_{ih}$;
this refers to two-paths weighted by the covariate: each $X - X$ two-path $i \overset{X}{\to} j \overset{X}{\to} h$ is weighted by the value w_{ih} for the pair of nodes at the start and the end of the two-path.
The covariate W here always is used non-centered.



13.1.1.3 Dyadic double covariate effects

For pairs of one-mode dyadic covariates W and Z with values w_{ij} and z_{ij}, the following effect is available. It can be useful, e.g., for bipartite (two-mode) networks. Then W must be a dyadic covariate for the first mode, and Z for the second mode. For one-mode networks X, the same dyadic covariate can be used for W and Z.

83. *WXZ => X doubly (W, Z) mixed four-cycle closure* (`WXZX`),
$s^{\text{el}}_{i83}(x) = x_{ij} \sum_{k,h;\text{all different}} w_{ih}\, x_{hk}\, z_{kj}$;
The covariates W and Z here always are used non-centered.
This is programmed as an elementary effect for
dependent tie variable x_{ij}.



13.1.1.4 Monadic covariate effects

For actor-dependent covariates v_j (recall that these are centered internally by SIENA, unless they are constructed with `centered = FALSE`), as well as for dependent behavior variables (for notational simplicity here also denoted v_j; these variables also are centered), the following effects are available:

84. *covariate-alter* or *covariate-related popularity* (`altX`), defined by the sum of the covariate over all actors to whom i has a tie,
$s^{\text{net}}_{i84}(x) = \sum_j x_{ij}\, v_j$;

85. *covariate squared - alter* or *squared covariate-related popularity* (`altSqX`), defined by the sum of the squared covariate over all actors to whom i has a tie, (not included if the variable has range less than 2)
$s^{\text{net}}_{i85}(x) = \sum_j x_{ij}\, v_j^2$;

86. *right threshold covariate alter* (`altRThresholdX`), defined by the number of actors to whom i has a tie and whose covariate value is greater than (to the right of) or equal to a threshold p,
$s^{\text{net}}_{i86}(x) = \sum_j x_{ij}\, I\{v_j \geq p\}$;
for an actor covariate this (and the following effect) can also be obtained as `altX` for a transformed covariate, but this would be impossible for a behavioral dependent variable V;
note that behavioral dependent variables always are centered!

87. *left threshold covariate alter* (`altLThresholdX`), defined by the number of actors to whom i has a tie and whose covariate value is smaller than (to the left of) or equal to a threshold p,
$s^{\text{net}}_{i87}(x) = \sum_j x_{ij}\, I\{v_j \leq p\}$;
note that behavioral dependent variables always are centered; i.e., the overall mean is subtracted!

88. *outdegree to actors with positive covariate more than* p, `outXMore`, where p is some constant (internal effect parameter, see above), defined by
$s^{\text{net}}_{i88}(x) = \max\big((\sum_j x_{ij}\, I\{v_j > 0\}) - p,\, 0\big)$;
this is meant primarily for non-centered dummy variables v_j;

89. *covariate-ego* or *covariate-related activity* `(egoX)`, defined by i's out-degree weighted by his covariate value,
$s^{\text{net}}_{i89}(x) = v_i\, x_{i+}$;

90. *covariate squared - ego* or *squared covariate-related activity* `(egoSqX)`, defined by i's out-degree weighted by his covariate value squared,
$s^{\text{net}}_{i90}(x) = v_i^2\, x_{i+}$;

91. *right threshold covariate ego* `(egoRThresholdX)`, defined by i's out-degree when i's covariate value is greater than (to the right of) or equal to a threshold p,
$s^{\text{net}}_{i91}(x) = I\{v_i \geq p\}\, x_{i+}$;
for an actor covariate this (and the following effect) can also be obtained by transforming the covariate, but this would be impossible for a behavioral dependent variable V;
note that behavioral dependent variables always are centered!

92. *left threshold covariate ego* `(egoLThresholdX)`, defined by i's out-degree when i's covariate value is less than (to the right of) or equal to a threshold p,
$s^{\text{net}}_{i92}(x) = I\{v_i \leq p\}\, x_{i+}$;
note that behavioral dependent variables always are centered!

93. *absolute difference outdegree – covariate* `(degAbsDiffX)`, defined by the absolute difference between i's out-degree and his covariate value,
$s^{\text{net}}_{i93}(x) = \mid x_{i+} - \lfloor v_i \rfloor \mid$;
here $\lfloor v_i \rfloor$ is v_i truncated to the largest integer not larger than v_i; working with a non-centered covariate may be advisable for this effect;

94. *positive difference outdegree – covariate* `(degPosDiffX)`, defined by the positive part of the difference between i's out-degree and her covariate value,
$s^{\text{net}}_{i94}(x) = \max\{x_{i+} - \lfloor v_i \rfloor, 0\}$;
here $\lfloor v_i \rfloor$ is v_i truncated to the largest integer not larger than v_i; working with a non-centered covariate may be advisable for this effect;

95. *negative difference outdegree – covariate* `(degNegDiffX)`, defined by the negative part of the difference between i's out-degree and her covariate value,
$s^{\text{net}}_{i95}(x) = -\min\{x_{i+} - \lfloor v_i \rfloor, 0\} = \max\{\lfloor v_i \rfloor - x_{i+}, 0\}$;
here $\lfloor v_i \rfloor$ is v_i truncated to the largest integer not larger than v_i; working with a non-centered covariate may be advisable for this effect;

96. *covariate-related similarity* `(simX)`, defined by the sum of centered similarity scores sim^v_{ij} between i and the other actors j to whom he is tied,
$s^{\text{net}}_{i96}(x) = \sum_j x_{ij}(\text{sim}^v_{ij} - \widehat{\text{sim}^v})$,
where $\widehat{\text{sim}^v}$ is the mean of all similarity scores, which are defined as $\text{sim}^v_{ij} = \frac{\Delta - |v_i - v_j|}{\Delta}$ with $\Delta = \max_{ij} |v_i - v_j|$ being the observed range of the covariate v (this mean is given in the output file just before the 'initial data description'; it is also given, e..g., for data set `mydata` and constant covariate `mycov`, by `attr(mydata$cCovars$mycov, "simMean")`);
the definition of sim^v_{ij} is discussed in section 5.2, see (1);

97. *covariate-difference* or *covariate-related difference* (`diffX`), defined by the alter-minus-ego difference of the covariate over all actors to whom i has a tie,
$s^{\text{net}}_{i97}(x) = \sum_j x_{ij}\,(v_j - v_i)$;

98. *absolute covariate-difference* or *covariate-related absolute difference* (`absDiffX`), defined by the absolute value of the alter-ego difference of the covariate over all actors to whom i has a tie,
$s^{\text{net}}_{i98}(x) = \sum_j x_{ij}\,|\,v_j - v_i\,|$;

99. *covariate-squared-difference* or *covariate-related squared difference* (`diffSqX`), defined by the squared alter-minus-ego difference of the covariate over all actors to whom i has a tie,
$s^{\text{net}}_{i99}(x) = \sum_j x_{ij}\,(v_j - v_i)^2$;

100. *covariate-ego* $\times$ *difference* or *covariate-related ego-difference interaction* (`egoDiffX`), defined by ego's value times the alter-minus-ego difference of the covariate over all actors to whom i has a tie,
$s^{\text{net}}_{i100}(x) = \sum_j x_{ij}\,v_i\,(v_j - v_i)$;

101. *covariate-ego* $\times$ *alter* (`egoXaltX`), defined by the product of i's covariate and the sum of those of his alters,
$s^{\text{net}}_{i101}(x) = v_i \sum_j x_{ij}\,v_j$;

102. *sign(alter – ego) for covariate* (`altHigherEgoX`), defined by the number of ties $i \to j$ where j's covariate is larger than i's minus the number of ties where j's covariate is larger than i's,
$s^{\text{net}}_{i102}(x) = \sum_j x_{ij}\,\text{sign}(v_j - v_i)$,
where $\text{sign}(d) = 1$ for $d > 0$, 0 for $d = 0$, and –1 for $d < 0$;

103. *ego* $>$ *alter for covariate* (`higher`), defined by the number of ties where i's covariate is larger than alter's, while equality counts for half,
$s^{\text{net}}_{i103}(x) = \sum_j x_{ij}\,\text{dsign}(v_i - v_j)$,
where $\text{dsign}(d) = 0$ for $d < 0$, 0.5 for $d = 0$, and 1 for $d > 0$;

104. *covariate-ego* $\times$ *alter* $\times$ *reciprocity* (`egoXaltXRecip`), defined by the product of i's covariate and the sum of those of his reciprocated alters,
$s^{\text{net}}_{i104}(x) = v_i \sum_j x_{ij}\,x_{ji}\,v_j$;

105. *covariate-related similarity* $\times$ *reciprocity* (`simRecipX`), defined by the sum of centered similarity scores for all reciprocal dyads in which i is situated,
$s^{\text{net}}_{i105}(x) = \sum_j x_{ij}x_{ji}(\text{sim}^v_{ij} - \widehat{\text{sim}^v})$;

106. *same covariate*, which can also be called *covariate-related identity* (`sameX`), defined by the number of ties of i to all other actors j who have exactly the same value on the covariate,
$s^{\text{net}}_{i106}(x) = \sum_j x_{ij}\,I\{v_i = v_j\}$,
where the indicator function $I\{v_i = v_j\}$ is 1 if the condition $\{v_i = v_j\}$ is satisfied, and 0 if it is not;

for equality between different variables for ego and alter, there is the `sameXV` effect, mentioned below;

107. *same covariate* $\times$ *reciprocity* (`sameXRecip`) defined by the number of reciprocated ties between i and all other actors j who have exactly the same value on the covariate,
$s^{\text{net}}_{i107}(x) = \sum_j x_{ij} x_{ji}\, I\{v_i = v_j\}$;

108. *different covariate* (`unequalX`), defined by the number of ties of i to all other actors j who have a different value on the covariate,
$s^{\text{net}}_{i108}(x) = \sum_j x_{ij}\, I\{v_i \neq v_j\}$,
where the indicator function $I\{v_i \neq v_j\}$ is 1 if the condition $\{v_i \neq v_j\}$ is satisfied, and 0 if it is not;
this is the converse of the `sameX` effect, and can be useful in interactions;

109. *in-degree popularity weighted by sender's V* (`inPopX`)
defined by the sum of values of V for actors h for which there are ties $i \rightarrow j \leftarrow h$, i.e., an in-two-star; the value $\sum_h x_{hj}\, v_h$ can be regarded as a V-weighted version of the indegree of j:
$s^{\text{net}}_{i109}(x) = \sum_j x_{ij} \sum_h x_{hj}\, v_h$
or for $p = 2$
$s^{\text{net}}_{i109}(x) = \sum_j x_{ij} \sqrt{\sum_h x_{hj}\, v_h}$;
note that to use this effect for $p = 2$ the variable V must be nonnegative, which implies that it must be non-centered;
further note that if V is a behavioral variable, it will be used in the centered version, so that p must be 1; if weighting by a non-centered behavioral variable is desired, a new effect is necessary;

110. *in-degree activity weighted by sender's V* (`inActX`)
defined by the sum of values of V for actors h for which there are ties $h \rightarrow i \rightarrow j$, i.e., a two-path with i in the middle; the value $\sum_h x_{hi}\, v_h$ can be regarded as a V-weighted version of the indegree of i:
$s^{\text{net}}_{i110}(x) = \sum_j x_{ij} \sum_h x_{hi}\, v_h$
or for $p = 2$
$s^{\text{net}}_{i110}(x) = \sum_j x_{ij} \sqrt{\sum_h x_{hi}\, v_h}$;
note that to use this effect for $p = 2$ the variable V must be nonnegative, which implies that it must be non-centered;

111. *out-degree popularity weighted by receiver's V* (`outPopX`)
defined by the sum of values of V for actors h for which there are ties $i \rightarrow j \rightarrow h$, i.e., at outdistance 2; the value $\sum_h x_{jh}\, v_h$ can be regarded as a V-weighted version of the outdegree of j:
$s^{\text{net}}_{i111}(x) = \sum_j x_{ij} \sum_h x_{jh}\, v_h$
or for $p = 2$
$s^{\text{net}}_{i111}(x) = \sum_j x_{ij} \sqrt{\sum_h x_{jh}\, v_h}$;
note that to use this effect for $p = 2$ the variable V must be nonnegative, which implies that it must be non-centered;

112. *out-degree activity weighted by receiver's V* `(outActX)`
defined by the sum of values of V for actors h for which there are ties $h \leftarrow i \rightarrow j$, i.e., an out-two-star; the value $\sum_h x_{ih}\, v_h$ can be regarded as a V-weighted version of the outdegree of i:
$s^{\text{net}}_{i112}(x) = \sum_j x_{ij} \sum_h x_{ih}\, v_h$
or for $p = 2$
$s^{\text{net}}_{i112}(x) = \sum_j x_{ij} \sqrt{\sum_h x_{ih}\, v_h}$;
note that to use this effect for $p = 2$ the variable V must be nonnegative, which implies that it must be non-centered.

113. *indegree popularity from same covariate* `(sameXInPop)` defined by the sum over j of outgoing ties $i \rightarrow j$ weighted by the number of actors with the same covariate value as i who also have a tie to j:
$s^{\text{net}}_{i113}(x) = \sum_j x_{ij} \sum_h x_{hj}\, I\{v_i = v_h\}$;
for internal effect parameter $p = 2$, the square root of the sum over h is taken;
for internal effect parameter $p = 3$, available only for covariates with integer values from 0 to 20, the weights are the proportion of actors with the same covariate value as i who also have a tie to j:
$$s^{\text{net}}_{i113}(x) = \sum_j x_{ij} \frac{\sum_h x_{hj}\, I\{v_i = v_h\}}{\sum_h I\{v_i = v_h\}};$$
for internal effect parameter $p = 4$, the square root of the ratio of sums is taken;

⊙ the analogue of the `sameXOutAct` effect (see below) (restricting indegree popularity to only within V-groups) is the interaction of `(sameXInPop)` with `sameX`, for the same covariate:
$s^{\text{net}}_{i113}(x) = \sum_j x_{ij}\, I\{v_i = v_j\} \sum_h x_{hj}\, I\{v_i = v_h\}$;

114. *indegree popularity from different covariate* `(diffXInPop)` defined by the number of incoming ties received by those to whom i is tied and sent by others who have a different covariate value than i,
$s^{\text{net}}_{i114}(x) = \sum_j x_{ij} \sum_h x_{hj}\, I\{v_i \neq v_h\}$;
for internal effect parameter $p = 2$, the square root of the sum over h is taken;
for internal effect parameter $p = 3$, available only for covariates with integer values from 0 to 20, it is defined by
$$s^{\text{net}}_{i114}(x) = \sum_j x_{ij} \frac{\sum_h x_{hj}\, I\{v_i \neq v_h\}}{\sum_h I\{v_i \neq v_h\}};$$
for internal effect parameter $p = 4$, the square root of the ratio of sums is taken;

115. *outdegree activity to same covariate* `(sameXOutAct)` defined by the squared number of ties to those others who have the same covariate value as i,
$s^{\text{net}}_{i115}(x) = \big(\sum_j x_{ij}\, I\{v_i = v_j\}\big)^2$;
for internal effect parameter $p = 2$, it is defined by
$s^{\text{net}}_{i115}(x) = \sum_j x_{ij}\, I\{v_i = v_j\} \sqrt{\sum_h x_{ih}\, I\{v_i = v_h\}}$;

116. *outdegree activity to different covariate* (`diffXOutAct`) defined by the squared number of ties to those others who have a different covariate value than i,
$s^{\text{net}}_{i116}(x) = \left(\sum_j x_{ij}\, I\{v_i \neq v_j\}\right)^2$;
for internal effect parameter $p = 2$, it is defined by
$s^{\text{net}}_{i116}(x) = \sum_j x_{ij}\, I\{v_i \neq v_j\} \sqrt{\sum_h x_{ih}\, I\{v_i \neq v_h\}}$;

117. *outdegree activity different-same covariate* (`crossXOutAct`) defined by the combination of ties to others who have a different covariate value than i and others who have the same covariate value as i. The internal effect parameter p has three values, depending on where the square root is taken. The effect is defined by

$$s^{\text{net}}_{i117}(x) = \sum_j x_{ij}\, I\{v_i = v_j\} \sum_h x_{ih}\, I\{v_i \neq v_h\}\,, \qquad (p = 1)$$

$$s^{\text{net}}_{i117}(x) = \sum_j x_{ij}\, I\{v_i = v_j\} \sqrt{\sum_h x_{ih}\, I\{v_i \neq v_h\}}\,, \qquad (p = 2)$$

$$s^{\text{net}}_{i117}(x) = \sum_j x_{ij}\, I\{v_i \neq v_j\} \sqrt{\sum_h x_{ih}\, I\{v_i = v_h\}}\,; \qquad (p = 3)$$

118. *outdegree activity to homogeneous covariate* (`homXOutAct`) defined by the sum of outgoing ties weighted by the number of outgoing ties to those with the same covariate value (as alter),
$s^{\text{net}}_{i118}(x) = \sum_j x_{ij} \sum_h x_{ih}\, I\{v_h = v_j\}$;

119. *outdegree activity to homogeneous covariate* (`homXOutAct2`),
almost the same as `homXOutAct`, but much more efficiently programmed, and only for covariates with integer values from 0 to 20; defined by the sum of outgoing ties weighted by the number of outgoing ties to those with the same covariate value (as alter), but it should be positive:
$s^{\text{net}}_{i119}(x) = \sum_j x_{ij} \sum_h x_{ih}\, I\{v_h = v_j > 0\}$;

The effect parameter p can take the values 1, 2, 3 and 4. The formula given above is for $p = 1$.
For $p = 3$, the effect is defined by the sum of i's outgoing ties, where the tie to j is weighted by the fraction of alters with the same covariate value as j who receive a tie from i:

$$s^{\text{net}}_{i119}(x) = \sum_j x_{ij}\, \frac{\sum_h x_{ih}\, I\{v_h = v_j > 0\}}{\sum_h I\{v_h = v_j > 0\}}\,. \qquad (p = 3)$$

For $p = 2$ and $p = 4$, the square root is taken:

$$s^{\text{net}}_{i119}(x) = \sum_j x_{ij} \sqrt{\sum_h x_{ih}\, I\{v_h = v_j > 0\}} \qquad (p = 2)$$

and

$$s^{\text{net}}_{i119}(x) = \sum_j x_{ij} \sqrt{\frac{\sum_h x_{ih}\, I\{v_h = v_j > 0\}}{\sum_h I\{v_h = v_j > 0\}}}\,. \qquad (p = 4)$$

The default is $p = 3$. Covariate values equal to 0 are excluded as a way of dealing with missing covariate data.

120. *outdegree activity weighted by alter's covariate* (`altXOutAct`) defined by the squared sum of ties weighted by alter's covariate values,
$s_{i120}^{\text{net}}(x) = \left(\sum_j x_{ij}\, v_j\right)^2$;
since the covariate here is used as a weight, this probably makes sense especially for non-centered covariates;

121. *reciprocal degree activity to same covariate* (`sameXReciAct`) defined by the outdegree of i, multiplied by the number of i's reciprocated ties to others who have the same covariate value as i,
$s_{i121}^{\text{net}}(x) = \sum_j x_{ij} \sum_h x_{ih}\, x_{hi}\, I\{v_i = v_h\}$;

122. *reciprocal degree activity to different covariate* (`diffXReciAct`) defined by the outdegree of i, multiplied by the number of i's reciprocated ties to others who have a different covariate value than i,
$s_{i122}^{\text{net}}(x) = \sum_j x_{ij} \sum_h x_{ih}\, x_{hi}\, I\{v_i \neq v_h\}$;

123. only in RSienaTest: *Simmelian alter X effect*, (`simmelianAltX`), the effect of Simmelian ties weighted by alter's value of X. The Simmelian transformation of the network is the network composed of all ties embedded in at least one complete triad:

$$x_{ij}^{\text{simm}} = \begin{cases} 1 & x_{ij} = x_{ji} = 1 \text{ and} \\ & \quad \text{there is at least one } h \text{ for which } x_{hi} = x_{ih} = x_{hj} = x_{jh} = 1 \\ 0 & \text{else.} \end{cases}$$

In other words, these ties must be reciprocal, and there must be at least one third actor to whom both have a mutual tie. The Simmelian alter V effect is
$s_{i123}^{\text{net}}(x) = \sum_j x_{ij}^{\text{simm}}\, v_j$.

124. *covariate* $\times$ *transitive triplets* (`transTripX`), defined by the sum of transitive triplets $i \to h \to j \leftarrow i$ weighted by the covariate value for the intermediate actor h,
$s_{i124}^{\text{net}}(x) = \sum_j v_h\, x_{ij}\, x_{ih}\, x_{hj}$;
this is meaningful especially if covariate V is non-centered; else some weights would be negative, and the term 'weight' would not be appropriate. Note that weighting by the covariate values for i and/or j can be obtained by 'user-defined' interactions of `transTrip` with `egoX` and/or `altX`.

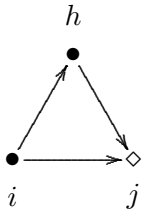
125. *same covariate* $\times$ *transitive triplets* (`sameXTransTrip`), defined by the number of transitive triplets $i \to h \to j \leftarrow i$ that have the same covariate value for i and j,
$s_{i125}^{\text{net}}(x) = \sum_j x_{ij}\, x_{ih}\, x_{hj}\, I\{v_i = v_j\}$;

126. *different covariate* $\times$ *transitive triplets* (`diffXTransTrip`), defined by the number of transitive triplets $i \to h \to j \leftarrow i$ that have different covariate values for i and j,
$s_{i126}^{\text{net}}(x) = \sum_j x_{ij}\, x_{ih}\, x_{hj}\, I\{v_i \neq v_j\}$;

127. *covariate-related similarity* $\times$ *transitive triplets* (`simXTransTrip`), defined by the sum of transitive triplets $i \to h \to j \leftarrow i$ weighted by the centered similarity scores between i and j,
$s_{i127}^{\text{net}}(x) = \sum_j x_{ij}\, x_{ih}\, x_{hj} (\text{sim}_{ij}^{v} - \widehat{\text{sim}^{v}})$;

128. *homogeneous covariate × transitive triplets* (`homXTransTrip`), defined by the number of transitive triplets $i \to h \to j \leftarrow i$ that have the same covariate value for i, j, and h,
$s^{\text{net}}_{i128}(x) = \sum_j x_{ij}\, x_{ih}\, x_{hj}\, I\{v_i = v_j = v_h\}$;

129. *transitive triplets jumping to different V* (`jumpXTransTrip`),
$s^{\text{net}}_{i129}(x) = \sum_{j,h} x_{ij}\, x_{ih}\, x_{hj}\, I\{v_i = v_h \neq v_j\}$;
this refers to transitive closure, restricted to 'jump outside of V-groups' in the sense that the focal actor and the mediating actor have the same value of V, but the target actor has a different value;



130. *same covariate × transitive reciprocated triplets* (`sameXTransRecTrip`), defined by the number of transitive reciprocated triplets $i \to h \to j \leftrightarrow i$ that have the same covariate value for i and j,
$s^{\text{net}}_{i130}(x) = \sum_j x_{ij}\, x_{ji}\, x_{ih}\, x_{hj}\, I\{v_i = v_j\}$;

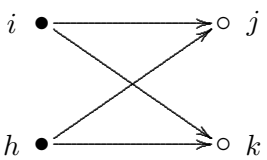
131. *homogeneous covariate × transitive reciprocated triplets* (`homXTransRecTrip`), defined by the number of transitive reciprocated triplets $i \to h \to j \leftrightarrow i$ for which the covariate values for i, j, h all are the same,
$s^{\text{net}}_{i131}(x) = \sum_j x_{ij}\, x_{ji}\, x_{ih}\, x_{hj}\, I\{v_i = v_j = v_h\}$;

132. *different covariate × transitive reciprocated triplets* (`diffXTransRecTrip`), defined by the number of transitive reciprocated triplets $i \to h \to j \leftrightarrow i$ that have different covariate values for i and j,
$s^{\text{net}}_{i132}(x) = \sum_j x_{ij}\, x_{ji}\, x_{ih}\, x_{hj}\, I\{v_i \neq v_j\}$;

133. *ego > alter for covariate* (`higher`), defined by the number of ties where i's covariate is larger than alter's, while equality counts for half,
$s^{\text{net}}_{i133}(x) = \sum_j x_{ij}\, \text{dsign}(v_i - v_j)$,
where $\text{dsign}(d) = 0$ for $d < 0$, 0.5 for $d = 0$, and 1 for $d > 0$.

134. *covariate of indirect ties* (`IndTies`), defined by the sum of the covariate over the actors to whom i is tied indirectly (at a geodesic distance of 2),
$s^{\text{net}}_{i134}(x) = \sum_j (1 - x_{ij}) \big(\max_h x_{ih} x_{hj} \big) v_j$;

135. *number of four-cycles from same covariate* (`sameXCycle4`), defined by the number of four-cycles sent by actors with the same covariate value,
$s^{\text{net}}_{i135}(x) = \frac{1}{4} \sum_{j,k,h;\text{all different}} x_{ij}\, x_{ik}\, x_{hj}\, x_{hk}\, I\{v_i = v_h\}$;
for parameter $p = 2$ the square root is taken.



136. *number of four-cycles to same covariate* (`sameInXCycle4`),
defined by the number of four-cycles sent to actors with the same covariate value,

$s^{\text{net}}_{i136}(x) = \frac{1}{4}\sum_{j,k,h;\text{all different}} x_{ij}\, x_{ik}\, x_{hj}\, x_{hk}\, I\{v_j = v_k\}$;
for parameter $p = 2$ the square root is taken.

137. *group average* (`avGroupEgoX`), defined by ego's value multiplied by the average of the covariate for this period,
$s^{\text{beh}}_{i137}(x, z) = \sum_j x_{ij}\, \overline{v}$;
here $\overline{v}$ is the mean of the values v_h for all actors h in the actor set: for changing covariates, the mean of the wave at the start of the period; for behavioral dependent variables, the current mean. This effect is not meaningful for constant covariates in single-group data. It is useful especially for multi-group data sets, where the average value varies between groups, and where there are no constant covariates in the multi-group object. Note that always, the average $\overline{v}$ is centered by the global mean. For multi-group data, global centering means centering over all groups.

138. *group total* (`totGroupEgoX`), defined by ego's value multiplied by the sum of the covariate for this period,
$s^{\text{beh}}_{i138}(x, z) = \sum_j x_{ij} \sum_h v_h$;
here $\sum_h v_h$ is the sum of the values v_h for all actors h in the actor set: for changing covariates, the total of the wave at the start of the period; for behavioral dependent variables, the current total.

The following group of effects uses an auxiliary variable $\breve{v}_j$ which can be called 'alters' v-average'. It is described as the average value of v_h for those h to whom j is tied, and defined mathematically by

$$\breve{v}_j = \begin{cases} \dfrac{\sum_h x_{jh} v_h}{x_{j+}} & \text{if } x_{j+} > 0 \\ \overline{v} & \text{if } x_{j+} = 0, \end{cases} \tag{22}$$

where $\overline{v}$ is the overall observed mean of V (for changing actor covariates and behavioral variables, the mean over all waves).
(It may be noted that, for centered V, this constructed variable $\breve{v}_j$ will not itself have exactly a zero mean generally.)

Note that this value is being updated during the simulations. Network changes will change $\breve{v}_j$; if v_j is a dependent behavior variable, then behaviour changes will also change $\breve{v}_j$.

In the following list, there is no ego effect, because the ego effect of $\breve{v}_j$ would be the same as the alter effect of v_j.

139. *covariate - alter at distance 2* (`altDist2`). This effect is associated with an effect parameter which can have values 1 or 2. For parameter 1, it is defined as the sum of alters' covariate-average over all actors to whom i has a tie,

$$s^{\text{net}}_{i139}(x) = \sum_j x_{ij} \breve{v}_j \qquad \text{(parameter 1)}$$

For parameter 2, it is defined similarly, but for an alters' covariate-average excluding ego:

$$s^{\text{net}}_{i139}(x) = \sum_j x_{ij} \breve{v}^{(-i)}_j \qquad \text{(parameter 2)}$$

where

$$\breve{v}_j^{(-i)} = \begin{cases} \dfrac{\sum_{h \neq i} x_{jh} v_h}{x_{j+} - x_{ji}} & \text{if } x_{j+} - x_{ji} > 0 \\ \bar{v} & \text{if } x_{j+} - x_{ji} = 0. \end{cases} \tag{23}$$

To compute the contribution for this effect, note that

$$\sum_j x_{ij} \breve{v}_j^{(-i)} = \sum_j x_{ij} \, \frac{x_{j+} \breve{v}_j - x_{ji} v_i}{x_{j+} - x_{ji}}$$

This shows that, given that $\breve{v}_j$ is being updated for all j, the contribution for this effect for parameter 2 can be computed as

$$\frac{x_{j+} \breve{v}_j - x_{ji} v_i}{x_{j+} - x_{ji}}$$

(where 0/0 is interpreted as 0).

140. *total covariate - alter at distance 2* (`totDist2`). This is like the previous effect, but using the total instead of the average value of alter's covariate. For parameter 1, it is defined as

$$s_{i140}^{\text{net}}(x) = \sum_j x_{ij} \, x_{j+} \breve{v}_j \qquad \text{(parameter 1)}$$

and for parameter 2, as

$$s_{i140}^{\text{net}}(x) = \sum_j x_{ij} (x_{j+} - x_{ji}) \breve{v}_j^{(-i)} \qquad \text{(parameter 2)}$$

where $\breve{v}_j$ and $\breve{v}_j^{(-i)}$ are as above.

141. *ego-(alter-distance-2) covariate - similarity* (`simEgoDist2`), defined as the sum of centered similarity between i and alters' covariate-average, for all actors j to whom i has a tie,

$$s_{i141}^{\text{net}}(x) = \sum_j x_{ij} \left(\text{sim}(\grave{v})_{ij} - \widehat{\text{sim}^v} \right),$$

where the similarity scores $\text{sim}(\grave{v})_{ij}$ are defined as

$$\text{sim}(\grave{v})_{ij} = \frac{\Delta - |v_i - \breve{v}_j^{(-i)}|}{\Delta},$$

where $\Delta = \max_{ij} |v_i - v_j|$ is the observed range of the *original* covariate v, $\breve{v}_j^{(-i)}$ is as above in effect (`altDist2`) and $\widehat{\text{sim}^v}$ is the mean of all similarity scores as used also for the `simX` effect; this centering is applied since version 1.1-285. For a constant covariate `mycov`, this mean is given by `attr(mydata$cCovars$mycov, "simMean")`.

142. *covariate - similarity at distance 2* `(simDist2)` , defined as the sum of centered similarity values for alters' covariate-average between i and all actors j to whom i has a tie,

$$s^{\text{net}}_{i142}(x) = \sum_j x_{ij} \left(\text{sim}(\breve{v})_{ij} - \widehat{\text{sim}^v} \right) ,$$

where the similarity scores $\text{sim}(\breve{v})_{ij}$ are defined as

$$\text{sim}(\breve{v})_{ij} = \frac{\Delta - |\breve{v}_i - \breve{v}_j|}{\Delta} ,$$

while $\Delta = \max_{ij} |v_i - v_j|$ is the observed range of the *original* covariate v and $\widehat{\text{sim}^v}$ is the mean of all similarity scores as used also for the `simX` effect; this centering is applied since version 1.1-285. For a constant covariate `mycov`, this mean is given by `attr(mydata$cCovars$mycov, "simMean")`.
Note that for both ego (i) and alter (j) their alters' covariate-average is used, so that this effect is about a comparison between the out-neighbourhoods of i and j.

143. *ego-(alter-distance-2) same covariate* `(sameEgoDist2)` , defined as the sum, for all actors j to whom i has a tie, of the proportion of values v_h equal to v_i in their out-neighbourhoods (i.e., for all actors h, other than i, with ties from j; configurations $i \to j \to h$):

$$s^{\text{net}}_{i143}(x) = \sum_{j;\, j \neq i,\, x_{j+} > x_{ij}} \frac{x_{ij}}{x_{j+} - x_{ij}} \sum_{h \neq i} x_{jh}\, I\{v_h = v_i\} ,$$

where 0/0 is defined as 0; for internal effect parameter $p = 0$ it is defined as the number of out-ties j who have any outgoing tie to an actor h with $v_h = v_i$:

$$s^{\text{net}}_{i143}(x) = \sum_j x_{ij}\, I\Big\{ \Big(\sum_{h \neq i} x_{jh}\, I\{v_h = v_i\} \Big) > 0 \Big\} .$$

144. *covariate - in - alter at distance 2* `(altInDist2)`. This is defined as the sum of alters' values for the average of the covariate values of all their incoming ties, except for the possibly incoming tie from ego:

$$s^{\text{net}}_{i144}(x) = \sum_j x_{ij} \breve{v}^{(-i)}_j$$

where

$$\breve{v}^{(-i)}_j = \begin{cases} \dfrac{\sum_{h \neq i} x_{hj}\, v_h}{x_{+j} - x_{ij}} & \text{if } x_{+j} - x_{ij} > 0 \\ \bar{v} & \text{if } x_{+j} - x_{ij} = 0, \end{cases} \tag{24}$$

where $\bar{v}$ is (again) the global mean of V.

145. *total covariate - in - alter at distance 2* (`totInDist2`). This is defined as the sum of alters' values for the total of the covariate values of all their incoming ties, except for the possibly incoming tie from ego:

$$s^{\text{net}}_{i145}(x) = \sum_j x_{ij}(x_{+j} - x_{ij})\check{v}^{(-i)}_j \;=\; \sum_j x_{ij} \sum_{h\neq i} x_{hj}\, v_h \;.$$

For a non-centered binary variable V with values v_j = 0 or 1, this is a V-restricted version of indegree popularity.

146. *ego-(in-alter-distance-2) covariate - similarity* (`simEgoInDist2`), defined as the sum of centered similarity between i and alters' covariate-in-average, for all actors j to whom i has a tie,

$$s^{\text{net}}_{i146}(x) = \sum_j x_{ij} \left(\text{sim}(\acute{v})_{ij} - \widehat{\text{sim}^v}\right) ,$$

where the similarity scores $\text{sim}(\acute{v})_{ij}$ are defined as

$$\text{sim}(\acute{v})_{ij} = \frac{\Delta - |v_i - \check{v}^{(-i)}_j|}{\Delta} ,$$

while $\check{v}^{(-i)}_j$ is as in the definition of `altInDist2`, $\Delta = \max_{ij} |v_i - v_j|$ is the observed range of the *original* covariate v, and $\widehat{\text{sim}^v}$ is as above (see `simDist2`).
Note that for ego (i) the own value is used and for alter (j) the alters' covariate-average (these are alter's in-alters!), so that this effect is about a comparison between i and the in-neighbourhoods of the j's.

147. *ego-(in-alter-distance-2) same covariate* (`sameEgoInDist2`), defined as the sum, for all actors j to whom i has a tie, of the proportion of values v_h equal to v_i in their in-neighbourhoods (i.e., for all actors h, other than i, who send ties to j; configurations $i \rightarrow j \leftarrow h$):

$$s^{\text{net}}_{i147}(x) = \sum_{j;\, j\neq i,\, x_{+j} > x_{ij}} \frac{x_{ij}}{x_{+j} - x_{ij}} \sum_{h\neq i} x_{hj}\, I\{v_h = v_i\} ,$$

where 0/0 is defined as 0; for internal effect parameter $p = 0$ it is defined as the number of out-ties j who have any incoming tie from an actor h with $v_h = v_i$:

$$s^{\text{net}}_{i147}(x) = \sum_j x_{ij}\, I\Big\{ \Big(\sum_{h\neq i} x_{hj}\, I\{v_h = v_i\} \Big) > 0 \Big\} .$$

13.1.1.5 Monadic double covariate effects

For pairs of actor-dependent covariates V and U with values v_i and u_j, the following effects are available. These are useful, e.g., for bipartite (two-mode) networks. Then V must be a covariate for the first mode, and U for the second mode.

If it is desired to use them for the same variables ($V = U$), it is necessary to define a data set with two identical variables (i.e., with all values equal) bearing different names.

148. *same covariate1–covariate2*, which can also be called *covariate-related identity* `(sameXV)`, defined by the number of ties of i to all other actors j for which the value of i on covariate V is the same as the value of j on covariate U, and larger than 0,
$s_{i148}^{\text{net}}(x) = \sum_j x_{ij}\, I\{v_i = u_j \geq 1\}$,
where the indicator function $I\{v_i = u_j\}$ is 1 if the condition $\{v_i = u_j \geq 1\}$ is satisfied, and 0 if it is not.
This effect is implemented only for U and V that are non-centered categorical actor variables with integer values in the range from 0 to 20.
The exclusion of values 0 is for the treatment of missing values: these can be given `imputationValues` equal to 0, see the help page for as_covariate_rsiena and as_covariate_rsiena.

149. *indegree popularity from same covariate1 to same covariate2*,
`(sameXVInPop)` and `(sameXVInPop2)`. This is an elementary effect that is implemented only for non-centered categorical actor variables with values within the range of numbers from 0 to 20.
It represents the frequency, or relative frequency, of ties from actors in the same category of covariate V to nodes in the same category of covariate U.
It is defined, for the tie from i to j, by the number of ties sent by actors h who have the same non-zero covariate value $v_h = v_i$ as i to those k who have the same non-zero covariate value $u_k = u_j$ as j:
$$s_{i149}^{\text{el}}(x) = x_{ij} \sum_{h,k} x_{hk}\, I\{v_h = v_i \geq 1,\, u_k = u_j \geq 1\}\,, \qquad (p = 1)$$
where the indicator function $I\{v_h = v_i \geq 1,\, u_k = u_j \geq 1\}$ is 1 if the condition $\{v_h = v_i \geq 1,\, u_k = u_j \geq 1\}$ is satisfied, and 0 if it is not.
The exclusion of values 0 is for the treatment of missing values:
these can be given `imputationValues` equal to 0, see the help page for as_covariate_rsiena and as_covariate_rsiena.

The effect parameter p can take the values 1, 2, 3 and 4. The formula given above is for $p = 1$.
For $p = 3$, the effect is defined by the proportion of ties from same-V actors to same-U nodes, as given by the formula
$$s_{i149}^{\text{el}}(x) = x_{ij}\, \frac{\sum_{h,k} x_{hk}\, I\{v_h = v_i \geq 1,\, u_k = u_j \geq 1\}}{\sum_{h,k} I\{v_h = v_i \geq 1,\, u_k = u_j \geq 1\}}\,. \qquad (p = 3)$$
For $p = 2$ and $p = 4$, the square root is taken:
$$s_{i149}^{\text{el}}(x) = x_{ij} \sqrt{\sum_{h,k} x_{hk}\, I\{v_h = v_i \geq 1,\, u_k = u_j \geq 1\}}\,, \qquad (p = 2)$$
and
$$s_{i149}^{\text{el}}(x) = x_{ij} \sqrt{\frac{\sum_{h,k} x_{hk}\, I\{v_h = v_i \geq 1,\, u_k = u_j \geq 1\}}{\sum_{h,k} I\{v_h = v_i \geq 1,\, u_k = u_j \geq 1\}}}\,. \qquad (p = 4)$$

The default is $p = 3$.

This effect is perhaps rather time-consuming because it iterates over all x_{hk}. For this reason,

`sameXVInPop` and `sameXVInPop2` are different implementations of the same effect. The former iterates over all ties, the second over all nodes h, and then over their outgoing ties. You can choose whichever is more efficient for your data set.

Differences between `sameXInPop`, `sameXVInPop`, `sameXCycle4`:
All work with an actor variable V, and `sameXVInPop` works in addition with a nodal variable U.
All three express (for positive parameters) that actors with the same categories of V make related choices of other nodes.
For `sameXInPop`, they tend to choose the same nodes. For `sameXVInPop`, they tend to choose nodes of the same U category. For `sameXCycle4`, if they tend to choose some particular node in common, they will also tend to choose some other nodes in common.

Note effect `homXOutAct2`, which is an ego version of `sameXVInPop`, expressing that ego has differential propensities to ties with alters of different categories; the role of U in `sameXVInPop` is taken by V in the definition of `homXOutAct2`.

13.1.2 Special effects for non-directed networks

For non-directed (symmetric) networks (see Section 6.9 and Snijders and Pickup, 2017), in principle the same effects are used as above; you may search in this manual for 'non-directed' to see the cases where the definition of an effect for non-directed networks deviates from the definition for directed networks. Here we present a few effects that are defined specifically for non-directed networks. The reason is that non-directed data will not allow to differentiate between senders and receivers of ties, and instead of working hypothetically with either ego or alter effects, it may be preferable to utilize effects where ego and alter are assumed to have equal contributions.

1. *degree activity plus popularity effect* (`degPlus`). This is defined as the sum of the in=outdegree popularity and in=outdegree activity effects; equivalently, this effect operates as the degree popularity combined with the degree activity effect under the assumption that the parameters for both are the same:

$$s_{i1}^{\mathrm{net}}(x) = \sum_j x_{ij}(x_{j+} + x_{i+}) \; .$$

   For internal effect parameter equal to 2, the square roots are taken:

$$s_{i1}^{\mathrm{net}}(x) = \sum_j x_{ij}(\sqrt{x_{j+}} + \sqrt{x_{i+}}) \; .$$

2. A centered version of `degPlus`:
   degree activity plus popularity effect (centered) (`degPlus.c`); This is defined as the sum of the centered in=outdegree popularity and in=outdegree activity effects:

$$s_{i2}^{\mathrm{net}}(x) = \sum_j x_{ij}(x_{j+} + x_{i+} - 2d) \; ,$$

   where d is the average degree across all waves.

3. *covariate effect* (`egoPlusAltX`). This is defined as the sum of the covariate-ego and covariate-alter effects; equivalently, this effect operates as the covariate-ego combined with the covariate-alter effect under the assumption that the parameters for both are the same:

$$s_{i3}^{\text{net}}(x) = \sum_j x_{ij}(v_i + v_j) \; .$$

4. *covariate-squared effect* (`egoPlusAltSqX`). This is defined as the sum of the covariate-squared ego and covariate-squared alter effects; equivalently, this effect operates as the covariate-squared ego combined with the covariate-squared alter effect under the assumption that the parameters for both are the same:

$$s_{i4}^{\text{net}}(x) = \sum_j x_{ij}(v_i^2 + v_j^2) \; ;$$

5. *transitive triads effect* (`transTriads`), defined by the number of transitive patterns in i's relations, which for non-directed networks can be written as $s_{i5}^{\text{net}}(x) = \sum_{j<h} x_{ij}\, x_{ih}\, x_{hj}$;

For non-directed networks there is only one degree assortativity effect, the degree assortativity effect which is the outdegree-indegree version[18] still with short name `outInAss`. There is also only one GWESP effect, called simply `gwesp`.

13.1.3 Multiple network effects

An introduction to the analysis of multiple (multivariate) networks, with a discussion of the basic effects, is given in Snijders et al. (2013).

See Section 6.7 for the possibility that two networks have a deterministic constraint by being mutually exclusive, or by one being implied by the other, or (rarely used, if ever) by the union of them being the complete graph.

If there are multiple dependent networks, the definition of cross-network effects is such that always, one network has the role of the dependent variable, while the other network, or networks, have the role of explanatory variable(s). In the following list the network in the role of dependent variable is denoted by the tie variables x_{ij}, while the tie variables w_{ij} denote the network that is the explanatory variable.

Various of these effects are applicable only if the networks X and W satisfy certain conditions of conformability; for example, the first effect of W on X is meaningful only if W and X have the same dimensions, i.e., either both are one-mode networks, or both are two-mode networks with the same actor set for the second mode; as another example, the second effect, of incoming W on X, is applicable only if W and X are one-mode networks. These conditions are hopefully clear from logical considerations, and drawing a little diagram of the involved nodes and arrows will be helpful in cases of doubt.

[18] Of the four versions this has the correct change statistic, because the change in x_{ij} is associated with a change in both the outdegree of i and the indegree of j.

In the RSiena output for groups with multiple networks, the dependent network in each given effect is indicated by the first part of the effect name. In the list below, a more or less normally formulated name is given first, then the name used in RSiena between parentheses, using X as the name for the dependent network and W as the name for the explanatory network, then between parentheses in `typewriter font` the `shortName` as used by RSiena. Since this is a co-evolution model, RSiena will include also the effects where the roles of X and W are reversed.

The first three effects are dyadic. The first can be regarded as a main effect; the reciprocity and mutuality effects will require rather big data sets to be empirically distinguished from each other.

1. *Effect of W on X* (X: W) `(crprod)`,
   $s_{i1}^{\text{net}}(x) = \sum_j x_{ij}\, w_{ij}$;
   $i \overset{W}{\to} j$ leads to $i \overset{X}{\to} j$; this is also called the dyadic entrainment effect;
   if one of the contraints of Section 6.7 is in use, this effect should not be included because its role is taken over by the constraint;

2. *Effect of incoming W on X* (X: reciprocity with W) `(crprodRecip)`,
   $s_{i2}^{\text{net}}(x) = \sum_j x_{ij}\, w_{ji}$;
   this can be regarded as generalized exchange: $j \overset{W}{\to} i$ leads to $i \overset{X}{\to} j$;

3. *Effect of mutual ties in W on X* (X: mutuality with W) `(crprodMutual)`,
   $s_{i3}^{\text{net}}(x) = \sum_j x_{ij}\, w_{ij}\, w_{ji}$;
   $j \overset{W}{\leftrightarrow} i$ leads to $i \overset{X}{\to} j$.

The following are degree-related effects, where nodal degrees in the W network have effects on popularity or activity in the X network. They use an internal effect parameter p, which mostly will be 1 or 2.

To decrease correlation with other effects, the W-degrees are centered by subtracting the value $\bar{w}$, which is the average degree of W across all observations.

4. *Effect of in-degree in W on X-popularity* (X: indegree$^{1/p}$ W popularity) `(inPopIntn)`
   defined by the sum of the W-in-degrees of the others to whom i is tied, for parameter $p = 2$ the square roots of the W-in-degrees:
   $s_{i4}^{\text{net}}(x) = \sum_j x_{ij}\, (w_{+j} - \bar{w})$ or
   for $p = 2$ $s_{i4}^{\text{net}}(x) = \sum_j x_{ij}\, (\sqrt{w_{+j}} - \sqrt{\bar{w}})$;

5. *Effect of in-degree in W on X-activity* (X: indegree$^{1/p}$ W activity) `(inActIntn)`
   defined by the W-in-degrees of i (for $p = 2$ the square roots) times i's X-out-degrees:
   $s_{i5}^{\text{net}}(x) = \sum_j x_{ij}\, (w_{+i} - \bar{w}) = x_{i+}\, (w_{+i} - \bar{w})$ or
   for $p = 2$ $s_{i5}^{\text{net}}(x) = \sum_j x_{ij}\, (\sqrt{w_{+i}} - \sqrt{\bar{w}}) = x_{i+}\, (\sqrt{w_{+i}} - \sqrt{\bar{w}})$;

6. *Effect of out-degree in W on X-popularity* (X: outdegree$^{1/p}$ W popularity) `(outPopIntn)`
   defined by the sum of the W-out-degrees of the others to whom i is tied, for parameter $p = 2$

the square roots of the W-out-degrees:
$s^{\text{net}}_{i6}(x) = \sum_j x_{ij}\,(w_{j+} - \bar{w})$ or
for $p = 2$ $s^{\text{net}}_{i6}(x) = \sum_j x_{ij}\,(\sqrt{w_{j+}} - \sqrt{\bar{w}})$;

7. *Effect of out-degree in W on X-activity* (X: outdegree$^{1/p}$ W activity) `(outActIntn)`
defined by the centered W-out-degree of i (for $p = 2$ its square root, for $p = 0$ its logarithm) times i's X-out-degree:
$s^{\text{net}}_{i7}(x) = \sum_j x_{ij}\,(w_{i+} - \bar{w}) = x_{i+}\,(w_{i+} - \bar{w})$ or
for $p = 2$ $s^{\text{net}}_{i7}(x) = \sum_j x_{ij}\,(\sqrt{w_{i+}} - \sqrt{\bar{w}}) = x_{i+}\,(\sqrt{w_{i+}} - \sqrt{\bar{w}})$;
or for $p = 0$ $s^{\text{net}}_{i7}(x) = \sum_j x_{ij}\,\big(\log(w_{i+}) - \log(\bar{w})\big) = x_{i+}\,\big(\log(w_{i+}) - \log(\bar{w})\big)$,
where $\log(0)$ is defined as 0;

The following four effects are meant mainly for interactions. Recall from Section 6.12 that interactions are defined by the product of the change statistics. In many cases, the change statistics are defined by omitting $\sum_j x_{ij}$ from the formula for the effect. In Section 6.12.1 we saw an example in which the ego effect of the average value of covariate V for those to whom actor i has a W-outgoing tie can be defined as the interaction between `outActIntnX`, the effect of out-degree in W, weighted by receiver's V, on X-activity, with `divOutEgoIntn`. For a given ego, `outActIntnX` gives the sum of the V-values of ego's W-ties, and `divOutEgoIntn` divides this by the number of ego's W-ties; this yields the average of the V-values of ego's W-ties.

For such interaction effects, the internal effect parameters should match; see Section 6.12.1.

8. *1 divided by ego's out-degree in W* (X: 1/outdeg.$^{1/p}$ W ego) `(divOutEgoIntn)`
defined by i's X-out-degree divided by the W-out-degree of i (for $p = 2$ its square root), or 0 if the W-out-degree is 0:
$s^{\text{net}}_{i8}(x) = \sum_j x_{ij}\,\text{divi}(1, w_{i+})$ or
for $p = 2$ $s^{\text{net}}_{i8}(x) = \sum_j x_{ij}\,\text{divi}(1, \sqrt{w_{i+}})$;
where $\text{divi}(a, b)$ is defined as a/b if $b \neq 0$ and 0 if $b = 0$;

9. *1 divided by ego's in-degree in W* (X: 1/indeg.$^{1/p}$ W ego) `(divInEgoIntn)`
defined by i's X-out-degree divided by the W-in-degree of i (for $p = 2$ its square root), or 0 if the W-indegree is 0:
$s^{\text{net}}_{i9}(x) = \sum_j x_{ij}\,\text{divi}(1, w_{+i})$ or
for $p = 2$ $s^{\text{net}}_{i9}(x) = \sum_j x_{ij}\,\text{divi}(1, \sqrt{w_{+i}})$;
where $\text{divi}(a, b)$ is defined as above;

10. *1 divided by alter's out-degree in W* (X: 1/outdeg.$^{1/p}$ W alter) `(divOutAltIntn)`
defined by 1 divided by the sum of the W-out-degrees of those to whom i has an outgoing X-tie (for $p = 2$ its square root), or 0 if the W-out-degree is 0:
$s^{\text{net}}_{i10}(x) = \sum_j x_{ij}\,\text{divi}(1, w_{j+})$ or
for $p = 2$ $s^{\text{net}}_{i10}(x) = \sum_j x_{ij}\,\text{divi}(1, \sqrt{w_{j+}})$;

where divi(a, b) is defined as above;

11. *1 divided by alter's in-degree in W* (X: 1/indeg.$^{1/p}$ W alter) `(divInAltIntn)`
defined by 1 divided by the sum of the W-out-degrees of those from whom i has an incoming X-tie (for $p = 2$ its square root), or 0 if the W-indegree is 0:
$s^{\text{net}}_{i11}(x) = \sum_j x_{ij}$ divi$(1, w_{+j})$ or
for $p = 2$ $s^{\text{net}}_{i11}(x) = \sum_j x_{ij}$ divi$(1, \sqrt{w_{+j}})$;
where divi(a, b) is defined as above;

12. *Effect of average out-degree in W on X* (X: av. degree W) `(avDegIntn)`
defined by the X-outdegree of i multiplied by the average W-out-degree, centered by the internal effect parameter p
$s^{\text{net}}_{i12}(x) = \sum_j x_{ij} \left(\frac{1}{n} \sum_h w_{h+} - p\right) = x_{i+} \left(\frac{1}{n} \sum_h w_{h+} - p\right)$;
this effect is a network-by-period level effect and therefore is useful only for multi-group data sets or for data sets with many periods;
it may be beneficial for convergence to use a value of p that is close to the average W-degree (but constant, while this effect is for the dynamic average outdegree);

13. *Interaction W × in-degree W activity* (X: W x indegree$^{1/p}$ W activity) `(crprodInActIntn)`
defined by the entrainment of W by X weighted by W-in-degrees of i (for $p = 2$ the square roots):
$s^{\text{net}}_{i13}(x) = \sum_j x_{ij}\, w_{ij}\, (w_{+i} - \bar{w})$ or
for $p = 2$ $s^{\text{net}}_{i13}(x) = \sum_j x_{ij}\, w_{ij}\, (\sqrt{w_{+i}} - \sqrt{\bar{w}})$;

14. *Effect of absolute difference of both W out-degrees* (X: outdegree$^{1/p}$ W abs. diff.) `(absOutDiffIntn)`
defined by the sum of the absolute difference of the W-out-degrees of alter and ego for all alters to whom i is tied; for parameter $p = 2$ the square roots of the W-out-degrees are taken:
$s^{\text{net}}_{i14}(x) = \sum_j x_{ij}\, |w_{i+} - w_{j+}|$ or
for $p = 2$ $s^{\text{net}}_{i14}(x) = \sum_j x_{ij}\, |\sqrt{w_{i+}} - \sqrt{w_{j+}}|$;
this can be regarded as minus the similarity of the W-out-degrees (but without the division by the range);

15. *Effect of both in-degrees in W on X-popularity* (X: both indegrees$^{1/p}$ W) `(both)`
defined by the sum of the W-in-degrees of the others to whom i is tied multiplied by the centered W-in-degree of i, for parameter $p = 2$ the square roots of the W-in-degrees:
$s^{\text{net}}_{i15}(x) = \sum_j x_{ij}\, (w_{+i} - \bar{w})\, (w_{+j} - \bar{w})$ or
for $p = 2$ $s^{\text{net}}_{i15}(x) = \sum_j x_{ij}\, (\sqrt{w_{+i}} - \sqrt{\bar{w}})\, (\sqrt{w_{+j}} - \sqrt{\bar{w}})$;
this can be regarded as an interaction between the effect of W-in-degree on X-popularity and the effect of W-in-degree on X-activity;

16. *Duplex XW out-degree activity effect* (X: duplex W outdegree$^{1/p}$ activity)
`(doubleOutAct)`

which is like the out-degree activity effect, but now for the degrees in the joined X and W network; for parameter $p = 2$ the square root of the out-degrees is taken:

$s^{\text{net}}_{i16}(x) = \left(\sum_j x_{ij}\, w_{ij}\right)^2$ or

for $p = 2$ $s^{\text{net}}_{i16}(x) = \sum_j x_{ij}\, w_{ij} \sqrt{\sum_h x_{ih}\, w_{ih}}$;

17. *Duplex XW in-degree popularity effect* (X: duplex W indegree$^{1/p}$ popularity) `(doubleInPop)`
which is like the in-degree popularity effect, but now for the degrees in the joined X and W network; for parameter $p = 2$ the square root of the out-degrees is taken:

$s^{\text{net}}_{i17}(x) = \sum_j x_{ij}\, w_{ij} \sum_h x_{hj}\, w_{hj}$ or

for $p = 2$ $s^{\text{net}}_{i17}(x) = \sum_j x_{ij}\, w_{ij} \sqrt{\sum_h x_{hj}\, w_{hj}}$.

The betweenness effect is another positional effect: a positional characteristic in the W network affects the ties in the X network, but now the position is the betweenness count, defined as the number of pairs of nodes that are not directly connected: $j \overset{W}{\nrightarrow} h$, but that are connected through i: $j \overset{W}{\rightarrow} i \overset{X}{\rightarrow} h$. Again there is an internal effect parameter p, usually 1 or 2.

18. *Effect of W-betweenness on X-popularity* (X: betweenness$^{1/p}$ W popularity) `(betweenPop)` defined by the sum of the W-betweenness counts of the others to whom i is tied:

$s^{\text{net}}_{i18}(x) = \sum_j x_{ij} \left(\sum_{h,k;h\neq k} w_{hj}\, w_{jk}\, (1 - w_{hk})\right)^{1/p}$;

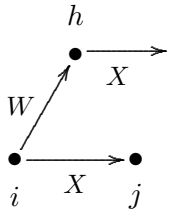
19. mixed twopath activity effect (X: mixed twopath$^{1/p}$ W activity) `(outOutActIntn)` defined by the outdegree of i multiplied by sum of X-outdegrees of those to whom i has a W-tie:
for $p = 1$, this is

$s^{\text{net}}_{i19}(x) = x_{i+} \sum_h w_{ih}\left(x_{h+} - \bar{x}\right)$

and for $p = 2$, there is a square root transformation:

$s^{\text{net}}_{i19}(x) = x_{i+} \sum_h w_{ih}\left(\sqrt{x_{h+}} - \sqrt{\bar{x}}\right)$

where $\bar{x}$ is the average outdegree over all waves.



20. mixed distance-2 out-in activity effect (X: nbr out-in dist.2 W activity) `(nDist2ActIntn)` defined by the outdegree of i multiplied by the number of actors at out-in distance 2 in the W network:
for $p = 1$, this is, for one-mode networks:

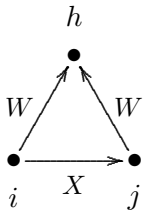
$s^{\text{net}}_{i20}(x) = x_{i+} \sum_h \left\{ \max_k \left(w_{ik}\, w_{hk}\right) - w_{ih} \right\}$

and for two-mode (bipartite) networks

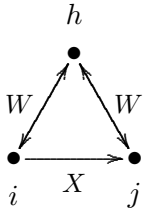
$s^{\text{net}}_{i20}(x) = x_{i+} \sum_h \left\{ \max_k \left(w_{ik}\, w_{hk} \right) \right\}$;
and for $p = 2$, there is a square root transformation:
$s^{\text{net}}_{i20}(x) = x_{i+} \sqrt{\sum_h \left\{ \max_k \left(w_{ik}\, w_{hk} \right) - w_{ih} \right\}}$
and
$s^{\text{net}}_{i20}(x) = x_{i+} \sqrt{\sum_h \left\{ \max_k \left(w_{ik}\, w_{hk} \right) \right\}}$,
respectively.

Further there are a number of mixed triadic effects.

21. *agreement along W leading to X*, (X: from W agreement) `(from)`
$s^{\text{net}}_{i21}(x) = \sum_{j,h} x_{ij}\, w_{ih}\, w_{jh}$;
this refers to agreement of actors with respect to their W-choices (structural equivalence with respect to outgoing W-choices); the contribution of the tie $i \overset{X}{\rightarrow} j$ is proportional to the number of joint W choices of others, $i \overset{W}{\rightarrow} h \overset{W}{\leftarrow} j$.
For internal effect parameter $p \geq 2$, the effect is
$s^{\text{net}}_{i21}(x) = \sum_j x_{ij} \sqrt{\sum_{h \neq i,j} w_{ih}\, w_{jh}}$.

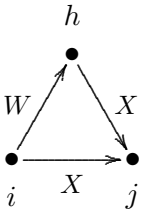


22. *agreement along mutual W-ties leading to X*, (X: from W mutual agreement) `(fromMutual)`
$s^{\text{net}}_{i22}(x) = \sum_{j,h} x_{ij}\, w_{ih}\, w_{hi}\, w_{jh}\, w_{hj}$;
this refers to agreement of actors with respect to their mutual W-choices (structural equivalence with respect to mutual W-choices); the contribution of the tie $i \overset{X}{\rightarrow} j$ is proportional to the number of joint mutual W choices of others, $i \overset{W}{\leftrightarrow} h \overset{W}{\leftrightarrow} j$.
For internal effect parameter $p \geq 2$, the effect is
$s^{\text{net}}_{i22}(x) = \sum_j x_{ij} \sqrt{\sum_{h \neq i,j} w_{ih}\, w_{hi}\, w_{jh}\, w_{hj}}$.



23. *any agreement along W leading to X*, (X: from W any agreement) `(fromAny)`
$s^{\text{net}}_{i23}(x) = \sum_{j,h} x_{ij}\, I\{w_{ih}\, w_{jh} > 0\}$;
this refers to agreement of actors with respect to their W-choices (structural equivalence with respect to outgoing W-choices), truncated to 1; the contribution of the tie $i \overset{X}{\rightarrow} j$ is proportional to whether there are any joint W choices of others, $i \overset{W}{\rightarrow} h \overset{W}{\leftarrow} j$.

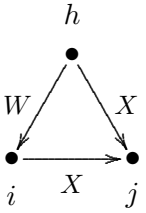
24. *W leading to agreement along X*, (X: W to agreement) `(to)`



$s^{\text{net}}_{i24}(x) = \sum_{j,h} x_{ij}\, w_{ih}\, x_{hj}$;
this refers to the closure of mixed $W-X$ two-paths; the contribution of the tie $i \overset{X}{\to} j$ is proportional to the number of mixed $W-X$ two-paths $i \overset{W}{\to} h \overset{X}{\to} j$.
This is an elementary effect for actor i: only the x_{ij} tie indicator in the formula, corresponding to the tie $i \overset{X}{\to} j$, is the dependent variable here.
The interpretation is that actors have the tendency to make the same outgoing X-choices as those to whom they have a W-tie.
For internal effect parameter $p = 2$, the effect is
$s^{\text{net}}_{i24}(x) = \sum_j x_{ij} \sqrt{\sum_{h\neq i,j} w_{ih}\, x_{hj}}$.
For internal effect parameter $p = 3$, the effect is
$s^{\text{net}}_{i24}(x) = \sum_j x_{ij}\, I\{\sum_{h\neq i,j} w_{ih}\, x_{hj} \geq 1\}$.

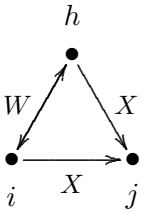
25. *W leading to any agreement along X*, (X: W to any agreement) `(toAny)`
$s^{\text{net}}_{i25}(x) = \sum_h w_{ih}\ \max_j\{x_{ij}\, x_{hj}\}$;
this is a truncated version of the `to` effect, somewhat similar to the `transTies` effect: the effect is the number of $i \overset{W}{\to} h$ ties accompanied by at least one $X-X$ out-two-star.
The interpretation is that with others to whom actors have a W-tie, they have the tendency to make at least one outgoing X-choice in common.

26. *W (backwards) leading to agreement along X*, (X: W (backwards) to agreement) `(toBack)` (earlier called `(MixedInWX)`)



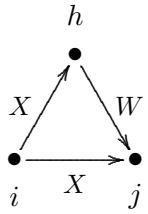
$s^{\text{net}}_{i26}(x) = \sum_{j,h} x_{ij}\, w_{hi}\, x_{hj}$;
this refers to the closure of mixed incoming W and X ties; the contribution of the tie $i \overset{X}{\to} j$ is proportional to the number of mixed $W-X$ out-two-stars $h \overset{W}{\to} i, h \overset{X}{\to} j$.
This is an elementary effect for actor i: only the x_{ij} tie indicator in the formula, corresponding to the tie $i \overset{X}{\to} j$, is the dependent variable here.
For internal effect parameter $p = 2$, the effect is
$s^{\text{net}}_{i26}(x) = \sum_j x_{ij} \sqrt{\sum_{h\neq i,j} w_{hi}\, x_{hj}}$.
For internal effect parameter $p = 3$, the effect is
$s^{\text{net}}_{i26}(x) = \sum_j x_{ij}\, I\{\sum_{h\neq i,j} w_{hi}\, x_{hj} \geq 1\}$.

27. *mutual W leading to agreement along X*, (X: mutual W to agreement) `(toRecip)`



$s^{\text{net}}_{i27}(x) = \sum_{j,h} x_{ij}\, w_{hi}\, w_{ih}\, x_{hj}$;
this refers to the closure of mixed mutual W and X ties; the contribution of the tie $i \overset{X}{\rightarrow} j$ is proportional to the number of mixed $W-X$ out-two-stars $h \overset{W}{\leftrightarrow} i, h \overset{X}{\rightarrow} j$.
This is an elementary effect for actor i: only the x_{ij} tie indicator in the formula, corresponding to the tie $i \overset{X}{\rightarrow} j$, is the dependent variable here.
For internal effect parameter $p = 2$, the effect is
$s^{\text{net}}_{i27}(x) = \sum_j x_{ij} \sqrt{\sum_{h\neq i,j} w_{hi}\, w_{ih}\, x_{hj}}$.
For internal effect parameter $p = 3$, the effect is
$s^{\text{net}}_{i27}(x) = \sum_j x_{ij}\, I\{\sum_{h\neq i,j} w_{hi}\, w_{ih}\, x_{hj} \geq 1\}$.

28. *XWX closure of* W, `(cl.XWX)`

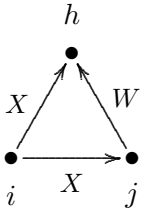


$s^{\text{net}}_{i28}(x) = \sum_{j,h} x_{ij}\, x_{ih}\, w_{hj}$;
this refers to the closure of mixed $X-W$ two-paths; the contribution of the tie $i \overset{X}{\rightarrow} j$ is proportional to the number of mixed $X-W$ two-paths $i \overset{X}{\rightarrow} h \overset{W}{\rightarrow} j$ plus the number of mixed $X-W$ two-in-stars $i \overset{X}{\rightarrow} h$, $j \overset{W}{\rightarrow} h$.
The interpretation is the closure of $X \rightarrow W$ paths: if there is a tie $h \overset{W}{\rightarrow} j$, then ties $i \overset{X}{\rightarrow} j$ and $i \overset{X}{\rightarrow} h$ will tend to entrain each other.
(The reported target statistic is multiplied by 2.)

There are two partial variants of this effect; they can be distinguished not by the Method of Moments, but only by Maximum Likelihood and Bayesian estimation.

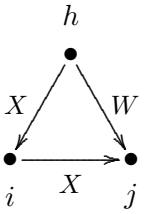
29. *XWX closure-1 of* W, `(cl.XWX1)`
This is an elementary effect, not an evaluation effect, comprising of the 'XWX closure of W' effect only the contribution of the the number of mixed $X-W$ two-paths $i \overset{X}{\rightarrow} h \overset{W}{\rightarrow} j$. So the dependent variable here is only the tie variable $i \rightarrow j$ in the figure above. The effect is defined as
$s^{\text{el}}_{i29}(x) = x_{ij} \sum_{h;h\neq j} x_{ih}\, w_{hj}$;

30. *XWX closure-2 of* W, `(cl.XWX2)`

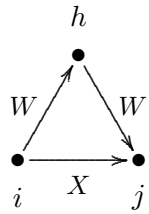


This is an elementary effect, not an evaluation effect, comprising of the 'XWX closure of W' effect only the contribution of the number of mixed $X-W$ two-in-stars $i \overset{X}{\rightarrow} h$, $j \overset{W}{\rightarrow} h$.
In other words, only the $i \rightarrow j$ tie in the figure here is the dependent variable. The effect is defined as
$s^{\text{el}}_{i30}(x) = x_{ij} \sum_{h;h\neq j} x_{ih}\, w_{jh}$.

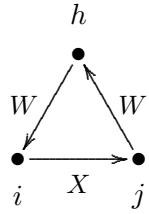
31. *shared mixed incoming X and W* , (X: mixed incoming with W) `(mixedInXW)`
$s_{i31}^{\text{net}}(x) = \sum_{j,h} x_{ij}\, x_{hi}\, w_{hj}$;
this refers to the closure of mixed incoming X and W ties; the contribution of the tie $i \overset{X}{\to} j$ is proportional to the number of mixed $X - W$ out-two-stars $h \overset{X}{\to} i, h \overset{W}{\to} j$.
For internal effect parameter $p = 2$, the effect is
$s_{i31}^{\text{net}}(x) = \sum_j x_{ij} \sqrt{\sum_{h \neq i,j} x_{hi}\, w_{hj}}$.
For internal effect parameter $p = 3$, the effect is
$s_{i31}^{\text{net}}(x) = \sum_j x_{ij}\, I\{\sum_{h \neq i,j} x_{hi}\, w_{hj} \geq 1\}$



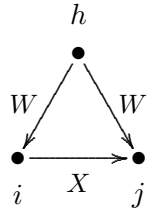
32. *mixed WW => X closure*, (X: closure of W) `(closure)`
$s_{i32}^{\text{net}}(x) = \sum_{j,h} x_{ij}\, w_{ih}\, w_{hj}$;
this refers to the closure of $W - W$ two-paths; the contribution of the tie $i \overset{X}{\to} j$ is proportional to the number of $W - W$ two-paths $i \overset{W}{\to} h \overset{W}{\to} j$.
The interpretation is that actors have the tendency to make and maintain X-ties to those to whom they have an indirect (distance 2) W-tie: 'W-ties of W-ties tend to become X-ties';



33. *mixed cyclic WW => X closure*, (X: cyclic closure of W) `(cyClosure)`
$s_{i33}^{\text{net}}(x) = \sum_{j,h} x_{ij}\, w_{jh}\, w_{hi}$;
this refers to the cyclic closure of $W - W$ two-paths; the contribution of the tie $i \overset{X}{\to} j$ is proportional to the number of $W - W$ two-paths $j \overset{W}{\to} h \overset{W}{\to} i$.
The interpretation is that actors have the tendency to make and maintain X-ties to those from whom they receive an indirect (distance 2) W-tie: 'W-ties of W-ties tend to become reciprocated by X-ties'.



34. *closure of shared incoming WW => X*,
(X: shared incoming W) `(sharedIn)`
$s_{i34}^{\text{net}}(x) = \sum_{j,h} x_{ij}\, w_{hi}\, w_{hj}$;
this refers to shared incoming W ties contributing to the tie $i \overset{X}{\to} j$.



The following two Jaccard similarity effects also are triadic, but not expressed as sums over triads.

35. *Jaccard similarity with respect to outgoing W ties effect* `(JoutMix)` , the Jaccard similarity with respect to outgoing W-ties,

$s^{\text{net}}_{i35}(x) = \sum_j x_{ij}\, J_{W,\,\text{out}}(i,j)$, where

$$J_{W,\,\text{out}}(i,j) \;=\; \frac{\sum_h w_{ih}\, w_{jh}}{w_{i+} + w_{j+} - \sum_h w_{ih}\, w_{jh}}$$

(where 0/0 is taken as 0).

36. *Jaccard similarity with respect to incoming W ties effect* (`JinMix`), defined by the Jaccard similarity with respect to incoming W-ties, $s^{\text{net}}_{i36}(x) = \sum_j x_{ij}\, J_{W,\,\text{in}}(i,j)$, where

$$J_{W,\,\text{in}}(i,j) \;=\; \frac{\sum_h w_{hi}\, w_{hj}}{w_{+i} + w_{+j} - \sum_h w_{hi}\, w_{hj}}$$

(where again 0/0 is taken as 0).

37. Five variations of mixed *GWESP (geometrically weighted edgewise shared partners)* effects: `gwespFFMix`, `gwespBBMix`, `gwespFBMix`, `gwespBFMix`, `gwespRRMix`, and for non-directed and two-mode networks the sixth version `gwespMix`.
See above for the GWESP effects for one-mode networks for their further background. These are geometrically downweighted analogues: `gwespFFMix` of `closure`, `gwespBBMix` of `cyClosure`, `gwespFBMix` of `from`, `gwespBFMix` of `sharedIn`, and `gwespRRMix` of `fromMutual`. The `gwespFFMix` effect is defined by

$$\text{GWESPFF}(i,\alpha) \;=\; \sum_{j=1}^{n} x_{ij}\, e^{\alpha} \left\{ 1 \;-\; \left(1 - e^{-\alpha}\right)^{\sum_{h=1}^{n} w_{ih} w_{hj}} \right\} ,$$

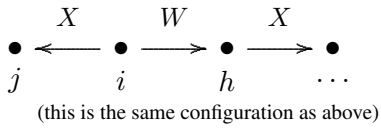
where the convention is used that $w_{jj} = 0$ for all j. See the figures above in the presentation of the `gwespFF` effect.

The parameter α is a tuning parameter that may range from 0 to ∞. The internal effect parameter is defined as $100 \times \alpha$. An often used value is $\alpha = \log(2) = 0.69$ (Snijders et al., 2006), corresponding to an internal effect parameter of 69, but it is worthwhile to try out different values of α to see which one gives the best fit. For large α the `gwespFFMix` effect will approximate the `closure` effect, and similarly for the other correspondences mentioned above.

The other types of mixed GWESP effect are analogous, with different tie orientations. They are defined as follows:
`gwespBBMix`: $\sum_{h=1}^{n} w_{ih} w_{hj}$ is replaced by $\sum_{h=1}^{n} w_{hi} w_{jh}$;
`gwespFBMix`: $\sum_{h=1}^{n} w_{ih} w_{hj}$ is replaced by $\sum_{h=1}^{n} w_{ih} w_{jh}$;
`gwespBFMix`: $\sum_{h=1}^{n} w_{ih} w_{hj}$ is replaced by $\sum_{h=1}^{n} w_{hi} w_{hj}$;
`gwespRRMix`: $\sum_{h=1}^{n} w_{ih} w_{hj}$ is replaced by $\sum_{h=1}^{n} w_{ih} w_{hi} w_{jh} w_{hj}$;
`gwespMix` for two-mode W: $\sum_{h=1}^{n} w_{ih} w_{hj}$ is replaced by $\sum_{h=1}^{m} w_{ih} w_{jh}$.

Then there are effects using mixed configurations on four nodes (cf. `sharedPop`).

38. mixed twopath average effect (X: mixed twopath$^{1/p}$ W average) `(outOutAvIntn)`, defined by the outdegree of i multiplied by the average of the X-outdegrees of those to whom i has a W-tie:
for $p = 1$, this is



(this is the same configuration as above)

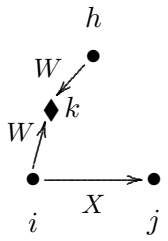
$$s^{\text{net}}_{i38}(x) = x_{i+} \frac{\sum_h w_{ih}\big(x_{h+} - \bar{x}\big)}{\sum_h w_{ih}}$$

and for $p = 2$, there is a square root transformation:

$$s^{\text{net}}_{i38}(x) = x_{i+} \frac{\sum_h w_{ih}\big(\sqrt{x_{h+}} - \sqrt{\bar{x}}\big)}{\sum_h w_{ih}}$$

where $\bar{x}$ is the average outdegree over all waves.

39. W twopath activity effect (X: W twopath$^{1/p}$ activity) `(dist2OutInActIntn)` defined by the outdegree of i multiplied by sum of W-indegrees of those k to whom i has a W-tie $i \rightarrow k$:
for $p = 1$, this is



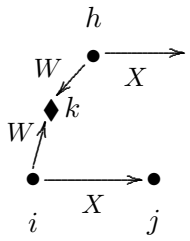
$$s^{\text{net}}_{i39}(x) = x_{i+} \sum_k w_{ik} \left(w_{+k} - \bar{w}\right)\big\}$$

and for $p = 2$, there is a square root transformation:

$$s^{\text{net}}_{i39}(x) = x_{i+} \sum_k w_{ik} \left(\sqrt{w_{+k}} - \sqrt{\bar{w}}\right)$$

where $\bar{w}$ is the average W-outdegree over all waves.

40. shared W outdegree activity effect (X: shared W outdegree$^{1/p}$ activity) `(outOutDist2ActIntn)` defined by the outdegree of i multiplied by sum of X-outdegrees of those h to whom i has a distance-2 W-tie $i \rightarrow k \leftarrow h$ (counting multiplicities).
This is an elementary effect. For $p = 1$, it is defined by



$$s^{\text{el}}_{ij40}(x) = x_{ij} \sum_h (x_{h+} - \bar{x})\big\{ \sum_k w_{ik}\, w_{hk}\big\}$$

and for $p = 2$, there is a square root transformation:

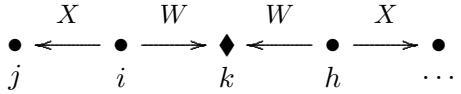
$$s^{\text{el}}_{ij40}(x) = x_{ij} \sum_h (x_{h+} - \bar{x}) \sum_h \sqrt{\sum_k w_{ik}\, w_{hk}}$$

where $\bar{x}$ is the average outdegree over all waves.

41. average two-mode W alter by total (X: av. 2M alter W by tot) `(avAlt.2M.tot)`, defined by the outdegree of i multiplied by the average of the X-outdegrees of those h to whom i has a distance-2 W-tie $i \rightarrow k \leftarrow h$ (counting multiplicities).
This is an elementary effect. For $p = 1$, it is defined by

for $p = 1$, this is

$$s^{\mathrm{el}}_{ij41}(x) = \frac{x_{ij} \sum_h (x_{h+} - \bar{x}) \{ \sum_k w_{ik}\, w_{hk} \}}{\sum_{k,h} w_{ik}\, w_{hk}}$$



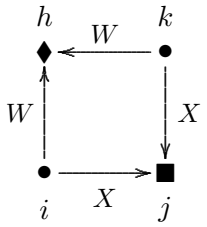
(this is the same configuration as above)

and for $p = 2$, there is a square root transformation:

$$s^{\mathrm{el}}_{ij41}(x) = \frac{x_{ij} \sum_h (x_{h+} - \bar{x}) \sum_h \sqrt{\sum_k w_{ik}\, w_{hk}}}{\sqrt{\sum_{k,h} w_{ik}\, w_{hk}}}$$

where $\bar{x}$ is the average outdegree over all waves, and $0/0$ is defined as 0.

42. *shared W leading to agreement along X*, (X: shared W to agreement) `(sharedTo)`.



This is an elementary effect. It is defined by

$s^{\mathrm{el}}_{ij42}(x) = x_{ij} \sum_{h,k;k\neq i} x_{kj} \left(w_{ih}\, w_{kh} - c \right)$;

this refers to the closure of mixed $W - X$ three-paths; the contribution of the tie $i \overset{X}{\rightarrow} j$ is proportional to the number of mixed $W - X$ three-paths $i \overset{W}{\rightarrow} h \overset{W}{\leftarrow} k \overset{X}{\rightarrow} j$.

This can be regarded also as a mixed 4-cycle effect.

The effect parameter p can take the values 1, 2, 3 and 4. The value c, a constant for centering, is the average of the observed values $w_{ih}\, w_{kh}$:

$$c = \begin{cases} \left(1/Mn(n-1)\right) \sum_{m,h} \left(w^2_{+h}(t_m) - w_{+h}(t_m) \right) & (p \geq 3) \\ 0 & (p < 3)\,. \end{cases}$$

Note that since this is the evaluation function for actor i with respect to network X, only the x_{ij} tie indicator in the formula, corresponding to the tie $i \overset{X}{\rightarrow} j$, is the dependent variable here.

The interpretation is that actors have the tendency to make the same outgoing X-choices as those (k) with whom they share many outgoing W-ties.

For $p = 2$ and $p = 4$, the square root is taken:

$s^{\mathrm{net}}_{i42}(x) = \sum_{j,k;k\neq i} x_{ij}\, x_{kj} \left(\sqrt{\sum_h w_{ih}\, w_{kh}} - \sqrt{c} \right)$.

This can be regarded as a higher-order effect related to the `inPopOutW` effect (see below) as well as indegree-popularity. The differences between the centered and non-centered versions amount to a multiple of the indegree-popularity (without $\sqrt{}$) `inPop` effect. Therefore, when the `sharedTo` effect is used, it may be advisable also to include the `inPop`, `outActIntn`, and `inPopOutW` effects, so as to avoid misinterpretations.

43. *average two-mode W alter by tie*, (X: av. 2M alter W by tie) `(avAlt.2M.tie)`.

This is an elementary effect. It is defined by

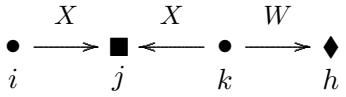
$$s^{\mathrm{el}}_{ij43}(x) = x_{ij} \frac{\sum_{h,k;k\neq i} x_{kj}\, (w_{ih}\, w_{kh})}{\sum_{h,k;k\neq i} (w_{ih}\, w_{kh})}\,;$$

this refers to the average closure of mixed $W - X$ three-paths, and is an alternative for `sharedTo`. This can be also regarded as a mixed 4-cycle effect.

For internal effect parameter $p = 2$, it is defined by

$$s^{\text{el}}_{ij43}(x) = x_{ij} \frac{\sum_{k;k\neq i} x_{kj} \sqrt{\sum_h (w_{ih}\, w_{kh})}}{\sum_{k;k\neq i} \sqrt{\sum_h (w_{ih}\, w_{kh})}}\,.$$

44. *indegree-popularity weighted by W-outdegree*, (X: ind. pop. by outd. W) (`inPopOutW`)



$s^{\text{net}}_{i44}(x) = \sum_{j,k;k\neq i} x_{ij}\, x_{kj} \left(w_{k+} - c\right)$;
this refers to mixed $X - X - W$ three-paths; the contribution of the tie $i \overset{X}{\to} j$ is proportional to the number of mixed $X - W$ three-paths $j \overset{X}{\leftarrow} k \overset{W}{\to} h$.
This can be regarded also as a mixed 3-path effect.

The effect parameter p can take the values 1, 2, 3 and 4. For $p = 1, 2$, there is no centering: $c = 0$. For $p = 3, 4$, the value c, a constant for centering, is the average of the W-outdegrees over all waves.

For $p = 2$ and $p = 4$, the square root is taken:
$s^{\text{net}}_{i44}(x) = \sum_{j,k;k\neq i} x_{ij}\, x_{kj} \left(\sqrt{w_{k+}} - \sqrt{c}\right)$.

Note that since this is the evaluation function for actor i with respect to network X, only the x_{ij} tie indicator in the formula, corresponding to the tie $i \overset{X}{\to} j$, is the dependent variable here.
The interpretation is that actors have the tendency to make more outgoing X-choices to those (j) who have many outgoing W-ties.

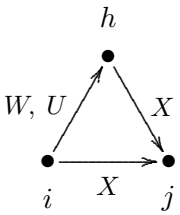
This can be regarded as a higher-order effect related to indegree-popularity. The differences between the centered and non-centered versions amount to a multiple of the indegree-popularity (without $\sqrt{}$) `inPop` effect. Therefore, when the `inPopOutW` effect is used, it is advisable also to include the `inPop` effect, so as to avoid misinterpretations.

13.1.3.1 Dyadic covariate effects

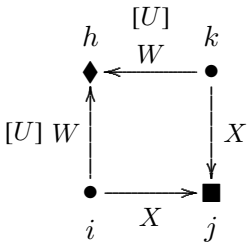
The effects for a dependent network X, with a co-evolving network W, and a dyadic covariate U_{ij} are the following.

For these effects the other network W is given as `covar1`, and the dyadic covariate U as `covar2`.

45. *W, weighted by U, leading to agreement along X,*
(X: W weighted by U to agreement) `(toU)`
$s^{\text{net}}_{i45}(x) = \sum_{j,h} x_{ij}\, w_{ih}\, u_{ih}\, x_{hj}$;
this refers to the closure of mixed $W - X$ two-paths, weighted by the dyadic covariate U; the contribution of the tie $i \overset{X}{\to} j$ is proportional to the sum of u_{ih} for all mixed $W - X$ two-paths $i \overset{W}{\to} h \overset{X}{\to} j$.
This is an elementary effect for actor i: only the x_{ij} tie indicator in the formula, corresponding to the tie $i \overset{X}{\to} j$, is the dependent variable here.
The interpretation is that actors have the tendency to make the same outgoing X-choices as those to whom they have a W-tie, weighted by U.
Here W is `covar1` and U is `covar2`.
This will be suitable mainly for non-centered covariates U, for which the term 'weight' is appropriate.
This effect was incorrect until and including version 1.6.11.



46. *shared W weighted by U leading to agreement along X,* (X: shared W wghtd by U $\Rightarrow$ agrmnt along X) `(sharedToU)`.
This is an elementary effect. It is defined by
$s^{\text{el}}_{i46}(x) = x_{ij} \sum_{k;k\neq i} x_{kj} \left\{ \sum_h \left(w_{ih}\, w_{kh} [u_{ih}]\, [u_{kh}] \right) \right\}$;
this refers to the closure of mixed $W - X$ three-paths, where the W ties are weighted by the dyadic covariate U; the contribution of the tie $i \overset{X}{\to} j$ is proportional to the weighted sum of mixed $W - X$ three-paths $i \overset{WU}{\longrightarrow} h \overset{WU}{\longleftarrow} k \overset{X}{\to} j$.
Usually, it will be good to work with a non-centered dyadic covariate.
This can be regarded also as a weighted mixed 4-cycle effect.



The effect parameter p can take the values 1 to 6. For parameters $p = 1, 3, 4, 6$, the tie $i \overset{WU}{\longrightarrow} h$ is weighted by u_{ih}; for parameters $p = 2, 3, 5, 6$, the tie $k \overset{WU}{\longrightarrow} h$ is weighted by u_{kh}; and for parameters $p = 4, 5, 6$, the square root is taken of the form between braces $\left\{ \cdots \right\}$.
This means that for $p = 3, 6$, both ties are weighted, and for $p = 1, 2, 4, 5$, only one of them.
Note that since this is the evaluation function for actor i with respect to network X, only the x_{ij} tie indicator in the formula, corresponding to the tie $i \overset{X}{\to} j$, is the dependent variable here.
The interpretation is that actors have the tendency to make the same outgoing X-choices as those (k) with whom they share many outgoing W-ties, weighted by U. There is no

centering as in `sharedTo`.

average two-mode W alter by tie, (X: av. 2M alter W by tie) `(avAlt.2M.tie)`.

47. *average two-mode W alter weighted by U, by tie*, (X: av. 2M alter W weight U by tie) `(avAltU.2M.tie)`.
    This is an elementary effect. It is defined for internal effect parameter $p = 1, 2, 3$ by
    $$s^{\mathrm{el}}_{ij47}(x) = x_{ij} \frac{\sum_{k;k\neq i} x_{kj} \sum_h \left(w_{ih}\, w_{kh}[u_{ih}]\,[u_{kh}]\right)}{\sum_{k;k\neq i} \sum_h \left(w_{ih}\, w_{kh}[u_{ih}]\,[u_{kh}]\right)};$$
    this refers to the average closure of mixed $W-X$ three-paths, where the W ties are weighted by the dyadic covariate U; it is an alternative to `(sharedToU)`. This can be also regarded as a weighted mixed 4-cycle effect.
    The weighting depends on the internal effect parameter p just like it is for `(sharedToU)`. This can be also regarded as a weighted mixed 4-cycle effect.

    For internal effect parameter $p = 4, 5, 6$, it is defined by
    $$s^{\mathrm{el}}_{ij47}(x) = x_{ij} \frac{\sum_{k;k\neq i} x_{kj} \sqrt{\sum_h \left(w_{ih}\, w_{kh}[u_{ih}]\,[u_{kh}]\right)}}{\sum_{k;k\neq i} \sqrt{\sum_h \left(w_{ih}\, w_{kh}[u_{ih}]\,[u_{kh}]\right)}}.$$

13.1.3.2 Monadic covariate effects

The mixed degree-related effects can be weighted with monadic covariates V, as implemented by the following four effects. (For two-mode networks, these effects are meaningful only in special cases; however, this is not checked when creating the effects object, so errors can lead here to meaningless results or to crashes.) Note that, unlike the unweighted effects `inPopIntn`, `inActIntn`, `outPopIntn`, `outActIntn`, the W-degrees here are not centered (well, they are also not really degrees).

Weights at the other side of the W-tie can be obtained by interactions of `inPopIntn`, `inActIntn`, `outPopIntn`, `outActIntn` with the ego or alter effects of V. Different types of weighting are available through the distance-two effects `altDist2W` and `totDist2W`, see below.

For these effects the other network W is given as `covar1`, and the covariate V as `covar2`.

These weighted effects are defined by sums over all outgoing or incoming W-ties. To obtain averages, they can be interacted by the appropriate one of `divOutEgoIntn`, `divInEgoIntn`, `divOutAltIntn`, or `divInAltIntn`. The internal effect parameters should match; see Section 6.12.1.

48. *Effect of in-degree in W, weighted by sender's V, on X-popularity* (X: indegree$^{1/p}$ W weighted by V popularity) `(inPopIntnX)`
    defined by the sum of values of V for actors h for which there are ties $i \overset{X}{\to} j \overset{W}{\leftarrow} h$, i.e., at mixed $X - W$ out-in distance 2; the value $\sum_h w_{hj}\, v_h$ can be regarded as a V-weighted

version of the W-indegree of j:
$s^{\text{net}}_{i48}(x) = \sum_j x_{ij} \sum_h w_{hj}\, v_h$
or for $p = 2$
$s^{\text{net}}_{i48}(x) = \sum_j x_{ij} \sqrt{\sum_h w_{hj}\, v_h}$;
note that to use this effect for $p = 2$ the variable V must be nonnegative, which implies that it must be non-centered;

49. *Effect of in-degree in W, weighted by sender's V, on X-activity* (X: indegree$^{1/p}$ W weighted by V activity) `(inActIntnX)`
defined by the sum of values of V for actors h for which there are ties $h \overset{W}{\to} i \overset{X}{\to} j$, i.e., a mixed $W - X$ two-path with i in the middle; the value $\sum_h w_{hi}\, v_h$ can be regarded as a V-weighted version of the W-indegree of i:
$s^{\text{net}}_{i49}(x) = \sum_j x_{ij} \sum_h w_{hi}\, v_h$
or for $p = 2$
$s^{\text{net}}_{i49}(x) = \sum_j x_{ij} \sqrt{\sum_h w_{hi}\, v_h}$;
note that to use this effect for $p = 2$ the variable V must be nonnegative, which implies that it must be non-centered;

50. *Effect of out-degree in W, weighted by receiver's V, on X-popularity* (X: outdegree$^{1/p}$ W weighted by V popularity) `(outPopIntnX)`
defined by the sum of values of V for actors h for which there are ties $i \overset{X}{\to} j \overset{W}{\to} h$, i.e., at mixed $X - W$ outgoing distance 2; the value $\sum_h w_{jh}\, v_h$ can be regarded as a V-weighted version of the W-outdegree of j:
$s^{\text{net}}_{i50}(x) = \sum_j x_{ij} \sum_h w_{jh}\, v_h$
or for $p = 2$
$s^{\text{net}}_{i50}(x) = \sum_j x_{ij} \sqrt{\sum_h w_{jh}\, v_h}$;
note that to use this effect for $p = 2$ the variable V must be nonnegative, which implies that it must be non-centered;

51. *Effect of out-degree in W, weighted by receiver's V, on X-activity* (X: outdegree$^{1/p}$ W weighted by V activity) `(outActIntnX)`
defined by the sum of values of V for actors h for which there are ties $h \overset{W}{\leftarrow} i \overset{X}{\to} j$, i.e., a mixed $W - X$ two-star; the value $\sum_h w_{ih}\, v_h$ can be regarded as a V-weighted version of the W-outdegree of i:
$s^{\text{net}}_{i51}(x) = \sum_j x_{ij} \sum_h w_{ih}\, v_h$
or for $p = 2$
$s^{\text{net}}_{i51}(x) = \sum_j x_{ij} \sqrt{\sum_h w_{ih}\, v_h}$;
note that to use this effect for $p = 2$ the variable V must be nonnegative, which implies that it must be non-centered.

Similarly, there are some mixed degree effects restricting the summation to third actors h with the same covariate value as i. Here again, the other network W is given as `covar1`, and the covariate V as `covar2`.

52. *Effect of in-degree in W, restricted to same V, on X-popularity* (X: indegree$^{1/p}$ W from same V - popularity) `(sameXinPopIntn)`

defined by the number of incoming W ties to alter from other actors h having the same value of V as ego:
$s^{\text{net}}_{i52}(x) = \sum_j x_{ij} \sum_h w_{hj}\, I\{v_i = v_h\}$
or for $p = 2$
$s^{\text{net}}_{i52}(x) = \sum_j x_{ij} \sqrt{\sum_h w_{hj}\, I\{v_i = v_h\}}$;
for internal effect parameter $p = 3$, available only for covariates with integer values from 0 to 20, it is defined by
$$s^{\text{net}}_{i52}(x) = \sum_j x_{ij} \frac{\sum_h w_{hj}\, I\{v_i = v_h\}}{\sum_h I\{v_i = v_h\}};$$
for internal effect parameter $p = 4$, the square root of the ratio of sums is taken;

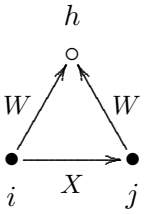
53. *Effect of in-degree in W, restricted to same V, on X-activity* (X: indegree$^{1/p}$ W from same V - activity) `(sameXinActIntn)`
defined by the number of incoming W ties to ego from other actors h having the same value of V as ego:
$s^{\text{net}}_{i53}(x) = \sum_j x_{ij} \sum_h w_{hi}\, I\{v_i = v_h\}$
or for $p = 2$
$s^{\text{net}}_{i53}(x) = \sum_j x_{ij} \sqrt{\sum_h w_{hi}\, I\{v_i = v_h\}}$;
for internal effect parameter $p = 3$, available only for covariates with integer values from 0 to 20, it is defined by
$$s^{\text{net}}_{i53}(x) = \sum_j x_{ij} \frac{\sum_h w_{hi}\, I\{v_i = v_h\}}{\sum_h I\{v_i = v_h\}};$$
for internal effect parameter $p = 4$, the square root of the ratio of sums is taken;

54. *Effect of out-degree in W, restricted to same V, on X-popularity* (X: outdegree$^{1/p}$ W to same V - popularity) `(sameXoutPopIntn)`
defined by the number of outgoing W ties from alter to other actors h having the same value of V as ego:
$s^{\text{net}}_{i54}(x) = \sum_j x_{ij} \sum_h w_{jh}\, I\{v_i = v_h\}$
or for $p = 2$
$s^{\text{net}}_{i54}(x) = \sum_j x_{ij} \sqrt{\sum_h w_{jh}\, I\{v_i = v_h\}}$;

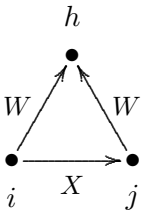
55. *Effect of out-degree in W, restricted to same V, on X-activity* (X: outdegree$^{1/p}$ W to same V - activity) `(sameXoutActIntn)`
defined by the number of outgoing W ties from ego to other actors h having the same value of V as ego:
$s^{\text{net}}_{i55}(x) = \sum_j x_{ij} \sum_h w_{ih}\, I\{v_i = v_h\}$
or for $p = 2$
$s^{\text{net}}_{i55}(x) = \sum_j x_{ij} \sqrt{\sum_h w_{ih}\, I\{v_i = v_h\}}$.

Then there are some mixed triadic effects restricted to triples with the same or different values on a monadic covariate V.

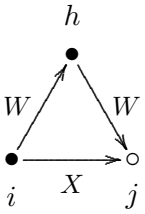
56. *agreement along W leading to X, for same V*,
(X: from W agr. $\times$ same V) `(covNetNet)`,
$s^{\text{net}}_{i56}(x) = \sum_{j,h} x_{ij}\, w_{ih}\, w_{jh}\, I\{v_i = v_j\}$;
(specified with `covar1` = W, `covar2` = V)
this refers to agreement of actors with respect to their W-choices (structural equivalence with respect to outgoing W-choices), but only for actors sharing the same value of a covariate V; the contribution of the tie $i \overset{X}{\rightarrow} j$ is proportional to the number of joint W choices of others, $i \overset{W}{\rightarrow} h \overset{W}{\leftarrow} j$, counting only those for which $v_i = v_j$.



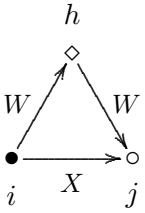
57. *agreement along W leading to X, for homogeneous V*,
(X: from W agr. $\times$ hom. V) `(homCovNetNet)`,
$s^{\text{net}}_{i57}(x) = \sum_{j,h} x_{ij}\, w_{ih}\, w_{jh}\, I\{v_i = v_j = v_h\}$;
(specified with `covar1` = W, `covar2` = V)
this refers to agreement of actors with respect to their W-choices (structural equivalence with respect to outgoing W-choices), but only for actors and choices sharing the same value of a covariate V; the contribution of the tie $i \overset{X}{\rightarrow} j$ is proportional to the number of joint W choices of others, $i \overset{W}{\rightarrow} h \overset{W}{\leftarrow} j$, counting only those for which $v_i = v_j = v_h$.



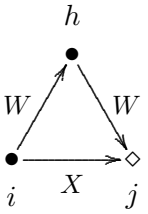
58. *mixed WW $\Rightarrow$ X closure, same-V path* (X: mixed WW closure same V), `(sameWWClosure)`
(specified with `covar1` = W, `covar2` = V)
$s^{\text{net}}_{i58}(x) = \sum_{j,h} x_{ij}\, w_{ih}\, w_{hj}\, I\{v_i = v_h\}$;
this refers to the closure by X of $W - W$ two-paths, restricted to two-paths $i \overset{W}{\rightarrow} h \overset{W}{\rightarrow} j$ in which the focal actor and the mediating actor have the same value of V.



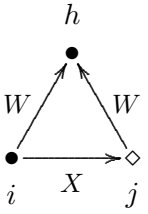
59. *mixed WW $\Rightarrow$ X closure, different-V path* (X: mixed WW closure diff. V), `(diffWWClosure)`
(specified with `covar1` = W, `covar2` = V)
$s^{\text{net}}_{i59}(x) = \sum_{j,h} x_{ij}\, w_{ih}\, w_{hj}\, I\{v_i \neq v_h\}$;
this refers to the closure by X of $W - W$ two-paths, restricted to two-paths $i \overset{W}{\rightarrow} h \overset{W}{\rightarrow} j$ in which the focal actor and the mediating actor have a different value of V.



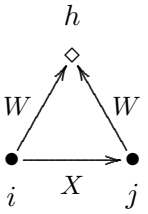
60. *mixed WW ⇒ X closure, same-V path jumping to different V* (X: W closure jumping V), `(jumpWWClosure)`
(specified with `covar1` = W, `covar2` = V)
$s^{\text{net}}_{i60}(x) = \sum_{j,h} x_{ij}\, w_{ih}\, w_{hj}\, I\{v_i = v_h \neq v_j\}$;
this refers to the closure of $W - W$ paths, restricted to 'jump outside of V-groups' in the sense that the focal actor and the mediating actor have the same value of V, but the target actor has a different value.



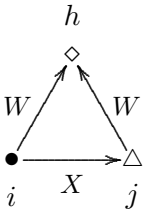
61. *agreement along W, same-V, leading to X for different V* (X: from W agreement jumping V), `(jumpFrom)`
(specified with `covar1` = W, `covar2` = V)
$s^{\text{net}}_{i61}(x) = \sum_{j,h} x_{ij}\, w_{ih}\, w_{jh}\, I\{v_i = v_h \neq v_j\}$;
this refers to agreement about W ties, restricted to 'jump outside of V-groups' in the sense that the focal actor and the agreed actor have the same value of V, but the target actor has a different value.



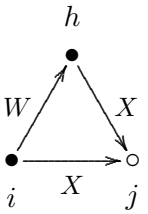
62. *agreement along W, same-V about contrasting V, leading to X* (X: from W agreement contrasting V), `(contrastCovNetNet)`
(specified with `covar1` = W, `covar2` = V)
$s^{\text{net}}_{i62}(x) = \sum_{j,h} x_{ij}\, w_{ih}\, w_{jh}\, I\{v_i = v_j \neq v_h\}$;
this refers to agreement 'outside of V-groups' about W ties, in the sense that the focal actor and the target actor have the same value of V, but the agreed actor has a different value;



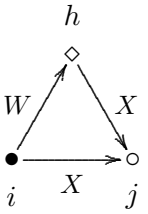
63. *agreement along W, all different V, leading to X* (X: from W agreement all diff. V), `(allDifCovNetNet)`
(specified with `covar1` = W, `covar2` = V)
$s^{\text{net}}_{i63}(x) = \sum_{j,h} x_{ij}\, w_{ih}\, w_{jh}\, I\{v_i, v_j, v_h \text{ all different}\}$;
this refers to agreement about W ties between actors who have all three a different value for V;



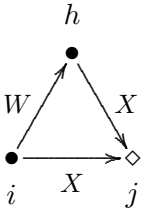
64. *mixed WX ⇒ X closure, same-V path* (X: mixed WX closure same V), `(sameWXClosure)`
(specified with `covar1` = W, `covar2` = V)
$s^{\text{net}}_{i64}(x) = \sum_{j,h} x_{ij}\, w_{ih}\, x_{hj}\, I\{v_i = v_h\}$;
this refers to the closure of $W - X$ two-paths, restricted to two-paths $i \overset{W}{\rightarrow} h \overset{X}{\rightarrow} j$ in which the focal actor and the mediating actor have the same value of V.
A variant of this effect with weighting by a dyadic covariate is the `toU` effect.



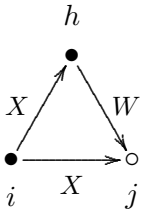
65. *mixed WX ⇒ X closure, different-V path* (X: mixed WX closure diff. V), `(diffWXClosure)`
(specified with `covar1` = W, `covar2` = V)
$s^{\text{net}}_{i65}(x) = \sum_{j,h} x_{ij}\, w_{ih}\, x_{hj}\, I\{v_i \neq v_h\}$;
this refers to the closure of $W - X$ two-paths, restricted to two-paths $i \overset{W}{\rightarrow} h \overset{X}{\rightarrow} j$ in which the focal actor and the mediating actor have a different value of V.



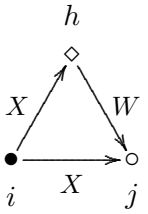
66. *mixed WX ⇒ X closure, same-V path jumping to different V* (X: mixed WX closure jumping V), `(jumpWXClosure)`
(specified with `covar1` = W, `covar2` = V)
$s^{\text{net}}_{i66}(x) = \sum_{j,h} x_{ij}\, w_{ih}\, x_{hj}\, I\{v_i = v_h \neq v_j\}$;
this refers to the closure of $W - X$ paths, restricted to 'jump outside of V-groups' in the sense that the focal actor and the mediating actor have the same value of V, but the target actor has a different value.



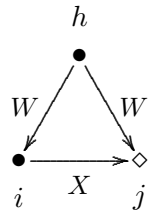
67. *mixed XW ⇒ X closure, same-V path* (X: mixed XW closure same V), `(sameXWClosure)`
(specified with `covar1` = W, `covar2` = V)
$s^{\text{net}}_{i67}(x) = \sum_{j,h} x_{ij}\, x_{ih}\, w_{hj}\, I\{v_i = v_h\}$;
this refers to the closure of $X - W$ two-paths, restricted to two-paths $i \overset{X}{\rightarrow} h \overset{W}{\rightarrow} j$ in which the focal actor and the mediating actor have the same value of V.



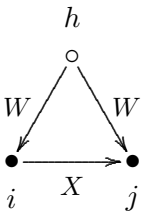
68. *mixed XW ⇒ X closure, different-V path* (X: mixed XW closure diff. V), `(diffXWClosure)`
(specified with `covar1` = W, `covar2` = V)
$s^{\text{net}}_{i68}(x) = \sum_{j,h} x_{ij}\, x_{ih}\, w_{hj}\, I\{v_i \neq v_h\}$;
this refers to the closure of $X - W$ two-paths, restricted to two-paths $i \overset{X}{\rightarrow} h \overset{W}{\rightarrow} j$ in which the focal actor and the mediating actor have a different value of V.



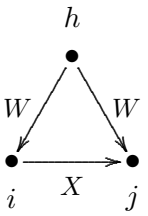
69. *closure of shared incoming WW => X same-V, leading to X for different V*,
(X: shared incoming W jumping V) `(jumpSharedIn)`
$s^{\text{net}}_{i69}(x) = \sum_{j,h} x_{ij}\, w_{hi}\, w_{hj}\, I\{v_i = v_h \neq v_j\}$;
(specified with `covar1` = W, `covar2` = V)
this refers to shared incoming W ties contributing to the tie $i \overset{X}{\rightarrow} j$, restricted to 'jump outside of V-groups' in the sense that the focal actor and the count of shared incoming actors is for those having the same value of V, but the target actor has a different value.



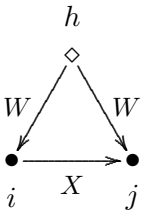
70. *shared incoming W leading to X, for same V*,
(X: shared incoming W same V) `(covNetNetIn)`,
$s^{\text{net}}_{i70}(x) = \sum_{j,h} x_{ij}\, w_{hi}\, w_{hj}\, I\{v_i = v_j\}$;
(specified with `covar1` = W, `covar2` = V)
this refers to shared incoming W-choices (structural equivalence with respect to incoming W-choices), but only for actors sharing the same value of a covariate V; the contribution of the tie $i \overset{X}{\rightarrow} j$ is proportional to the number of joint W choices of others, $i \overset{W}{\leftarrow} h \overset{W}{\rightarrow} j$, counting only those for which $v_i = v_j$.



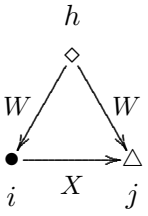
71. *shared incoming W leading to X, for homogeneous V*,
(X: shared incoming W hom. V) `(homCovNetNetIn)`,
$s^{\text{net}}_{i71}(x) = \sum_{j,h} x_{ij}\, w_{hi}\, w_{hj}\, I\{v_i = v_j = v_h\}$;
(specified with `covar1` = W, `covar2` = V)
this refers to shared incoming W-choices (structural equivalence with respect to incoming W-choices), but only for actors and choices sharing the same value of a covariate V; the contribution of the tie $i \overset{X}{\rightarrow} j$ is proportional to the number of joint incoming W choices of others, $i \overset{W}{\leftarrow} h \overset{W}{\rightarrow} j$, counting only those for which $v_i = v_j = v_h$.



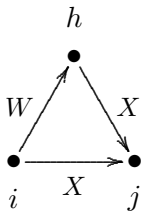
72. *shared incoming W, same-V about contrasting V, leading to X*
(X: shared incoming W contrasting V), `(contrastCovNetNetIn)`
(specified with `covar1` = W, `covar2` = V)
$s^{\text{net}}_{i72}(x) = \sum_{j,h} x_{ij}\, w_{hi}\, w_{hj}\, I\{v_i = v_j \neq v_h\}$;
this refers to shared incoming W-choices (structural equivalence with respect to incoming W-choices), restricted in the sense that the focal actor and the target actor have the same value of V, but the actors making the shared incoming choices have a different value;



73. *shared incoming W, all different V, leading to X*
(X: shared incoming W all different V), `(allDifCovNetNetIn)`
(specified with `covar1` = W, `covar2` = V)
$s^{\text{net}}_{i73}(x) = \sum_{j,h} x_{ij}\, w_{hi}\, w_{hj}\, I\{v_i, v_j, v_h \text{ all different}\}$;
this refers to shared incoming W-choices (structural equivalence with respect to incoming W-choices), restricted in the sense that the focal actor, the target actor, and the actors making the shared incoming choices all have a different value of V;



74. *mixed WX ⇒ X closure, homogeneous on V*
(X: mixed W closure homog. V), `(homWXClosure)`
(specified with `covar1` = W, `covar2` = V)
$s^{\text{net}}_{i74}(x) = \sum_{j,h} x_{ij}\, w_{ih}\, x_{hj}\, I\{v_i = v_j = v_h\}$;
this refers to the closure of $W - X$ paths, restricted to triples that are homogeneous with respect to V.



There are several effects similar to the 'alter' and 'similarity' effects described above, now depending on the auxiliary variable $\breve{v}_i$, 'alters' v-average'. Here the 'alter' is defined, however, by the other network W. Thus, *W-alters' v-average* $\breve{v}^W_i$ is defined by

$$\breve{v}^W_i = \begin{cases} \dfrac{\sum_j w_{ij} v_j}{w_{i+}} & \text{if } w_{i+} > 0 \\ \bar{v} & \text{if } w_{i+} = 0, \end{cases} \tag{25}$$

where $\bar{v}$ is the overall mean of variable V.

75. *covariate - alter at W-distance 2* `(altDist2W)`
This effect is associated with an effect parameter which can have values 1 or 2. For parameter 1, it is defined as the sum of W-alters' covariate-average over all actors to whom i has a tie,

$$s^{\text{net}}_{i75}(x) = \sum_j x_{ij} \breve{v}^W_j \qquad \text{(parameter 1)}.$$

For parameter 2, it is defined similarly, but for an alters' covariate-average excluding ego:

$$s^{\text{net}}_{i75}(x) = \sum_j x_{ij} \breve{v}^{W(-i)}_j \qquad \text{(parameter 2)}$$

where

$$\breve{v}^{W(-i)}_j = \begin{cases} \dfrac{\sum_{h \neq j} w_{jh} v_h}{w_{j+} - w_{ji}} & \text{if } w_{j+} - w_{ji} > 0 \\ \bar{v} & \text{if } w_{j+} - w_{ji} = 0. \end{cases} \tag{26}$$

76. *total covariate - alter at W-distance 2* `(totDist2W)`
This effect is like the previous effect, but now defined defined as the sum of W-alters' covariate-total over all actors to whom i has a tie. For parameter 1 it is

$$s^{\text{net}}_{i76}(x) = \sum_j x_{ij}\, x_{j+}\, \breve{v}^W_j \qquad \text{(parameter 1)}.$$

For parameter 2, it is defined similarly, but for an alters' covariate-total excluding ego:

$$s^{\text{net}}_{i76}(x) = \sum_j x_{ij}\, (x_{j+} - x_{ji})\, \breve{v}^{W(-i)}_j \qquad \text{(parameter 2)}$$

where $\breve{v}_j$ and $\breve{v}^{W(-i)}_j$ are as above.

77. *ego-(alter-W-distance-2) covariate - similarity* `(simEgoDist2W)`, defined as the sum of centered similarity between i and alters' covariate-average, for all actors j to whom i has a tie,

$$s_{i77}^{\text{net}}(x) = \sum_j x_{ij} \left(\text{sim}(\grave{v}^W)_{ij} - \widehat{\text{sim}^v} \right) ,$$

where the similarity scores $\text{sim}(\grave{v}^W)_{ij}$ are defined as

$$\text{sim}(\grave{v}^W)_{ij} = \frac{\Delta - |v_i^W - (\breve{v}_j^W)^{(-i)}|}{\Delta} ,$$

while $\Delta = \max_{ij} |v_i - v_j|$ is the observed range of the *original* covariate v, $(\breve{v}_j^W)^{(-i)}$ is as above in effect `(altDist2W)`,

and $\widehat{\text{sim}^v}$ is the mean of all similarity scores as used also for the `simX` effect; this centering is applied since version 1.1-285. For a constant covariate `mycov`, this mean is given by `attr(mydata$cCovars$mycov, "simMean")`.

78. *covariate - similarity at W-distance 2* `(simDist2W)`, defined as the sum of centered similarity values for alters' covariate-average between i and all actors j to whom i has a tie,

$$s_{i78}^{\text{net}}(x) = \sum_j x_{ij} \left(\text{sim}(\breve{v}^W)_{ij} - \widehat{\text{sim}^v} \right) ,$$

where the similarity scores $\text{sim}(\breve{v}^W)_{ij}$ are defined as

$$\text{sim}(\breve{v}^W)_{ij} = \frac{\Delta - |\breve{v}_i^W - \breve{v}_j^W|}{\Delta} ,$$

while $\Delta = \max_{ij} |v_i - v_j|$ is the observed range of the *original* covariate v, and $\widehat{\text{sim}^v}$ is as above.

79. *covariate - in-alter at W-distance 2* `(altInDist2W)`
This effect is associated with an effect parameter which can have values 1 or 2. For parameter 1, it is defined as the sum of W-in-alters' covariate-average over all actors to whom i has a tie,

$$s_{i79}^{\text{net}}(x) = \sum_j x_{ij} \, \breve{v}_j^W \qquad \text{(parameter 1)},$$

where

$$\breve{v}_j^W = \begin{cases} \dfrac{\sum_{h \neq i} w_{hj} \, v_h}{x_{+j} - x_{ij}} & \text{if } x_{+j} > 0 \\ \bar{v} & \text{if } x_{+j} = 0, \end{cases} \tag{27}$$

where $\bar{v}$ is (again) the global mean of V.

For parameter 2, it is defined similarly, but for an alters' covariate-average excluding ego:

$$s_{i79}^{\text{net}}(x) = \sum_j x_{ij}\, \check{v}_j^{W(-i)} \qquad \text{(parameter 2)}$$

where

$$\check{v}_j^{W(-i)} = \begin{cases} \dfrac{\sum_{h\neq j} w_{hj}\, v_h}{w_{j+} - w_{ji}} & \text{if } w_{+j} - w_{ij} > 0 \\ \bar{v} & \text{if } w_{+j} - w_{ij} = 0. \end{cases} \tag{28}$$

80. *total covariate - in-alter at W-distance 2* (`totInDist2W`)
This effect is like the previous effect, but now defined defined as the sum of W-in-alters' covariate-total over all actors to whom i has a tie. For parameter 1 it is

$$s_{i80}^{\text{net}}(x) = \sum_j x_{ij}\, x_{j+}\, \check{v}_j^{W} \qquad \text{(parameter 1)}.$$

For parameter 2, it is defined similarly, but for an alters' covariate-total excluding ego:

$$s_{i80}^{\text{net}}(x) = \sum_j x_{ij}\, (x_{j+} - x_{ji})\, \check{v}_j^{W(-i)} \qquad \text{(parameter 2)}$$

where $\check{v}_j$ and $\check{v}_j^{W(-i)}$ are as above.

81. *ego-(in-alter-W-distance-2) covariate - similarity* (`simEgoInDist2W`), defined as the sum of centered similarity between i and alters' covariate-in-average, for all actors j to whom i has a tie,

$$s_{i81}^{\text{net}}(x) = \sum_j x_{ij} \left(\text{sim}(\acute{v}^W)_{ij} - \widehat{\text{sim}^v} \right),$$

where the similarity scores $\text{sim}(\acute{v}^W)_{ij}$ are defined as

$$\text{sim}(\acute{v})_{ij} = \frac{\Delta - |v_i - \check{v}_j^{(-i)}|}{\Delta},$$

while $\check{v}_j^{(-i)}$ is as in the definition of `altInDist2W`, $\Delta = \max_{ij} |v_i - v_j|$ is the observed range of the *original* covariate v, and
$\widehat{\text{sim}^v}$ is as above.

13.1.3.3 Effects for three networks

The following is an effect for three networks: the dependent network is X, the two explanatory networks are W and Z.

82. *from W agreement weighted by Z indegrees* (`from.w.ind`), defined as the sum of Z-indegrees of all actors who have incoming W-ties from i and from those actors j to whom i has a tie,

$$s_{i82}^{\text{net}}(x) = \sum_j x_{ij} \left(\sum_h w_{ih}\, w_{jh}\, z_{+h} \right) .$$

For internal effect parameter $p = 2$, the effect is

$$s_{i82}^{\text{net}}(x) = \sum_j x_{ij} \left(\sum_h w_{ih}\, w_{jh}\, \sqrt{z_{+h}} \right) .$$

For internal effect parameter $p = -1$, the effect is

$$s_{i82}^{\text{net}}(x) = \sum_j x_{ij} \left(\sum_h \frac{w_{ih}\, w_{jh}}{(z_{+h} + 1)} \right) .$$

W is given as `covar1`, Z as `covar2`.

This effect also is available if all or some of X, W, and Z are the same. If X and W are the same, this effect is defined as an elementary effect (see Section 6.1.1).

13.1.4 Network creation and endowment functions

The *network creation function* is one way of modeling effects which operate in different strengths for the creation and the dissolution of relations. The network creation function is zero for dissolution of ties, and is given by

$$c_i^{\text{net}}(x) \;=\; \sum_k \zeta_k^{\text{net}}\, s_{ik}^{\text{net}}(x) \tag{29}$$

for creation of ties. In this formula, the ζ_k^{net} are the parameters for the creation function. The potential effects $s_{ik}^{\text{net}}(x)$ in this function, and their formulae, are the same as in the evaluation function; except that not all are available, as indicated in the preceding subsection. For further explication, consult Snijders (2001, 2005); (here, the 'gratification function' is used rather than the creation function), Snijders et al. (2007), and Steglich et al. (2010) (here only the endowment function is treated and not the creation function, but they are similar in an opposite way).

The *network endowment function* is another way of modeling effects which operate in different strengths for the creation and the dissolution of relations. The network endowment function is zero for creation of ties, and is given by

$$e^{\text{net}}(x) \;=\; \sum_k \gamma_k^{\text{net}}\, s_{ik}^{\text{net}}(x) \tag{30}$$

for dissolution of ties. In this formula, the γ_k^{net} are the parameters for the endowment function. The potential effects $s_{ik}^{\text{net}}(x)$ in this function, and their formulae, are the same as in the evaluation function; except that not all are available, as indicated in the preceding subsection. For further

explication, consult Snijders (2001, 2005); (here, the 'gratification function' is used rather than the endowment function), Snijders et al. (2007), and Steglich et al. (2010).

A better term than endowment is perhaps *maintenance*.

These functions are combined in the following way. For the creation of ties, the objective function used is

$$f_i^{\text{net}}(x) \; + \; c_i^{\text{net}}(x) \; , \tag{31}$$

in other words, the parameters for the evaluation and creation effects are added. For the dissolution of ties, on the other hand, the objective function is

$$f_i^{\text{net}}(x) \; + \; e_i^{\text{net}}(x) \; , \tag{32}$$

in other words, the parameters for the evaluation and endowment effects are added. Therefore, a model with a parameter with some value β_k for a given evaluation effect, and for which there are no separate creation and endowment effects, has exactly the same consequences as a model for which this evaluation effect is excluded, and that includes a creation as well as an endowment effect, both with the same parameter value $\zeta_k = \beta_k$ and $\gamma_k = \beta_k$.

Of the three types of effect — evaluation, creation, and endowment —, one therefore should use one or two, not all three, because this leads to collinearity.

13.1.5 Network rate function

To define rate effects by the function set_effect or set_effect, note that in the call of these functions the specification `type = "rate"` should be included.

The network rate function λ^{net} (lambda) is defined for Model Type 1 (which is the default Model Type) as a product

$$\lambda_i^{\text{net}}(\rho, \alpha, x, m) = \lambda_{i1}^{\text{net}} \lambda_{i2}^{\text{net}} \lambda_{i3}^{\text{net}}$$

of factors depending, respectively, on period m, actor covariates, and actor position (see Snijders, 2001, p. 383). The corresponding factors in the rate function are the following:

1. The dependence on the period `(Rate)` can be represented by a simple factor

   $$\lambda_{i1}^{\text{net}} = \rho_m^{\text{net}}$$

   for $m = 1, ..., M-1$.
   This basic rate parameter is always period-dependent (i.e., it may and will be different for different values of m); this is in contrast to all other parameters.

2. The effect of actor covariates `(RateX)` with values v_{hi} can be represented by the factor

   $$\lambda_{i2}^{\text{net}} = \exp(\sum_h \alpha_h \, v_{hi}) \, .$$

   If v_h is a behavior variable, the non-centered value is used.

3. The dependence on the position of the actor can be modeled as a function of the actor's out-degree `(outRate)`, in-degree `(inRate)`, and number of reciprocated relations `(recipRate)`, the 'reciprocated degrees'. Define these by

$$x_{i+} = \sum_j x_{ij}, \qquad x_{+i} = \sum_j x_{ji}, \qquad x_{i(r)} = \sum_j x_{ij}x_{ji}$$

(recalling that $x_{ii} = 0$ for all i).

The contribution of the out-degrees to $\lambda_{i3}^{\text{net}}$ is a factor

$$\exp(\alpha_h \, x_{i+}) \,,$$

if the associated parameter is denoted α_h for some h, and similarly for the contributions of the in-degrees and the reciprocated degrees.

Nonlinear dependence of the exponent on the degrees can also be specified:

- inverse outdegree effect `(outRateInv)`
  Denoting again the corresponding parameter by α_h (but always for different index numbers h), this effect multiplies the factor $\lambda_{i3}^{\text{net}}$ by
  $$\exp\big(\alpha_h/(x_{i+}+1)\big) \,.$$
- logarithmic outdegree effect `(outRateLog)`
  This effect multiplies the factor $\lambda_{i3}^{\text{net}}$ by
  $$\exp\big(\alpha_h \, \ln(x_{i+}+1)\big) \;=\; \big(x_{i+}+1\big)^{\alpha_h} \,.$$
- inverse indegree effect `(inRateInv)`
  This effect multiplies the factor $\lambda_{i3}^{\text{net}}$ by
  $$\exp\big(\alpha_h/(x_{+i}+1)\big) \,.$$
- logarithmic indegree effect `(inRateLog)`
  This effect multiplies the factor $\lambda_{i3}^{\text{net}}$ by
  $$\exp\big(\alpha_h \, \ln(x_{+i}+1)\big) \;=\; \big(x_{+i}+1\big)^{\alpha_h} \,.$$
- inverse reciprocal degree effect `(recipRateInv)`
  This effect multiplies the factor $\lambda_{i3}^{\text{net}}$ by
  $$\exp\big(\alpha_h/(x_{i(r)}+1)\big) \,.$$
- logarithmic reciprocal degree effect `(recipRateLog)`
  This effect multiplies the factor $\lambda_{i3}^{\text{net}}$ by
  $$\exp\big(\alpha_h \, \ln(x_{i(r)}+1)\big) \;=\; \big(x_{i(r)}+1\big)^{\alpha_h} \,.$$

  The exponential link function and logarithmic transformation collaborate to produce direct proportionality to (outdegree + 1) or (indegree + 1), respectively, in case the parameter is $\alpha_h = 1$.

For these effects, the addition of 1 to the degrees avoids problems (division by 0, logarithm of 0) that otherwise would occur when $x_{i+} = 0$ or $x_{+i} = 0$.

For making the rates dependent on the degrees, the logarithmic effects often give the best fit.

These effects work properly only for non-conditional estimation (set `cond = FALSE` *in* set_algorithm_saom*).*

When rate effects are included, sometimes it can be helpful to specify a smaller value for the initial gain parameter in set_algorithm_saom; e.g., `firstg = 0.01` or even `firstg = 0.001`. When the estimation algorithm diverges and any rate effects are in the model, this option should be considered. With a smaller `firstg` parameter convergence will be slower, but also there is a smaller probability that the parameter will diverge accidentally.

13.2 Behavioral evolution

The model of the dynamics of a dependent actor variable consists of a model of actors' decisions (according to *evaluation*, *creation*, and *endowment functions*) and a model of the timing of these decisions (according to a *rate function*), just like the model for the network dynamics. The decisions now do not concern the creation or dissolution of network ties, but whether an actor increases or decreases his score on the dependent actor variable by one, or keeps it as it is.

13.2.1 Behavioral evaluation function

The behavior evaluation function for actor i is defined as

$$f_i^{\mathrm{beh}}(x, z) = \sum_k \beta_k^{\mathrm{beh}} s_{ik}^{\mathrm{beh}}(x, z) \tag{33}$$

where β_k^{beh} are parameters and $s_{ik}^{\mathrm{beh}}(x, z)$ are effects as defined below. The behavioral dependent variable is denoted by z and the dependent network variable by x. Here the dependent variable is transformed to have an overall average value of 0; in other words, z denotes the original input variable minus the overall mean[19], which is given in the output file produced by write_report under the heading *Reading dependent actor variables*.

First there are effects that have to do only with the behavioral variable itself.

1. *behavioral shape effect* `(linear)`,
   $s_{i1}^{\mathrm{beh}}(x, z) = z_i$,
   where z_i denotes the value of the dependent behavior variable of actor i;

2. *quadratic shape effect, or effect of the behavior upon itself* `(quad)`, where the attractiveness of further steps up the behavior 'ladder' depends on where the actor is on the ladder:
   $s_{i2}^{\mathrm{beh}}(x, z) = z_i^2$;
   note that here z_i is the centered dependent behavioral variable; by default, this centering

[19] More precisely: this is the mean of the means per wave, and the means per wave are means of the non-missing observations for that wave.

is done by the grand mean in the data; for using the contemporaneous mean, work with the effect `(quad_cc)` instead; for using no centering, work with the effect `(quad_nc)` instead;

3. *threshold shape effect* `(threshold)`, `(threshold2)`, `(threshold3)`, `(threshold4)`, where the attractiveness of the behavior depends on whether it is larger than the internal effect parameter p:
$s_{i3}^{\text{beh}}(x, z) = I\{z_i \geq p\}$;
note that by exception z_i here is the uncentered dependent behavioral variable! Multiple versions of this effect are available to allow the use of more than one threshold.

4. *p-near similarity effect* `(simAllNear)`,
$s_{i4}(z) = \sum_{j \neq i} \big(\max\{p - |z_j - z_i|, 0\} \big) = \sum_{j \neq i} I\{|z_j - z_i| \leq p\} \big(p - |z_j - z_i|\big)$
where $p \geq 1$ is an internal effect parameter; note that this is an influence effect from all actors in the network, which does not take a network into account;

5. *p-far similarity effect* `(simAllFar)`,
$s_{i5}(z) = \sum_{j \neq i} \min\{p - |z_j - z_i|, 0\} = \sum_{j \neq i} I\{|z_j - z_i| \geq p\} \big(p - |z_j - z_i|\big)$
where $p \geq 1$ is an internal effect parameter; note that this also is an influence effect from all actors in the network, which does not take a network into account;

6. *group average* `(avGroup)`, defined by ego's value multiplied by the average of all values z_h for this period,
$s_{i6}^{\text{beh}}(x, z) = z_i \, (\overline{z} - c_p)$;
here $\overline{z}$ is the mean of the values z_h for all actors h in the group, and c_p is a centering constant depending on the internal effect parameter p. Denote the overall mean used for centering the observed Z values by m_Z (see the footnote on p. 194). If $p \leq 0.5$, centering is no different than for the z_i values generally, so $c_p = 0$; if $p > 0.5$, centering is by $c_p = p - m_Z$. For the original non-centered values of Z, this means that the mean is centered around the value p, the effect parameter itself. Note that this can be any real value, not necessarily integer.
This effect is useful especially for multi-group data sets, where the average value varies between groups. For multi-group data sets, centering by the groupwise mean may be less desirable, and it will be better to center by a value p close to the median of Z. (Note that the centering in a multi-group data set will affect the linear shape parameter.)

7. *group total* `(totGroup)`, defined by ego's value multiplied by the sum of all values z_h for this period,
$s_{i7}^{\text{beh}}(x, z) = z_i \, (\sum_h z_h - c_p)$;
here $\sum_h z_h$ is the sum of the values z_h for all actors h in the group, and c_p is a centering constant depending on the internal effect parameter p, defined as for the `avGroup` effect above.

8. *constant effect* `(constant)`,
an elementary effect for behavior, indicating that the behavior stays constant:
$s_{i8}^{\text{beh}}(x, z^0, z) = I\{z_i = z_i^0\}$.
The advice for this effect is not to use it, unless you have very specific reasons.

For estimation, this effect is totally collinear with the rate effect; therefore its parameter cannot be estimated. It is only of theoretical importance, and can be used perhaps in interaction with other effects.

Next there is a list of effects that have to do with the influence of the network on the behavior. To specify such effects in RSiena using, e.g., function set_effect, it is necessary[20] to specify the dependent behavior variable in the argument depvar as well as the network in the argument covar1. For example,

```
myCoEvolutionEff <- set_effect(myCoEvolutionEff, list(avSim, indeg, outdeg),
        depvar="drinkingbeh", covar1="friendship")
```

The list of these effects is the following.

9. *average similarity effect* (avSim), defined by the average of centered similarity scores sim_{ij}^z between i and the other actors j to whom he is tied,
   $s_{i9}^{\mathrm{beh}}(x,z) = x_{i+}^{-1} \sum_j x_{ij}(\mathrm{sim}_{ij}^z - \widehat{\mathrm{sim}^z})$;
   (and 0 if $x_{i+} = 0$) ;
   the definition of sim_{ij}^z is discussed in section 5.2, see (1);

10. *total similarity effect* (totSim), defined by the sum of centered similarity scores sim_{ij}^z between i and the other actors j to whom he is tied,
    $s_{i10}^{\mathrm{beh}}(x,z) = \sum_j x_{ij}(\mathrm{sim}_{ij}^z - \widehat{\mathrm{sim}^z})$;
    it may be helpful for estimating this effect also to include the outdeg effect, with which it is correlated;

11. *average incoming similarity effect* (avInSim), defined by the average of centered similarity scores sim_{ij}^z between i and the other actors j who send a tie to i,
    $s_{i11}^{\mathrm{beh}}(x,z) = x_{+i}^{-1} \sum_j x_{ji}(\mathrm{sim}_{ij}^z - \widehat{\mathrm{sim}^z})$;
    (and 0 if $x_{+i} = 0$) ;

12. *total incoming similarity effect* (totInSim), defined by the sum of centered similarity scores sim_{ij}^z between i and the other actors j who send a tie to i,
    $s_{i12}^{\mathrm{beh}}(x,z) = \sum_j x_{ji}(\mathrm{sim}_{ij}^z - \widehat{\mathrm{sim}^z})$;

13. *indegree effect* (indeg),
    $s_{i13}^{\mathrm{beh}}(x,z) = z_i \sum_j x_{ji}$;

14. *outdegree effect* (outdeg),
    $s_{i14}^{\mathrm{beh}}(x,z) = z_i \sum_j x_{ij}$;

15. *reciprocated degree effect* (recipDeg),
    $s_{i15}^{\mathrm{beh}}(x,z) = z_i \sum_j x_{ij}\, x_{ji}$;

16. *in-isolate effect* (isolate), the differential attractiveness of the behavior for in-isolates,
    $s_{i16}^{\mathrm{beh}}(x,z) = z_i\, I\{x_{+i} = 0\}$,
    where again $I\{A\}$ denotes the indicator function of the condition A;
    (before version 1.2-16 this was called the *isolate effect*;)

[20] If this behavior variable is the only dependent variable, then this is not necessary. But this seldom happens.

17. *out-isolate effect* (`outIsolate`), the differential attractiveness of the behavior for out-isolates,
$s^{\text{beh}}_{i17}(x,z) = z_i\, I\{x_{i+} = 0\}$,
where $I\{A\}$ still denotes the indicator function of the condition A;

18. *average similarity* $\times$ *reciprocity effect* (`avSimRecip`), defined by the sum of centered similarity scores sim^z_{ij} between i and the other actors j to whom he is reciprocally tied,
$s^{\text{beh}}_{i18}(x,z) = x^{-1}_{i(r)} \sum_j x_{ij}x_{ji}(\text{sim}^z_{ij} - \widehat{\text{sim}^z})$;
(and 0 if $x_{i(r)} = 0$) ;

19. *total similarity* $\times$ *reciprocity effect* (`totSimRecip`), defined by the sum of centered similarity scores sim^z_{ij} between i and the other actors j to whom he is reciprocally tied,
$s^{\text{beh}}_{i19}(x,z) = \sum_j x_{ij}x_{ji}(\text{sim}^z_{ij} - \widehat{\text{sim}^z})$;

20. *average similarity* $\times$ *popularity alter effect* (`avSimPopAlt`), defined by the average of centered similarity scores sim^z_{ij} between i and the other actors j to whom he is tied, multiplied by their indegrees,
$s^{\text{beh}}_{i20}(x,z) = x^{-1}_{i+} \sum_j x_{ij}x_{+j}(\text{sim}^z_{ij} - \widehat{\text{sim}^z})$;
(and 0 if $x_{i+} = 0$) ;

21. *average incoming similarity* $\times$ *popularity alter effect* (`avInSimPopAlt`), defined by the average of centered similarity scores sim^z_{ij} between i and the other actors j who send a tie to i, multiplied by their indegrees,
$s^{\text{beh}}_{i21}(x,z) = x^{-1}_{+i} \sum_j x_{ji}x_{+j}(\text{sim}^z_{ij} - \widehat{\text{sim}^z})$;
(and 0 if $x_{+i} = 0$) ;

22. *total popularity alter effect* (`popAlt`), defined by the sum of in-degrees of the other actors j to whom i is tied,
$s^{\text{beh}}_{i22}(x,z) = z_i \sum_j x_{ij}x_{+j}$;
this effect may be useful, e.g., as a control effect for the total similarity $\times$ popularity alter effect and in distance 2 influence effects;

23. *total popularity alter effect* (`totPopAlt`), defined by the average in-degrees of the other actors j to whom i is tied,
$s^{\text{beh}}_{i23}(x,z) = z_i\, x^{-1}_{i+} \sum_j x_{ij}x_{+j}$;
(and 0 if $x_{i+} = 0$) ;
this effect may be useful, e.g., as a control effect for the average similarity $\times$ popularity alter effect;

24. *total similarity* $\times$ *popularity alter effect* (`totSimPopAlt`), defined by the sum of centered similarity scores sim^z_{ij} between i and the other actors j to whom he is tied, multiplied by their indegrees,
$s^{\text{beh}}_{i24}(x,z) = \sum_j x_{ij}x_{+j}(\text{sim}^z_{ij} - \widehat{\text{sim}^z})$;

25. *total similarity* $\times$ *popularity alter effect* (`totInSimPopAlt`), defined by the sum of centered similarity scores sim^z_{ij} between i and the other actors j who send a tie to i, multiplied

by their indegrees,
$s^{\text{beh}}_{i25}(x,z) = \sum_j x_{ji}x_{+j}(\text{sim}^z_{ij} - \widehat{\text{sim}^z})$;

26. *average similarity* $\times$ *reciprocity* $\times$ *popularity alter effect* (`avSimRecPop`), defined by the sum of centered similarity scores sim^z_{ij} between i and the other actors j to whom he is reciprocally tied, multiplied by their indegrees,
$s^{\text{beh}}_{i26}(x,z) = x^{-1}_{i(r)} \sum_j x_{ij}x_{ji}x_{+j}(\text{sim}^z_{ij} - \widehat{\text{sim}^z})$;
(and 0 if $x_{i(r)} = 0$) ;

27. *total similarity* $\times$ *reciprocity* $\times$ *popularity alter effect* (`totSimRecPop`), defined by the sum of centered similarity scores sim^z_{ij} between i and the other actors j to whom he is reciprocally tied, multiplied by their indegrees,
$s^{\text{beh}}_{i27}(x,z) = \sum_j x_{ij}x_{ji}x_{+j}(\text{sim}^z_{ij} - \widehat{\text{sim}^z})$;

28. *average similarity* $\times$ *variance alter effect* (`avSimVarAlt`), defined by the average of centered similarity scores sim^z_{ij} between i and the other actors j to whom he is tied, multiplied the centered variance of the behavior of his alters,
$s^{\text{beh}}_{i28}(x,z) = x^{-1}_{i+} \sum_j x_{ij}(\text{sim}^z_{ij} - \widehat{\text{sim}^z}) \times \left(x^{-1}_{i+} \sum_j (x_{ij}z_j)^2 - x^{-2}_{i+} \left(\sum_j x_{ij}z_j\right)^2 - \widehat{\text{var}^z}\right)$
(and 0 if $x_{i+} = 0$) ;
where $\widehat{\text{var}^z}$ is the variance of the observed behavior values in all but the final wave;

29. *average attraction toward higher effect* (`avAttHigher`), defined by the average of non-centered similarity scores sim^z_{ij} between i and the actors j to whom he is tied, where the similarity is replaced by 1 for those who have a lower behavior than i,

$$\begin{aligned} s^{\text{beh}}_{i29}(x,z) &= x^{-1}_{i+} \sum_j x_{ij} \left(I\{z_j > z_i\} \, \text{sim}^z_{ij} + I\{z_j \leq z_i\} \right) \\ &= x^{-1}_{i+} \sum_j x_{ij} \left(1 - \frac{\max(z_j - z_i, 0)}{r_Z} \right); \end{aligned}$$

(and 0 if $x_{i+} = 0$) ;
the definition of sim^z_{ij} is discussed in section 5.2, see (1); r_Z is the range of Z;

30. *average attraction toward lower effect* (`avAttLower`), defined by the average of non-centered similarity scores sim^z_{ij} between i and the actors j to whom he is tied, where the similarity is replaced by 1 for those who have a higher behavior than i,

$$\begin{aligned} s^{\text{beh}}_{i30}(x,z) &= x^{-1}_{i+} \sum_j x_{ij} \left(I\{z_j < z_i\} \, \text{sim}^z_{ij} + I\{z_j \geq z_i\} \right) \\ &= x^{-1}_{i+} \sum_j x_{ij} \left(1 - \frac{\max(z_i - z_j, 0)}{r_Z} \right); \end{aligned}$$

(and 0 if $x_{i+} = 0$) ;

31. *total attraction toward higher effect* (`totAttHigher`), defined by the sum of non-centered similarity scores sim_{ij}^{z} between i and the actors j to whom he is tied, where the similarity is replaced by 1 for those who have a lower behavior than i,

$$\begin{aligned} s_{i31}^{\mathrm{beh}}(x,z) &= \sum_j x_{ij} \Big(I\{z_j > z_i\} \, \mathrm{sim}_{ij}^{z} \; + \; I\{z_j \leq z_i\} \Big) \\ &= \sum_j x_{ij} \left(1 \; - \; \frac{\max(z_j - z_i, 0)}{r_Z} \right); \end{aligned}$$

32. *total attraction toward lower effect* (`totAttLower`), defined by the sum of non-centered similarity scores sim_{ij}^{z} between i and the actors j to whom he is tied, where the similarity is replaced by 1 for those who have a higher behavior than i,

$$\begin{aligned} s_{i32}^{\mathrm{beh}}(x,z) &= \sum_j x_{ij} \Big(I\{z_j < z_i\} \, \mathrm{sim}_{ij}^{z} \; + \; I\{z_j \geq z_i\} \Big) \\ &= \sum_j x_{ij} \left(1 \; - \; \frac{\max(z_i - z_j, 0)}{r_Z} \right); \end{aligned}$$

33. *average alter effect* (`avAlt`), defined by i's behavior multiplied by the average behavior of his alters (a kind of ego-alter behavior covariance),
$s_{i33}^{\mathrm{beh}}(x,z) = z_i \big(\sum_j x_{ij} \, z_j \big) / \big(\sum_j x_{ij} \big)$
(and the mean behavior, i.e. 0, if the ratio is 0/0) ; note that the mean behavior by default is the empirical grand mean in the data; for using the contemporaneous mean, work with (`avAlt_cc`) instead;

34. *total alter effect* (`totAlt`), defined by i's behavior multiplied by the sum of behavior of his alters,
$s_{i34}^{\mathrm{beh}}(x,z) = z_i \big(\sum_j x_{ij} \, z_j \big)$;
also here a contemporaneously centered variant (`totAlt_cc`) exists;

35. *average in-alter effect* (`avInAlt`), defined by i's behavior multiplied by the average behavior of his in-alters (a kind of ego-alter behavior covariance),
$s_{i35}^{\mathrm{beh}}(x,z) = z_i \big(\sum_j x_{ji} \, z_j \big) / \big(\sum_j x_{ji} \big)$
(and the mean behavior, i.e. 0, if the ratio is 0/0) ;

36. *total alter effect* (`totInAlt`), defined by i's behavior multiplied by the sum of behavior of his in-alters,
$s_{i36}^{\mathrm{beh}}(x,z) = z_i \big(\sum_j x_{ji} \, z_j \big)$;

37. *average reciprocated alter effect* (`avRecAlt`), defined by i's behavior multiplied by the average behavior of his reciprocated alters,
$s_{i37}^{\mathrm{beh}}(x,z) = z_i \big(\sum_j x_{ij} \, x_{ji} \, z_j \big) / \big(\sum_j x_{ij} \, x_{ji} \big)$
(and 0 if the ratio is 0/0) ;

38. *total reciprocated alter effect* (`totRecAlt`), defined by i's behavior multiplied by the sum of behavior of his reciprocated alters,
$s^{\text{beh}}_{i38}(x,z) = z_i\big(\sum_j x_{ij}\, x_{ji}\, z_j\big)$;

39. *average alter* $\times$ *popularity effect* (`avAltPop`), defined by the behavior multiplied by the average behavior of his alters, multiplied by their indegrees,
$s^{\text{beh}}_{i39}(x,z) = z_i\, x_{i+}^{-1}\,\big(\sum_j x_{ij}\, x_{+j}\, z_j\big)$;
(and 0 if $x_{i+} = 0$) ;

40. *total alter* $\times$ *popularity effect* (`totAltPop`), defined by the behavior multiplied by the sum of the behavior of his alters, multiplied by their indegrees,
$s^{\text{beh}}_{i40}(x,z) = z_i\,\big(\sum_j x_{ij}\, x_{+j}\, z_j\big)$;

41. only in RSienaTest: *average Simmelian alter effect*, (`avSimmelianAlt`), the effect of the average behavior for the Simmelian alters. The Simmelian transformation of the network is the network composed of all ties embedded in at least one complete triad:

$$x^{\text{simm}}_{ij} = \begin{cases} 1 & x_{ij} = x_{ji} = 1 \text{ and} \\ & \quad \text{there is at least one } h \text{ for which } x_{hi} = x_{ih} = x_{hj} = x_{jh} = 1 \\ 0 & \text{else.} \end{cases}$$

In other words, these ties must be reciprocal, and there must be at least one third actor to whom both have a mutual tie. The average Simmelian alter effect is
$s^{\text{beh}}_{i41}(x,z) = z_i\big(\sum_j x^{\text{simm}}_{ij}\, z_j\big)/\big(\sum_j x^{\text{simm}}_{ij}\big)$
(and the mean behavior, i.e. 0, if the ratio is 0/0) ;

42. only in RSienaTest: *total Simmelian alter effect*, (`totSimmelianAlt`), the effect of the sum of behavior for the Simmelian alters, defined by
$s^{\text{beh}}_{i42}(x,z) = z_i\big(\sum_j x^{\text{simm}}_{ij}\, z_j\big)$;

43. *average alter effect at distance 2* (`avAltDist2`), defined by i's behavior multiplied by the average of the alter-averages $\breve{z}^{(-i)}_j$ of his alters, excluding the contribution from a tie $j \to i$ (if any), defined by

$$\breve{z}^{(-i)}_j = \begin{cases} \dfrac{\sum_{h\neq i} x_{jh}\, z_h}{x_{j+} - x_{ji}} & \text{if } x_{j+} - x_{ji} > 0 \\ 0 & \text{if } x_{j+} - x_{ji} = 0. \end{cases} \tag{34}$$

(also see (23)),
$s^{\text{beh}}_{i43}(x,z) = z_i\big(\sum_j x_{ij}\, \breve{z}^{(-i)}_j\big)/\big(\sum_j x_{ij}\big)$
(and the mean behavior, i.e. 0, if the ratio is 0/0) ;

44. *total alter effect at distance 2* (`totAltDist2`), defined by i's behavior multiplied by the total of the alter-totals of his alters, excluding the contribution from a tie $j \to i$ (if any),
$s^{\text{beh}}_{i44}(x,z) = z_i \sum_j x_{ij}\, \sum_{h\neq i} x_{jh}\, z_h = z_i \sum_j x_{ij}\,(x_{j+} - x_{ji})\, \breve{z}^{(-i)}_j$;

45. *average total alter effect at distance 2* (`avTAltDist2`), defined by i's behavior multiplied by the average of the alter-totals of his alters, excluding the contribution from a tie $j \to i$ (if any),
$s^{\text{beh}}_{i45}(x,z) = z_i\big(\sum_j x_{ij}\,(x_{j+}-x_{ji})\,\check{z}^{(-i)}_j\big)/\big(\sum_j x_{ij}\big) = z_i\big(\sum_j x_{ij}\sum_{h\neq i} x_{jh}\,z_h\big)/\big(\sum_j x_{ij}\big)$
(and the mean behavior, i.e. 0, if the ratio is 0/0) ;

46. *total average alter effect at distance 2* (`totAAltDist2`), defined by i's behavior multiplied by the total of the alter-averages $z^{(-i)}_j$ of his alters, excluding the contribution from a tie $j \to i$ (if any),
$s^{\text{beh}}_{i46}(x,z) = z_i\big(\sum_j x_{ij}\,\check{z}^{(-i)}_j\big)$;

47. *average incoming alter effect at distance 2* (`avInAltDist2`), defined by i's behavior multiplied by the average of the incoming alter averages $\check{z}^{(-i)}_j$ of his alters, excluding the contribution from a tie $i \to j$ (if any), defined by

$$\check{z}^{(-i)}_j = \begin{cases} \dfrac{\sum_{h\neq i} x_{hj}\,z_h}{x_{+j}-x_{ij}} & \text{if } x_{+j}-x_{ij} > 0 \\ 0 & \text{if } x_{+j}-x_{ij} = 0. \end{cases} \tag{35}$$

(cf. (34)),
$s^{\text{beh}}_{i47}(x,z) = z_i\big(\sum_j x_{ij}\,\check{z}^{(-i)}_j\big)/\big(\sum_j x_{ij}\big)$
(and the mean behavior, i.e. 0, if the ratio is 0/0) ; for using no centering, work with the effect (`avInAltDist2_nc`) instead;

48. *total incoming alter effect at distance 2* (`totInAltDist2`), defined by i's behavior multiplied by the total of the incoming alter-totals of his alters, excluding the contribution from a tie $i \to j$ (if any),
$s^{\text{beh}}_{i48}(x,z) = z_i\sum_j x_{ij}\,\sum_{h\neq i} x_{hj}\,z_h = z_i\sum_j x_{ij}\,(x_{+j}-x_{ij})\,\check{z}^{(-i)}_j$; for using no centering, work with the effect (`totInAltDist2_nc`) instead;

49. *average total incoming alter effect at distance 2* (`avTInAltDist2`), defined by i's behavior multiplied by the average of the incoming alter-totals of his alters, excluding the contribution from a tie $i \to j$ (if any),
$s^{\text{beh}}_{i49}(x,z) = z_i\big(\sum_j x_{ij}\,(x_{+j}-x_{ij})\,\check{z}^{(-i)}_j\big)/\big(\sum_j x_{ij}\big)$
(and the mean behavior, i.e. 0, if the ratio is 0/0) ; for using no centering, work with the effect (`avTInAltDist2_nc`) instead;

50. *total average incoming alter effect at distance 2* (`totAInAltDist2`), defined by i's behavior multiplied by the total of the incoming alter-averages $z^{(-i)}_j$ of his alters, excluding the contribution from a tie $i \to j$ (if any),
$s^{\text{beh}}_{i50}(x,z) = z_i\big(\sum_j x_{ij}\,\check{z}^{(-i)}_j\big)$; for using no centering, work with the effect (`totAInAltDist2_nc`) instead;

51. two variations of the *GWDSP (geometrically weighted dyadwise shared partners)* alter effects: *total GWDSP I -> K -> J alter* (`totGwdspFFAlt`) and *total GWDSP I -> K <- J*

alter (totGwdspFBAlt). These are geometrically weighted versions of totAltDist2 and totInAltDist2, respectively, penalizing additional paths between ego and his distance 2 alters defined by

$$s_{i51}^{\mathrm{beh}}(x,z,\alpha) \;=\; z_i \sum_{h;\, h\neq i}^{n} z_h e^{\alpha} \left\{ 1 \;-\; \left(1-e^{-\alpha}\right)^{\sum_{j=1} x_{ij}x_{jh}} \right\} \tag{36}$$

and

$$s_{i51}^{\mathrm{beh}}(x,z,\alpha) \;=\; z_i \sum_{h;\, h\neq i}^{n} z_h e^{\alpha} \left\{ 1 \;-\; \left(1-e^{-\alpha}\right)^{\sum_{j=1} x_{ij}x_{hj}} \right\} \;; \tag{37}$$

for using no centering, work with the effect (totGwdspFFAlt_nc) respectively (totGwdspFBAlt_nc) instead;

52. *average in-similarity effect at distance 2* (avInSimDist2), defined by i's average similarity to the incoming alter averages $\check{z}_j^{(-i)}$ of his out-alters, excluding the contribution from a tie $i \to j$ (if any),
$s_{i52}^{\mathrm{beh}}(x,z) = \Big(\sum_j x_{ij}\, I\{x_{+j} - x_{ij} > 0\} \left(\mathrm{sim}^z(z_i, \check{z}_j^{(-i)}) - \widehat{\mathrm{sim}^z}\right) \Big) / \sum_j x_{ij}$,
where the similarity scores sim^z are defined as usual;
this is the behavior/influence effect parallel to simEgoInDist2;

53. *total in-similarity effect at distance 2* (totInSimDist2), defined by the sum of i's similarity to the incoming alter averages $\check{z}_j^{(-i)}$ of his out-alters, excluding the contribution from a tie $i \to j$ (if any),
$s_{i53}^{\mathrm{beh}}(x,z) = \sum_j x_{ij}\, I\{x_{+j} - x_{ij} > 0\} \left(\mathrm{sim}^z(z_i, \check{z}_j^{(-i)}) - \widehat{\mathrm{sim}^z}\right)$,
where again the similarity scores sim^z are defined as usual;

54. *average in-similarity effect at distance 2* (avInSimDist2), defined by i's average similarity to the incoming alter averages $\check{z}_j^{(-i)}$ of his out-alters, excluding the contribution from a tie $i \to j$ (if any),
$s_{i54}^{\mathrm{beh}}(x,z) = \Big(\sum_j x_{ij}\, I\{x_{+j} - x_{ij} > 0\} \left(\mathrm{sim}^z(z_i, \check{z}_j^{(-i)}) - \widehat{\mathrm{sim}^z}\right) \Big) / \sum_j x_{ij}$,
where the similarity scores sim^z are defined as usual;
this is the behavior/influence effect parallel to simEgoInDist2;

55. *total in-similarity effect at distance 2* (totInSimDist2), defined by the sum of i's similarity to the incoming alter averages $\check{z}_j^{(-i)}$ of his out-alters, excluding the contribution from a tie $i \to j$ (if any),
$s_{i55}^{\mathrm{beh}}(x,z) = \sum_j x_{ij}\, I\{x_{+j} - x_{ij} > 0\} \left(\mathrm{sim}^z(z_i, \check{z}_j^{(-i)}) - \widehat{\mathrm{sim}^z}\right)$,
where again the similarity scores sim^z are defined as usual;

56. *maximum alter effect* (maxAlt), defined by i's behavior multiplied by the maximum behavior of his alters,
$s_{i56}^{\mathrm{beh}}(x,z) = z_i \big(\max_j x_{ij}\, z_j \big)$
(and the mean behavior, i.e. 0, if $\sum_j x_{ij} = 0$) ;

57. *minimum alter effect* (`minAlt`), defined by i's behavior multiplied by the minimum behavior of his alters,
$s^{\text{beh}}_{i57}(x,z) = z_i\big(\min_j x_{ij}\, z_j\big)$
(and the mean behavior, i.e. 0, if $\sum_j x_{ij} = 0$) ;

58. *variance alter effect* (`varAlt`), defined by i's behavior multiplied by the centered variance of the behavior of his alters,
$s^{\text{beh}}_{i58}(x,z) = z_i\big(x_{i+}^{-1}\sum_j (x_{ij}z_j)^2 - x_{i+}^{-1}(\sum_j x_{ij}z_j)^2 - \widehat{\text{var}^z}\big)$
(and 0 if $x_{i+} = 0$) ;
where $\widehat{\text{var}^z}$ is the variance of the observed behavior values in all but the final wave;

59. *dense triads effect* (`behDenseTriads`), defined by i's behavior multiplied by the number of dense triads in which actor i is located,
$s^{\text{beh}}_{i59}(x,z) = z_i \sum_{j,h} I\{x_{ij} + x_{ji} + x_{ih} + x_{hi} + x_{jh} + x_{hj} \geq p\}$,
where the internal effect parameter p is either 5 or 6;

60. *similarity in dense triads effect* (`simDenseTriads`), defined by the total sum of i's behavior similarity to others in the dense triads in which actor i is located,
$s^{\text{beh}}_{i60}(x,z) = \sum_{j,h}(\text{sim}(v_i,v_j) + \text{sim}(v_i,v_h))\, I\{x_{ij} + x_{ji} + x_{ih} + x_{hi} + x_{jh} + x_{hj} \geq p\}$,
where p is either 5 or 6;

61. *peripheral effect*, defined by i's behavior multiplied by the number of dense triads to which actor i stands in a unilateral-peripheral relation,
$s^{\text{beh}}_{i61}(x,z) =$
$\quad z_i \sum_{j,h,k} x_{ij}(1-x_{ji})(1-x_{hi})(1-x_{ki}) I\{x_{ij} + x_{ji} + x_{ih} + x_{hi} + x_{jh} + x_{hj}\,) \geq p\}$,
where p is the same constant as in the *dense triads* effect;
for symmetric networks, the unilateral condition is dropped, and the effect is
$s^{\text{beh}}_{i61}(x,z) = z_i \sum_{j,h,k} x_{ij}(1-x_{hi})(1-x_{ki}) I\{x_{ij} + x_{ji} + x_{ih} + x_{hi} + x_{jh} + x_{hj}\,) \geq p\}$;

62. *average similarity* $\times$ *popularity ego effect* (`avSimPopEgo`), defined by the sum of centered similarity scores sim^z_{ij} between i and the other actors j to whom he is tied, multiplied by ego's indegree,
$s^{\text{beh}}_{i62}(x,z) = x_{+i}\, x_{i+}^{-1} \sum_j x_{ij}(\text{sim}^z_{ij} - \widehat{\text{sim}^z})$;
(and 0 if $x_{i+} = 0$) ;
because of collinearity, under the Method of Moments this cannot be estimated together with the average similarity $\times$ popularity alter effect;

63. *indegree-weighted group average* (`indegAvGroup`), defined by ego's value multiplied by the indegree-weighted average of all values z_h for this period,
$s^{\text{beh}}_{i63}(x,z) = z_i \left(\frac{\sum_h x_{+h} z_h}{\sum_h x_{+h}} - c_p\right)$;

here x_{+h} is the indegree of actor h in the specified network, and c_p is a centering constant as for `avGroup`.

64. *indegree-weighted group total* (`indegTotGroup`), defined by ego's value multiplied by the indegree-weighted sum of all values z_h for this period,
$s^{\rm beh}_{i64}(x,z) = z_i\,(\sum_h x_{+h} z_h - c_p)$;
here x_{+h} is the indegree of actor h in the specified network, and c_p is a centering constant as for `totGroup`.

13.2.1.1 Effects of multiple networks

If there are more than one dependent network variables, denoted X_1 and X_2, they can operate jointly on the behavioural dependent variable using the following effects. X_1 is given as `covar1`, X_2 as `covar2`.

For the combined degree-effects, mentioned first, `F` means 'Forward', `B` means 'Backward', and `R` means 'Reciprocal'.

65. *double outdegree effect* (`FFDeg`),
$s^{\rm beh}_{i65}(x,z) = z_i \sum_j x_{1ij}\, x_{2ij}$;
if the internal effect parameter p is at least 2, the outdegree for X_1 is subtracted (perhaps this leads to confusion...):
$s^{\rm beh}_{i65}(x,z) = z_i \Big(\sum_j x_{1ij}\, x_{2ij} \;-\; \sum_j x_{1ij} \Big)$;
note that in this case, the factor between $\Big(\ldots \Big)$ is non-positive;

66. *double indegree effect* (`BBDeg`),
$s^{\rm beh}_{i66}(x,z) = z_i \sum_j x_{1ji}\, x_{2ji}$;
if the internal effect parameter p is at least 2, the indegree for X_1 is subtracted:
$s^{\rm beh}_{i66}(x,z) = z_i \Big(\sum_j x_{1ji}\, x_{2ji} \;-\; \sum_j x_{1ji} \Big)$;
note that also in this case, the factor between $\Big(\ldots \Big)$ is non-positive;

67. *combined out-indegree effect* (`FBDeg`),
$s^{\rm beh}_{i67}(x,z) = z_i \sum_j x_{1ij}\, x_{2ji}$;
if the internal effect parameter p is at least 2, the outdegree for X_1 is subtracted:
$s^{\rm beh}_{i67}(x,z) = z_i \Big(\sum_j x_{1ij}\, x_{2ji} \;-\; \sum_j x_{1ij} \Big)$;
note that also in this case, the factor between $\Big(\ldots \Big)$ is non-positive;

68. *combined out-reciprocated degree effect* (`FRDeg`),
$s^{\rm beh}_{i68}(x,z) = z_i \sum_j x_{1ij}\, x_{2ij}\, x_{2ji}$;
if the internal effect parameter p is at least 2, the outdegree for X_1 is subtracted:
$s^{\rm beh}_{i68}(x,z) = z_i \Big(\sum_j x_{1ij}\, x_{2ij}\, x_{2ji} \;-\; \sum_j x_{1ij} \Big)$;
note that also in this case, the factor between $\Big(\ldots \Big)$ is non-positive;

69. *combined in-reciprocated degree effect* (BRDeg),
$s^{\text{beh}}_{i69}(x,z) = z_i \sum_j x_{1ji}\, x_{2ij}\, x_{2ji}$;
if the internal effect parameter p is at least 2, the indegree for X_1 is subtracted:
$s^{\text{beh}}_{i69}(x,z) = z_i \Big(\sum_j x_{1ji}\, x_{2ij}\, x_{2ji} \; - \; \sum_j x_{1ji} \Big)$;
note that also in this case, the factor between $\Big(\ldots \Big)$ is non-positive.

70. *outdegree mixed popularity effect* (outdegMixedPop), defined by i's behavior multiplied by the sum of the X_2-indegrees of ego's X_1-out-alters :
$s^{\text{beh}}_{i70}(x,z) = z_i \sum_j x_{1ij}\, x_{2+j}$;

71. *indegree mixed popularity effect* (indegMixedPop), defined by i's behavior multiplied by the sum of the X_2-indegrees of ego's X_1-in-alters :
$s^{\text{beh}}_{i71}(x,z) = z_i \sum_j x_{1ji}\, x_{2+j}$;

13.2.1.2 Monadic covariate effects

For each actor-dependent covariate v_j (recall that by default these are centered internally by SIENA) as well as for each of the other dependent behavior variables (for notational simplicity here also denoted v_j), there are the following effects.

72. *main covariate effect* (effFrom),
$s^{\text{beh}}_{i72}(x,z) = z_i v_i$;
here too, the other dependent behavioral variables are centered so that they have overall mean 0;

73. *alter's covariate average* effect on behavior z (avXAlt), formerly called (AltsAvAlt), defined as the product of i's behavior z_i and i's alters' covariate-average $\breve{v}_i$ as defined in (22),
$s^{\text{beh}}_{i73}(x,z) = z_i\, \breve{v}_i$.
This is similar to the 'average alter' effect; for $v_i = z_i$ it would reduce to the latter effect.

74. *alter's covariate total* effect on behavior z (totXAlt),
defined as the product of i's behavior z_i and i's alters' covariate sum $x_{i+}\, \breve{v}_i$ as defined in (44),
$s^{\text{beh}}_{i74}(x,z) = z_i\, x_{i+}\, \breve{v}_i$.
This is similar to the 'total alter' effect; for $v_i = z_i$ it would reduce to the latter effect.

75. *in-alter's covariate average* effect on behavior z (avXInAlt),
defined as the product of i's behavior z_i and i's in-alters' covariate-average $\breve{v}_i$ as defined in

$$\breve{v}_i = \begin{cases} \dfrac{\sum_j x_{ji} v_j}{x_{+i}} & \text{if } x_{+i} > 0 \\ 0 & \text{if } x_{+i} = 0, \end{cases} \tag{38}$$

the effect being
$s^{\text{beh}}_{i75}(x, z) = z_i\, \check{v}_i$.
This is similar to the 'average in-alter' effect; for $v_i = z_i$ it would reduce to the latter effect.

76. *in-alter's covariate total* effect on behavior z `(totXInAlt)`,
defined as the product of i's behavior z_i and i's alters' covariate sum $x_{i+}\, \check{v}_i$ as defined in (39),
$s^{\text{beh}}_{i76}(x, z) = z_i\, x_{i+}\, \check{v}_i$.
This is similar to the 'total in-alter' effect; for $v_i = z_i$ it would reduce to the latter effect.

77. *reciprocal alter's covariate average* effect on behavior z `(avXRecAlt)`,
defined as the product of i's behavior z_i and the average of the covariate values v_j for those to whom i has a reciprocal tie: $s^{\text{beh}}_{i77}(x, z) = z_i\, \dfrac{\sum_j x_{ij}\, x_{ji}\, v_j}{\sum_j x_{ij}\, x_{ji}}$,
where 0/0 is defined as 0;

78. *reciprocal alter's covariate total* effect on behavior z `(totXRecAlt)`,
defined as the product of i's behavior z_i and the sum of the covariate values v_j for those to whom i has a reciprocal tie: $s^{\text{beh}}_{i78}(x, z) = z_i\, \sum_j x_{ij}\, x_{ji}\, v_j$;

79. *maximum alter's covariate effect* `(maxXAlt)`, defined by i's behavior multiplied by the maximum covariate value of his alters,
$s^{\text{beh}}_{i79}(x, z) = z_i\big(\max_j x_{ij}\, v_j\big)$
(and the mean covariate if $\sum_j x_{ij} = 0$) ;

80. *minimum alter's covariate effect* `(minXAlt)`, defined by i's behavior multiplied by the minimum covariate value of his alters,
$s^{\text{beh}}_{i80}(x, z) = z_i\big(\min_j x_{ij}\, v_j\big)$
(and the mean covariate if $\sum_j x_{ij} = 0$) ;

81. *alter's same covariate average* effect on behavior z `(avSameXAlt)`,
defined as the product of i's behavior z_i and the proportion of those to whom i has a tie who also have the same covariate value: $s^{\text{beh}}_{i81}(x, z) = z_i\, \dfrac{\sum_j x_{ij}\, I\{v_j = v_i\}}{\sum_j x_{ij}}$,
where 0/0 is defined as 0;

82. *alter's same covariate total* effect on behavior z `(totSameXAlt)`,
defined as the product of i's behavior z_i and the number of actors j to whom i has a tie and who also have the same covariate value:
$s^{\text{beh}}_{i82}(x, z) = z_i\, \sum_j x_{ij}\, I\{v_j = v_i\}$;

83. *in-alter's same covariate average* effect on behavior z `(avSameXInAlt)`,
defined as the product of i's behavior z_i and the proportion of those from whom i has an

incoming tie who also have the same covariate value: $s_{i83}^{\text{beh}}(x,z) = z_i \dfrac{\sum_j x_{ji}\, I\{v_j = v_i\}}{\sum_j x_{ji}}$, where 0/0 is defined as 0;

84. *in-alter's same covariate total* effect on behavior z (`totSameXInAlt`), defined as the product of i's behavior z_i and the number of actors j from whom i has an incoming tie and who also have the same covariate value: $s_{i84}^{\text{beh}}(x,z) = z_i \sum_j x_{ji}\, I\{v_j = v_i\}$;

85. *reciprocal alter's same covariate average* effect on behavior z (`avSameXRecAlt`), defined as the product of i's behavior z_i and the proportion of those to whom i has a reciprocal tie who also have the same covariate value: $s_{i85}^{\text{beh}}(x,z) = z_i \dfrac{\sum_j x_{ij}\, x_{ji}\, I\{v_j = v_i\}}{\sum_j x_{ij}\, x_{ji}}$, where 0/0 is defined as 0;

86. *reciprocal alter's same covariate total* effect on behavior z (`totSameXRecAlt`), defined as the product of i's behavior z_i and the number of actors j to whom i has a reciprocal tie and who also have the same covariate value: $s_{i86}^{\text{beh}}(x,z) = z_i \sum_j x_{ij}\, x_{ji}\, I\{v_j = v_i\}$;

87. *average alter for same covariate* effect on behavior z (`avAltSameX`), defined as the product of i's behavior z_i and the average behavior of those to whom i has a tie and who also have the same covariate value: $s_{i87}^{\text{beh}}(x,z) = z_i \dfrac{\sum_j x_{ij}\, I\{v_j = v_i\}\, z_j}{\sum_j x_{ij}\, I\{v_j = v_i\}}$, where 0/0 is defined as 0;

88. *total alter for same covariate* effect on behavior z (`totAltSameX`), defined as the product of i's behavior z_i and the sum of behavior of those to whom i has a tie and who also have the same covariate value: $s_{i88}^{\text{beh}}(x,z) = z_i \sum_j x_{ij}\, I\{v_j = v_i\}\, z_j$;

89. *average reciprocated alter for same covariate* effect on behavior z (`avRecAltSameX`), defined as the product of i's behavior z_i and the average behavior of those to whom i has a reciprocated tie and who also have the same covariate value: $s_{i89}^{\text{beh}}(x,z) = z_i \dfrac{\sum_j x_{ij}\, x_{ji}\, I\{v_j = v_i\}\, z_j}{\sum_j x_{ij}\, x_{ji}\, I\{v_j = v_i\}}$, where 0/0 is defined as 0;

90. *total reciprocated alter for same covariate* effect on behavior z (`totRecAltSameX`), defined as the product of i's behavior z_i and the sum of behavior of those to whom i has a

reciprocated tie and who also have the same covariate value:
$s^{\text{beh}}_{i90}(x, z) = z_i \sum_j x_{ij}\, x_{ji}\, I\{v_j = v_i\}\, z_j$;

91. *absolute contrast outdegree – covariate* `(degAbsContrX)`, defined as the product of i's behavior z_i and the absolute difference between i's out-degree and his covariate value,
$s^{\text{beh}}_{i91}(x, z) = z_i \mid x_{i+} - v_i \mid$;
working with a non-centered covariate may be advisable for this effect;

92. *positive contrast outdegree – covariate* `(degPosContrX)`, defined as the product of i's behavior z_i and the positive part of the difference between i's out-degree and her covariate value,
$s^{\text{net}}_{i92}(x) = z_i \max\{x_{i+} - v_i, 0\}$;
working with a non-centered covariate may be advisable for this effect;

93. *negative contrast outdegree – covariate* `(degNegContrX)`, defined as the product of i's behavior z_i and the negative part of the difference between i's out-degree and her covariate value,
$s^{\text{net}}_{i93}(x) = -\, z_i \min\{x_{i+} - v_i, 0\} = z_i \max\{v_i - x_{i+}, 0\}$;
working with a non-centered covariate may be advisable for this effect;

94. *alter's covariate total* effect on behavior z `(totXAlt)`,
defined as the product of i's behavior z_i and i's alters' covariate sum $x_{i+}\, \breve{v}_i$ as defined in (44),
$s^{\text{beh}}_{i94}(x, z) = z_i\, x_{i+}\, \breve{v}_i$.
This is similar to the 'total alter' effect; for $v_i = z_i$ it would reduce to the latter effect.

95. *in-alter's covariate average* effect on behavior z `(avXInAlt)`,
defined as the product of i's behavior z_i and i's in-alters' covariate-average $\breve{v}_i$ as defined in

$$\breve{v}_i = \begin{cases} \dfrac{\sum_j x_{ji} v_j}{x_{+i}} & \text{if } x_{+i} > 0 \\ 0 & \text{if } x_{+i} = 0, \end{cases} \tag{39}$$

the effect being
$s^{\text{beh}}_{i95}(x, z) = z_i\, \breve{v}_i$.
This is similar to the 'average in-alter' effect; for $v_i = z_i$ it would reduce to the latter effect.

96. *in-alter's covariate total* effect on behavior z `(totXInAlt)`,
defined as the product of i's behavior z_i and i's alters' covariate sum $x_{i+}\, \breve{v}_i$ as defined in (39),
$s^{\text{beh}}_{i96}(x, z) = z_i\, x_{i+}\, \breve{v}_i$.
This is similar to the 'total in-alter' effect; for $v_i = z_i$ it would reduce to the latter effect.

97. *alter's distance-two covariate average* effect on behavior z `(avXAltDist2)`, defined by i's behavior multiplied by the average of the alter-averages $\breve{v}_j^{(-i)}$ of his alters, excluding the

contribution from a tie $j \to i$ (if any), defined by

$$\breve{v}_j^{(-i)} = \begin{cases} \dfrac{\sum_{h \neq i} x_{jh}\, v_h}{x_{j+} - x_{ji}} & \text{if } x_{j+} - x_{ji} > 0 \\ 0 & \text{if } x_{j+} - x_{ji} = 0. \end{cases} \tag{40}$$

(also see (23) and (34)),
$s_{i97}^{\text{beh}}(x, z) = z_i \big(\sum_j x_{ij}\, \breve{v}_j^{(-i)} \big) / \big(\sum_j x_{ij} \big)$
(and the mean behavior, i.e. 0, if the ratio is 0/0) ;
this is similar to `avAltDist2`, but now for covariate V instead of the behavior Z;

98. *alter's distance-two covariate total* effect on behavior z `(totXAltDist2)`, defined by i's behavior multiplied by the total of the alter-totals on V of his alters, excluding the contribution from a tie $j \to i$ (if any),
$s_{i98}^{\text{beh}}(x, z) = z_i \sum_j x_{ij} \sum_{h \neq i} x_{jh}\, v_h = z_i \sum_j x_{ij}\, (x_{j+} - x_{ji})\, \breve{v}_j^{(-i)}$;
this is similar to `totAltDist2`, but now for covariate V instead of the behavior Z;

99. *average total covariate alter effect at distance 2* `(avTXAltDist2)`, defined by i's behavior multiplied by the average of the alter-totals on V of his alters, excluding the contribution from a tie $j \to i$ (if any),
$s_{i99}^{\text{beh}}(x, z) = z_i \big(\sum_j x_{ij}\, (x_{j+} - x_{ji})\, \breve{v}_j^{(-i)} \big) / \big(\sum_j x_{ij} \big)$
(and the mean behavior, i.e. 0, if the ratio is 0/0) ;
this is similar to `avTAltDist2`, but now for covariate V instead of the behavior Z;

100. *total average covariate alter effect at distance 2* `(totAXAltDist2)`, defined by i's behavior multiplied by the total of the alter-averages on V of his alters, excluding the contribution from a tie $j \to i$ (if any),
$s_{i100}^{\text{beh}}(x, z) = z_i \big(\sum_j x_{ij}\, \breve{v}_j^{(-i)} \big)$;
this is similar to `totAAltDist2`, but now for covariate V instead of the behavior Z;

101. *alter's distance-two incoming covariate average* effect on behavior z `(avXInAltDist2)`, defined by i's behavior multiplied by the average of the incoming alter-averages $\breve{v}_j^{(-i)}$ of his alters, excluding the contribution from a tie $j \to i$ (if any), defined by

$$\breve{v}_j^{(-i)} = \begin{cases} \dfrac{\sum_{h \neq i} x_{hj}\, v_h}{x_{+j} - x_{ij}} & \text{if } x_{+j} - x_{ij} > 0 \\ 0 & \text{if } x_{+j} - x_{ij} = 0. \end{cases} \tag{41}$$

(also see (35)),
$s_{i101}^{\text{beh}}(x, z) = z_i \big(\sum_j x_{ij}\, \breve{v}_j^{(-i)} \big) / \big(\sum_j x_{ij} \big)$
(and the mean behavior, i.e. 0, if the ratio is 0/0) ;
this is similar to `avInAltDist2`, but now for covariate V instead of the behavior Z;

102. *alter's distance-two incoming covariate total* effect on behavior z `(totXInAltDist2)`, defined by i's behavior multiplied by the total of the incoming alter-totals on V of his alters,

excluding the contribution from a tie $i \to j$ (if any),
$s^{\text{beh}}_{i102}(x,z) = z_i \sum_j x_{ij} \sum_{h \neq i} x_{hj}\, v_h = z_i \sum_j x_{ij}\,(x_{+j} - x_{ij})\,\check{v}^{(-i)}_j$;
this is similar to `totInAltDist2`, but now for covariate V instead of the behavior Z;

103. *average total incoming covariate alter effect at distance 2* (`avTXInAltDist2`), defined by i's behavior multiplied by the average of the incoming alter-totals on V of his alters, excluding the contribution from a tie $i \to j$ (if any),
$s^{\text{beh}}_{i103}(x,z) = z_i\big(\sum_j x_{ij}\,(x_{+j} - x_{ij})\,\check{v}^{(-i)}_j\big)/\big(\sum_j x_{ij}\big)$
(and the mean behavior, i.e. 0, if the ratio is 0/0) ;
this is similar to `avTInAltDist2`, but now for covariate V instead of the behavior Z;

104. *total average incoming covariate alter effect at distance 2* (`totAXInAltDist2`), defined by i's behavior multiplied by the total of the incoming alter-averages on V of his alters, excluding the contribution from a tie $i \to j$ (if any),
$s^{\text{beh}}_{i104}(x,z) = z_i\big(\sum_j x_{ij}\,\check{v}^{(-i)}_j\big)$;
this is similar to `totAInAltDist2`, but now for covariate V instead of the behavior Z;

There are also a number of interaction effects between actor covariates (which includes other dependent behavior variables; note that these are used in their centered versions) and influence effects. These have to be specified using `covar1` for the covariate and `covar2` for the network, e.g.,

```
myCoevolutionEff <- set_effect(myCoevolutionEff, avAltEgoX, depvar="smoking",
           covar1="sex", covar2="friendship")
```

Between parentheses: the functionality of the 'ego'-variant of these effects is duplicated by interaction effects created, for example, as follows:

```
myCoevolutionEff <- set\_interaction(myCoevolutionEff, list(effFrom, avAlt),
           depvar="smoking", covar1=c("sex", "friendship"))
```

Furthermore, note that when the same dependent variable is used (i.e., `depvar` and `covar1` are the same), the dependent variable is used as a weight with its value before the ministep, and the potential change is not taken into account; this departs from the concept of evaluation effect, and is a case of an elementary effect.

For the 'alter'-variant, this way of construction will not work because the effect statistic cannot be decomposed into a product of two ego-level statistics. The available effects of this type are the following.

105. *interaction of ego (tie sender) variable with average similarity*, (`avSimEgoX`)
$s^{\text{beh}}_{i105}(x,z) = (v_i/x_{i+})\,\sum_j x_{ij}(\text{sim}^z_{ij} - \widehat{\text{sim}^z})$;
(and the mean behavior, i.e. 0, if $x_{i+} = 0$) ;

106. *interaction of ego (tie sender) variable with total similarity*, (`totSimEgoX`)
$s^{\text{beh}}_{i106}(x,z) = v_i\,\sum_j x_{ij}(\text{sim}^z_{ij} - \widehat{\text{sim}^z})$;

107. *interaction of ego (tie sender) variable with average alter*, `(avAltEgoX)`
$s^{\text{beh}}_{i107}(x,z) = v_i\, z_i\big(\sum_j x_{ij}\, z_j\big)/\big(\sum_j x_{ij}\big)$
(and the mean behavior, i.e. 0, if the ratio is 0/0) ;

108. *interaction of ego (tie sender) variable with total alter*, `(totAltEgoX)`
$s^{\text{beh}}_{i108}(x,z) = v_i\, z_i\big(\sum_j x_{ij}\, z_j\big)$;

109. *interaction of alter (tie receiver) variable with average similarity*, `(avSimAltX)`
$s^{\text{beh}}_{i109}(x,z) = (1/x_{i+})\,\sum_j x_{ij}\, v_j(\text{sim}^z_{ij} - \widehat{\text{sim}^z})$;
(and the mean behavior, i.e. 0, if $x_{i+} = 0$) ;

110. *interaction of alter (tie receiver) variable with total similarity*, `(totSimAltX)`
$s^{\text{beh}}_{i110}(x,z) = \sum_j x_{ij}\, v_j(\text{sim}^z_{ij} - \widehat{\text{sim}^z})$;

111. *interaction of alter (tie receiver) variable with average alter*, `(avAltAltX)`
$s^{\text{beh}}_{i111}(x,z) = z_i\big(\sum_j x_{ij}\, v_j\, z_j\big)/\big(\sum_j x_{ij}\big)$
(and the mean behavior, i.e. 0, if the ratio is 0/0) ;

112. *interaction of alter (tie receiver) variable with total alter*, `(totAltAltX)`
$s^{\text{beh}}_{i112}(x,z) = z_i\big(\sum_j x_{ij}\, v_j\, z_j\big)$;

13.2.1.3 Dyadic covariate effects

For each dyadic covariate W there are the following effects. Recall that the default is to center dyadic covariates; however, it often will be preferable here to use non-centered covariates. It is then important how to define the zero point of the covariate!

113. *alter's average of dyadic covariate* effect on behavior z `(avWAlt)`,
defined as the product of i's behavior z_i and i's alters' dyadic covariate-average $\breve{w}_i$ as defined by

$$\breve{w}_i = \begin{cases} \dfrac{\sum_h x_{ih} w_{ih}}{x_{i+}} & \text{if } x_{i+} > 0 \\ 0 & \text{if } x_{i+} = 0. \end{cases} \tag{42}$$

leading to the effect
$s^{\text{beh}}_{i113}(x,z) = z_i\, \breve{w}_i$.

114. *in-alter's average of dyadic covariate* effect on behavior z `(avWInAlt)`,
defined as the product of i's behavior z_i and i's in-alters' dyadic covariate-average $\breve{w}_h$ as defined by

$$\breve{w}_h = \begin{cases} \dfrac{\sum_h x_{hi} w_{hi}}{x_{+i}} & \text{if } x_{+i} > 0 \\ 0 & \text{if } x_{+i} = 0. \end{cases} \tag{43}$$

leading to the effect
$s^{\text{beh}}_{i114}(x,z) = z_i\, \breve{w}_h$.

115. *in-alter's average of dyadic covariate* effect on behavior z (`avWInAlt`),
defined as the product of i's behavior z_i and i's in-alters' dyadic covariate-average $\breve{w}_h$ as defined by

$$\breve{w}_h = \begin{cases} \dfrac{\sum_h x_{hi} w_{ih}}{x_{+i}} & \text{if } x_{+i} > 0 \\ 0 & \text{if } x_{+i} = 0. \end{cases} \tag{44}$$

leading to the effect
$s^{\text{beh}}_{i115}(x, z) = z_i \, \breve{w}_h$.

116. *alter's total of dyadic covariate W* effect on behavior z (`totWAlt`),
defined as the product of i's behavior z_i and the sum of i's alters' values of the dyadic covariate,
$s^{\text{beh}}_{i116}(x, z) = z_i \, \sum_h x_{ih} w_{ih}$.

117. *in-alter's total of dyadic covariate W* effect on behavior z (`totWInAlt`),
defined as the product of i's behavior z_i and the sum of i's in-alters' values of the dyadic covariate,
$s^{\text{beh}}_{i117}(x, z) = z_i \, \sum_h x_{hi} w_{ih}$.

118. *average alter weighted by dyadic covariate W* (`avAltW`), a weighted version of `avAlt`;
depending on the parameter value (1 or 2):
parameter = 1 : $s^{\text{beh}}_{i118}(x, z) = z_i \big(\sum_j x_{ij} \, w_{ij} \, z_j \big) / \big(\sum_j x_{ij} \big)$
parameter = 2 : $s^{\text{beh}}_{i118}(x, z) = z_i \big(\sum_j x_{ij} \, w_{ij} \, z_j \big) / \big(\sum_j w_{ij} \, x_{ij} \big)$;
(and the mean behavior, i.e. 0, if the ratio is 0/0) ;

119. *average in-alter weighted by dyadic covariate W* (`avInAltW`), a weighted version of `avInAlt`;
depending on the parameter value (1 or 2):
parameter = 1 : $s^{\text{beh}}_{i119}(x, z) = z_i \big(\sum_j x_{ji} \, w_{ij} \, z_j \big) / \big(\sum_j x_{ji} \big)$
parameter = 2 : $s^{\text{beh}}_{i119}(x, z) = z_i \big(\sum_j x_{ji} \, w_{ij} \, z_j \big) / \big(\sum_j w_{ij} \, x_{ji} \big)$;
(and the mean behavior, i.e. 0, if the ratio is 0/0) ;

120. *total alter weighted by dyadic covariate W* (`totAltW`), a weighted version of `totAlt`;
$s^{\text{beh}}_{i120}(x, z) = z_i \, \sum_j x_{ij} \, w_{ij} \, z_j$;

121. *total in-alter weighted by dyadic covariate W* (`totInAltW`), a weighted version of `totInAlt`;
$s^{\text{beh}}_{i121}(x, z) = z_i \, \sum_j x_{ji} \, w_{ij} \, z_j$;

122. *average similarity weighted by dyadic covariate W* (`avSimW`), a weighted version of `avSim`;
depending on the parameter value (1 or 2):
parameter = 1 : $s^{\text{beh}}_{i122}(x, z) = \sum_j x_{ij} \, w_{ij} \, (\text{sim}^z_{ij} - \widehat{\text{sim}^z}) / \big(\sum_j x_{ij} \big)$

parameter = 2 : $s^{\text{beh}}_{i122}(x,z) = \sum_j x_{ij}\, w_{ij}\, (\text{sim}^z_{ij} - \widehat{\text{sim}^z})/\big(\sum_j w_{ij}\, x_{ij}\big)$;
(and the mean behavior, i.e. 0, if the ratio is 0/0) ;
as usual, sim^z_{ij} denotes the similarity between i and j on the behavior Z;

123. *total similarity weighted by dyadic covariate* W (`totSimW`) , a weighted version of `totSim`;
$s^{\text{beh}}_{i123}(x,z) = \sum_j x_{ij}\, w_{ij}\, (\text{sim}^z_{ij} - \widehat{\text{sim}^z})$.

13.2.1.4 User-defined interaction effects

The *user-defined interaction effects* of Section 6.12.4 are defined as follows. Suppose we consider two effects $s^{\text{beh}}_{ia}(x,z)$ and $s^{\text{beh}}_{ib}(x,z)$. Then their interaction effect is defined by

$$s^{\text{beh}}_{i[a*b]}(x,z) \;=\; \frac{1}{z_i}\, s^{\text{beh}}_{ia}(x,z)\, s^{\text{beh}}_{ib}(x,z)\;. \tag{45}$$

For three effects $s^{\text{beh}}_{ia}(x,z)$, $s^{\text{beh}}_{ib}(x,z)$ and $s^{\text{beh}}_{ic}(x,z)$, the interaction effect is defined by

$$s^{\text{beh}}_{i[a*b*c]}(x,z) \;=\; \frac{1}{z_i^2}\, s^{\text{beh}}_{ia}(x,z)\, s^{\text{beh}}_{ib}(x,z)\, s^{\text{beh}}_{ic}(x,z)\;. \tag{46}$$

The division by z_i or z_i^2, respectively, is necessary to offset the fact that all behavior effects of `interactionType = OK` contain a factor z_i, and further do not depend on z_i. For example, the interaction between two main effects (`effFrom`) of actor covariates v_{1i} and v_{2i}, is the same as the main effect of the product variable $v_{1i} \times v_{2i}$, with the proviso that the user-defined interaction does not center the product variable; the user-defined interaction then is defined by

$$s^{\text{beh}}_{i[v_1*v_2]}(x,z) \;=\; \frac{1}{z_i}\,(z_i\, v_{1i})\,(z_i\, v_{2i}) \;=\; z_i\, v_{1i}\, v_{2i}\;, \tag{47}$$

where both component variables v_{1i} and v_{2i} are internally centered, but the product variable will generally not have mean 0.

13.2.2 Behavioral creation function

Also the behavioral model knows the distinction between evaluation, creation, and endowment effects. The formulae of the effects that can be included in the behavioral creation function c^{beh} are the same as those given for the behavioral evaluation function. However, they enter calculation of the creation function only when the actor considers increasing his behavioral score by one unit (downward steps), not when downward steps (or no change) are considered. For more details, consult Snijders et al. (2007) and Steglich et al. (2010) and replace 'going down' by 'going up'.

The statistics reported as *inc. beh.* (increase in behavior) are the sums of the changes in actor-dependent values for only those actors who increased in behavior. More precisely, it is

$$\sum_{m=1}^{M-1}\sum_{i=1}^{n} I\{z_i(t_{m+1}) > z_i(t_m)\}\,\big(s^{\text{beh}}_{ik}(x(t_{m+1})) - s^{\text{beh}}_{ik}(x(t_m))\big), \tag{48}$$

where M is the number of observations, $x(t_m)$ is the observed situation at observation m, and the indicator function $I\{A\}$ is 0 if event A is true and 0 if it is untrue.

13.2.3 Behavioral endowment function

Also the behavioral model knows the distinction between evaluation and endowment effects. The formulae of the effects that can be included in the behavioral endowment function e^{beh} are the same as those given for the behavioral evaluation function. However, they enter calculation of the endowment function only when the actor considers decreasing his behavioral score by one unit (downward steps), not when upward steps (or no change) are considered. For more details, consult Snijders et al. (2007) and Steglich et al. (2010).

The statistics reported as *dec. beh.* (decrease in behavior) are the sums of the changes in actor-dependent values for only those actors who decreased in behavior. More precisely, it is

$$\sum_{m=1}^{M-1} \sum_{i=1}^{n} I\{z_i(t_{m+1}) < z_i(t_m)\} \left(s_{ik}^{\text{beh}}(x(t_m)) - s_{ik}^{\text{beh}}(x(t_{m+1}))\right), \tag{49}$$

where M is the number of observations, $x(t_m)$ is the observed situation at observation m, and the indicator function $I\{A\}$ is 0 if event A is true and 0 if it is untrue.

13.2.4 Behavioral rate function

The behavioral rate function λ^{beh} consists of a constant term per period,

$$\lambda_i^{\text{beh}} = \rho_m^{\text{beh}}$$

for $m = 1, ..., M-1$, which can be called the basic rate; multiplied potentially by the following further effects.

1. The dependence on the position of the actor can be modeled as a function of the actor's out-degree `(outRate)`, in-degree `(inRate)`, and number of reciprocated relations `(recipRate)`, the 'reciprocated degrees'. These can be defined by

   $$x_{i+} = \sum_j x_{ij}, \qquad x_{+i} = \sum_j x_{ji}, \qquad x_{i(r)} = \sum_j x_{ij} x_{ji}$$

   (recalling that $x_{ii} = 0$ for all i).

   The contribution of the out-degrees to the rate function is a factor

   $$\exp(\alpha_h \, x_{i+}) \,,$$

   if the associated parameter is denoted α_h for some h, and similarly for the contributions of the in-degrees and the reciprocated degrees.

2. The dependence on actor covariates or behavioral variables `(RateX)` with values v_{hi} multiplies the rate by a factor

   $$\exp(\alpha_h \, v_{hi}) \,.$$

   If v_h is a behavioral variable, the non-centered value is used.

⊙ For the analysis of diffusion of innovations, which is applicable if the behavior variable is a non-decreasing variable with values 0 and 1, there are various contagion effects that render a model that would reduce to a proportional hazards model if the network were constant; see Greenan (2015a). This holds if they are part of the rate function, but not if they are included in the evaluation function. This also holds for effects depending on actor covariates. For all these effects, the rate function is multiplied by

$$\exp\left(\alpha_h\, a_{ih}(x)\right),$$

if the associated parameter is denoted α_h for some h, and the effect is $a_{ih}(x)$.

The dependence on the internal effect parameter p is similar for all these diffusion effects, and is described after the `susceptAvCovar` effect.

For models including these effects, it is recommended to use an algorithm that leads to smaller step sizes, i.e., with a value for `firstg` lower than the default of 0.2.

3. *average exposure effect* (`avExposure`), defined as the proportion of i's alters who have adopted the innovation,

$$a_{i3}(x,z) \;=\; \frac{\sum_{j=1}^{n} z_j\, x_{ij}}{\sum_{j=1}^{n} x_{ij}};$$

4. *total exposure effect* (`totExposure`), defined as the number of i's alters who have adopted the innovation,

$$a_{i4}(x,z) \;=\; \sum_{j=1}^{n} z_j\, x_{ij};$$

5. *infection by indegree effect* (`infectIn`), defined as the sum of indegrees of i's alters who are also adopters of the innovation,

$$a_{i5}(x,z) \;=\; \sum_{j=1}^{n} z_j\, x_{ij}\, x_{+j};$$

for non-directed networks this is called the *infection by degree effect* (`infectDeg`);

6. *infection by outdegree effect* (`infectOut`), defined as the sum of outdegrees of i's alters who are also adopters of the innovation,

$$a_{i6}(x,z) \;=\; \sum_{j=1}^{n} z_j\, x_{ij}\, x_{j+};$$

7. *infection by covariate effect* (`infectCovar`), defined as the sum of covariate values of i's alters who are also adopters of the innovation,

$$a_{i7}(x,z) \;=\; \sum_{j=1}^{n} z_j\, x_{ij}\, v_j;$$

8. *susceptibility to average exposure by indegree effect* (`susceptAvIn`), defined as the interaction between i's indegree and i's average exposure,

$$a_{i8}(x, z) \;=\; x_{+i}\,\frac{\sum_{j=1}^{n} z_j\, x_{ij}}{\sum_{j=1}^{n} x_{ij}};$$

for non-directed networks the equivalent effect is `totExposure`;

9. *susceptibility to average exposure by covariate effect* (`susceptAvCovar`), defined as the interaction between i's covariate value and i's average exposure,

$$a_{i9}(x, z) \;=\; v_i\,\frac{\sum_{j=1}^{n} z_j\, x_{ij}}{\sum_{j=1}^{n} x_{ij}}.$$

10. *Total In-Exposure at Distance 2* (`totInExposureDist2`):

$$a_{i10}(x, z) = \sum_{h \neq i} z_h \sum_j x_{ij} x_{hj}$$

This statistic sums, for each actor i, the number of indirect exposures to alters with attribute z_h via shared connections at distance 2 (i.e., through common alters j).

Multiple versions of averaging have been implemented:

11. *Average Total In-Exposure at Distance 2* (`avTinExposureDist2`):

$$a_{i11}(x, z) = \frac{\sum_{h \neq i} z_h \sum_j x_{ij} x_{hj}}{\sum_j x_{ij}}$$

This version averages the total in-exposure at distance 2 by the number of direct connections of actor i. If $\sum_j x_{ij} = 0$, the statistic is defined as 0.

12. *Total Average In-Exposure at Distance 2* (`totAInExposureDist2`):

$$a_{i12}(x, z) = \sum_{h \neq i} z_h \sum_j \frac{x_{ij} x_{hj}}{\sum_{h'} x_{h'j} - x_{ij}}$$

Here, for each shared alter j, the exposure is averaged over the number of other actors (excluding i) connected to j. If the denominator is 0, the term is defined as 0.

13. *Any In-Exposure at Distance 2* (`anyInExposureDist2`):

$$a_{i13}(x, z) = \sum_j \begin{cases} 1 & \text{if } x_{ij} \sum_h z_h x_{hj} > 0 \\ 0 & \text{otherwise} \end{cases}$$

This effect counts, for each alter j connected to i, whether there is at least one other actor h (with $z_h > 0$) also connected to j. This is a version of penalizing many indirect alters.

For all these diffusion effects there is an additional dependence on the internal effect parameter p; the formulae given above are the default, valid for $p = 0$.

The parameter p must be integer-valued. If $p > 0$, the contribution is changed to $a_{i13}(x, z) = 0$ if less than p alters have adopted the innovation, i.e., if

$$\sum_{j=1}^{n} z_j \, x_{ij} \; < \; p \; .$$

E.g., if $p = 3$, then there is influence on actor i only if this actor has at least three alters who have adopted the innovation.

For the effects `avExposure`, `avTinExposureDist2`, `totExposure`, `totInExposureDist2`, `totAInExposureDist2`, `susceptAvIn`, and `susceptAvCovar`,
if $p < 0$, the same rule is applied with the minimum $|p|$, but if there are more than $|p|$ alters who have adopted the innovation, only the value $|p|$ is used. This means that for $p < 0$, $\sum_{j=1}^{n} z_j \, x_{ij}$ in the formula for $a_{ik}(x, z)$ is replaced by

$$\begin{cases} 0 & \text{if } \sum_{j=1}^{n} z_j \, x_{ij} < |p| \\ |p| & \text{if } \sum_{j=1}^{n} z_j \, x_{ij} \geq |p| \; . \end{cases}$$

13.3 Effects for estimation by Generalized Method of Moments

A variety of effects can be specified with `type = "gmm"`, as shown in effectsDocumentation. These are not effects for the definition of the probability model, but for the estimation by the Generalized Method of Moments. This is documented in Section 7.10.

14 Parameter interpretation

The main 'driving force' of the actor-oriented model is the evaluation function (extended with the creation and/or endowment function, if these are part of the model specification). For the network, this is given in formula (15) as

$$f_i^{\mathrm{net}}(x) = \sum_k \beta_k^{\mathrm{net}} s_{ik}^{\mathrm{net}}(x) .$$

The evaluation function can be regarded as the 'attractiveness" of the network (or behavior, respectively) for a given actor. For getting a feeling of what are small and large values, is is helpful to note that the evaluation functions are used to compare how attractive various different tie changes are, and for this purpose random disturbances are added to the values of the evaluation function with standard deviations equal[21] to 1.28.

An alternative interpretation is that when actor i is making a 'ministep', i.e., a single change in his outgoing ties (where no change also is an option), and x_a and x_b are two possible results of this ministep, then $f_i^{\mathrm{net}}(x_b) - f_i^{\mathrm{net}}(x_a)$ is the log odds ratio for choosing between these two alternatives – so that the ratio of the probability of x_b and x_a as next states is

$$\exp\left(f_i^{\mathrm{net}}(x_b) - f_i^{\mathrm{net}}(x_a)\right) .$$

Note that, when the current state is x, the possibilities for x_a and x_b are x itself (no change), or x with one extra outgoing tie from i, or x with one fewer outgoing tie from i. Explanations about log odds ratios can be found in texts about logistic regression and loglinear models. For dependent behavior variables, the interpretation is similar, keeping in mind that permitted changes in the behavior variable are –1, 0, and +1 (as far as these changes do not lead beyond the permitted range).

If one wishes an extremely short description of what the parameters of the objective function stand for, one could write 'non-standardized contributions to log-probabilities' as a shorthand for 'contributions to log-probabilities of increasing the dependent variable by 1 unit when the effect is increased by 1 unit'. Here the dependent variable is either the tie variable X_{ij} or the behavior variable Z_i; the effect is $s_{ik}(x)$.

14.1 Networks

The evaluation function is a weighted sum of 'effects' $s_{ik}^{\mathrm{net}}(x)$. Their formulae can be found in Section 13.1.1. These formulae, however, are defined as a function of the whole network x, and in most cases the contribution of a single tie variable x_{ij} is just a simple component of this formula. The contribution to $s_{ik}^{\mathrm{net}}(x)$ of adding the tie $i \to h$ minus the contribution of adding the tie $i \to j$ is the log probability ratio comparing the probabilities of i sending a new tie to h versus sending the tie to j, if all other effects $s_{ik}^{\mathrm{net}}(x)$ yield the same values for these two hypothetical new configurations.

For example, suppose that actors j and h, actual or potential relation partners of actor i, have exactly the same network position and the same values on all variables included in the model,

[21] More exactly, the value is $\sqrt{\pi^2/6}$, the standard deviation of the Gumbel distribution; see Snijders (2001).

except that for some actor variable V for which only the popularity (alter) effect is included in the model, actor h is one unit higher than actor j: $v_h = v_j + 1$. It can be seen in Section 13.1.1 that the popularity (alter) effect is defined as

$$s_{ik}^{\text{net}}(x) \;=\; \sum_j x_{ij}\, v_j \;.$$

The contribution to this formula made by a single tie variable, i.e., the difference made by filling in $x_{ij} = 1$ or $x_{ij} = 0$ in this formula, is just v_j. Let us denote the weight of the V-alter effect by β_k. Then, the difference between extending a tie to h or to j that follows from the V-alter effect is $\beta_k \times (v_h - v_j) = \beta_k \times 1 = \beta_k$.

Thus, in this situation, β_k is the log probability ratio of the probability that h is chosen compared to the probability that j is chosen. E.g., if i currently has a tie neither to j nor to h, and supposing that $\beta_k = 0.3$, the probability for i to extend a new tie to h is $e^{0.3} = 1.35$ times as high as the probability for i to extend a new tie to j.

14.2 Behavior

The evaluation function for behavior is given by

$$f_i^{\text{beh}}(x, z) \;=\; \sum_k \beta_k^{\text{beh}} s_{ik}^{\text{beh}}(x, z) \;,$$

see (33). In many cases[22] the effect has the form of a product

$$s_{ik}^{\text{beh}}(x, z) \;=\; z_i\, s_{ik}^{0}(x, z) \;, \tag{50}$$

where $s_{ik}^0(x, z)$ is not dependent on z_i (although it might depend on z_j for other actors j), and therefore would not be affected by the outcome of a behavior ministep of actor i. Examples are the main effect of an actor attribute, but also the average alter effect. For such effects, when a ministep in behavior occurs, the contribution on the probability distribution of the change is as follows: a change of z_i by –1 will decrease the evaluation function by $\beta_k^{\text{beh}} s_{ik}^0(x, z)$, and a change by +1 will increase it by the same amount. (Note that this amount does not depend on the value of z_i because of the mentioned condition.) Therefore, the log probability-ratio of an increase in behavior compared to staying constant that can be attributed to a difference of +1 in the value of the predictor function $s_{ik}^0(x, z)$, is equal to β_k^{beh}. The probability ratio is $\exp(\beta_k^{\text{beh}})$.

For example, later in this section results are presented where, for an analysis of drinking behavior, the estimated parameter for average alter is 1.1414. This means that when comparing two individuals who are equal in all respects except that the friends of the first on average are 1 higher on the drinking scale than those of the second individual, the odds of increasing drinking behavior compared to no change (in the event of a ministep with respect to drinking behavior) are $\exp(1.1414) = 3.1$ times higher for the first individual than for the second.

The average similarity effect is defined by

$$s_{i13}^{\text{beh}}(x, z) = x_{i+}^{-1} \sum_j x_{ij} \big(\text{sim}_{ij}^{z} - \widehat{\text{sim}^{z}}\big)$$

[22] For effects satisfying this condition, the `interactionType` is defined as `"OK"`.

(see Section 13.2.1), where sim_{ij}^{z} is given in (1) in Section 5.2. (If the ratio is 0/0, the effect is defined as 0.) Subtracting $\widehat{\mathrm{sim}^{z}}$ is done just for the purpose of centering. Since the underlying measure sim_{ij}^{z} ranges from 0 (if z_i and z_j are the two different extreme values of Z) to 1 (if $z_i = z_j$), this effect has a range of 1 independently of the scaling of Z. Therefore the average similarity effect, and also the total similarity effect, differ from the total and average alter effects by the fact that the latter are dependent on the scaling of Z, whereas the former are not directly dependent on this scaling. This means that parameters for average similarity effects can be regarded as being on the same scale for different variables, but parameters for average alter are not.

The interpretations of total and average similarity are more laborious to explain than the interpretation of the average alter effect. This is because total and average similarity are not of the form (50). To explain the log-probabilities or probability ratios due to these effects, it has to be understood how a change in the behavior z_i will affect the values of these effects. Examining their formulae leads to the following.

For a given actor i, the out-degree (number of friends) is denoted x_{i+}. Let the number of friends whose values z_j are less than, equal to, or greater than the value z_i of i, be denoted by a, b, and c. Denote the range (maximum minus minimum value) of the behavior by r. Then, in the event of a ministep with respect to behavior, the contributions of the total similarity effect to the log-probabilities of changes –1, 0, or +1, are given by $\beta_k^{\mathrm{beh}}(a-b-c)/r$, 0, and $\beta_k^{\mathrm{beh}}(c-a-b)/r$, respectively. The contributions for the average similarity effect are $\beta_k^{\mathrm{beh}}(a-b-c)/(rx_{i+})$, 0, and $\beta_k^{\mathrm{beh}}(c-a-b)/(rx_{i+})$. This shows that the influence of the friends in the similarity effects depends only on whether they have larger or smaller values than the focal actor, not on how much larger the values are. It also shows that for the similarity effects the dispersion of the friends' values matters and not only their average, whereas for the average alter effect only the average matters.

To have a compact formulation, without all this detail, one could say the following. We use the example on one of the following pages, where an average similarity effect on drinking is reported of $\hat{\beta}_k^{\mathrm{beh}} = 3.9689$, where drinking has a range of $r = 5 - 1 = 4$. For an individual all of whose friends drink more than this individual does, the contribution of friends' influence to the probability ratio of an increase in drinking as compared to no change is a factor $\exp(3.9689/4) = 2.7$. (In this formulation, the condition 'in the event of a ministep with respect to drinking behavior" is left implicit.)

In the same situation, if hypothetically a total average similarity effect were found of 0.82, then one could say that having *one* additional friend who drinks more than oneself increases the probability ratio of an increase in drinking as compared to no change by a factor $\exp(0.82/4) = 1.23$. In general, parameters for the total similarity effect will tend to be smaller than those for the average similarity effect, because the former refer to comparisons about a single friend, and the latter to comparisons about all friends.

14.3 Ego – alter selection tables

When some variable V occurs in several effects in the model, then its effects can best be understood by considering all these effects simultaneously. This section treats this issue for the case that V is an actor variable with more than two values (for dichotomous variables the situation is

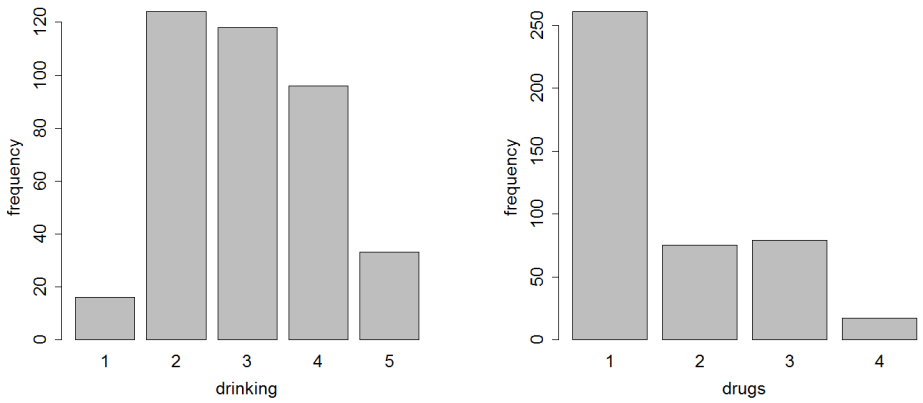


Figure 5: Frequencies of drinking and drug use over 3 waves combined.

simpler).

As an example, consider the Glasgow data set from the Teenage Friends and Lifestyle Study of West et al. (Michell and West, 1996; Pearson and West, 2003; Steglich et al., 2010). We refer to any of these papers for a further description of the data. The data is used for the 129 students who were present at all three measurement times; this data set is available from the SIENA website. We treat friendship as the dependent network, drinking as the dependent behaviour variable, and cannabis use ('drugs') as an actor covariate. Drinking has categories 1–5, coded as 1 (none), 2 (once or twice a year), 3 (once a month), 4 (once a week), and 5 (more than once a week); cannabis use is coded 1 (none), 2 (tried once), 3 (occasional) and 4 (regular). The frequencies (over the three waves combined) of these substance use variables are given in Figure 5.

For both actor variables, in the network part we use the four-parameter specification of Snijders and Lomi (2019), which is a combination of the ego minus alter squared, alter squared, ego, and alter effects.

The formulae in Section 13.1.1 imply that the components in the network evaluation function corresponding to the effects of variable V are

$$\beta_{\text{sq. diff}} \sum_j x_{ij}\,(v_j - v_i)^2 + \beta_{\text{ego}}\,(v_i - \bar{v})\,x_{i+} + \beta_{\text{alter}} \sum_j x_{ij}\,(v_j - \bar{v}) + \beta_{\text{sq. alter}} \sum_j x_{ij}\,(v_j - \bar{v})^2\,. \tag{51}$$

The contribution to this formula of the single tie variable x_{ij} — i.e., the difference between the values of (51) for $x_{ij} = 1$ and $x_{ij} = 0$ — is equal to

$$\beta_{\text{sq. diff}}(v_j - v_i)^2 \;+\; \beta_{\text{ego}}\,(v_i - \bar{v}) \;+\; \beta_{\text{alter}}\,(v_j - \bar{v}) \;+\; \beta_{\text{sq. alter}}\,(v_j - \bar{v})^2\,. \tag{52}$$

It should be noted that by default all variables are internally centered by RSiena. The mean values used for the centering are given near the beginning of the file produced by write_report and are given by the "mean" attributes of the covariates, as explained in Section 5.2.2. This is

made explicit in the formulae by the subtraction of the mean $\bar{v}$. From this equation (52) a table, or a plot, can be made that gives these contributions for some values of v_i and v_j.

From the output of write_report for these data it can be seen that the alcohol use variable assumes values from 1 to 5, with overall mean[23] equal to $\bar{v} = 2.99$. Drug use is a changing actor variable, with range 1–4, and mean $\bar{v} = 1.5$.

Suppose that we fit a model of network-behavior co-evolution to this data set with for the network evolution the effects of outdegree, reciprocity, geometrically weighted edgewise shared partners ('gwesp', in the `gwespFF` version, representing transitivity), indegree popularity, outdegree activity, outdegree popularity, and reciprocal degree activity; ego, alter, and same effect of sex; for drinking and drug use the four effects in (51); and for the behavior (i.e., alcohol drinking) dynamics the effects linear shape, quadratic shape (effect of drinking on itself), and average alter.

The results obtained are as follows.

Effect	par.	(s.e.)
Network Dynamics		
constant friendship rate (period 1)	12.643	(1.595)
constant friendship rate (period 2)	10.201	(1.024)
outdegree (density)	–2.723	(0.304)
reciprocity	3.387	(0.244)
gwesp (transitivity)	1.847	(0.088)
indegree – popularity	–0.030	(0.025)
outdegree – popularity	–0.107	(0.048)
outdegree – activity	0.075	(0.037)
reciprocal degree – activity	–0.287	(0.051)
sex alter	–0.120	(0.096)
sex ego	0.054	(0.108)
same sex	0.664	(0.089)
drinking alter	0.024	(0.082)
drinking squared alter	–0.113	(0.091)
drinking ego	0.104	(0.088)
drinking (alter-ego) squared	–0.117	(0.056)
drugs alter	–0.137	(0.091)
drugs squared alter	0.164	(0.067)
drugs ego	–0.123	(0.068)
drugs (alter-ego) squared	–0.051	(0.025)
Behaviour Dynamics		
rate drinking (period 1)	1.598	(0.302)
rate drinking (period 2)	2.431	(0.508)
drinking linear shape	0.452	(0.138)
drinking quadratic shape	–0.609	(0.181)
drinking average alter	1.281	(0.488)

convergence t ratios all < 0.04.
Overall maximum convergence ratio 0.15.

We interpret here the parameter estimates for the effects of drinking behavior and drug use without

[23] The mean of the three means per wave.

being concerned with the significance, or lack thereof. For the drinking behavior, formula (52) yields

$$-0.117\,(v_j - v_i)^2 \;+\; 0.104\,(v_i - \bar{v}) \;+\; 0.024\,(v_j - \bar{v}) \;-\; 0.113\,(v_j - \bar{v})^2\,.$$

These values can be calculated for v_i and v_j ranging from 1 to 5, with $\bar{v} = 2.99$. This is called the *selection table*. It can be calculated by the RSiena function interpret_selection.

Given that the data set is called `G129_Data`, the answer (sienaFit) object is called `ans5`, the network `"friendship"`, and the actor variable `"drinking"`, the following command can be used.

```
st_dr <- interpret_selection(ans5, G129_Data, "friendship", "drinking")
# It can be displayed
st_dr
# and if package xtable is loaded, also be written
# to a latex or html file. For example,
tab.drink <- xtable(st_dr)
print(tab.drink,file = "tab_drink.htm", type = "html",
             html.table.attributes = "rules = none")
# The html.table.attributes option gives the <table> tag
# used in the html file.
# and optional
xtable(st_dr) # for a LaTeX table
```

The table is (rounded to 2 decimals)

$v_i \;\backslash\; v_j$	1	2	3	4	5
1	–0.70	–0.46	–0.68	–1.35	–2.49
2	–0.72	–0.24	–0.22	–0.66	–1.57
3	–0.96	–0.25	0.00	–0.21	–0.88
4	–1.45	–0.50	–0.01	0.01	–0.42
5	–2.17	–0.98	–0.26	0.00	–0.20

Table 4: Selection table for friendship with respect to drinking.

It is instructive to make a plot of this table. This can be done using package autograph (Hollway, 2026).

This produces Figure 6 (colors to be improved).

From the table it can be concluded that there is a significant tendency toward homophily (negative effect of ego minus alter squared).

Interpreting the results along the lines of Snijders and Lomi (2019), there is a significant tendency to homophily (negative effect of ego minus alter squared), and a non-significant tendency to conformity (negative effect of alter squared), where the estimated social norm is $\bar{v} + \hat{\beta}_{\mathrm{altX}}/(2\,\hat{\beta}_{\mathrm{altSqX}}) = 2.99 + 0.024/(2 \times 0.113) = 3.1$, quite a middle value. We see that the location of the maximum of the curves increases with ego's value v_i, but for low v_i the maximum is larger than v_i, and for high v_i it is less than v_i: homophily is combined with a kind of 'regression to the mean'.

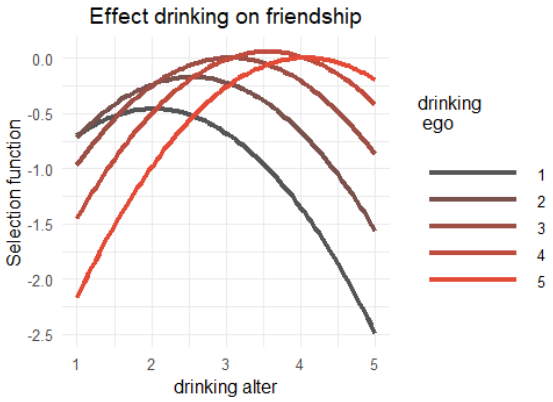


Figure 6: Plot of selection table for friendship with respect to drinking.

For drug use, formula (52) yields

$$-0.051\,(v_j - v_i)^2 \;+\; -0.123\,(v_i - \bar{v}) \;-\; 0.137\,(v_j - \bar{v}) \;+\; 0.164\,(v_j - \bar{v})^2\,,$$

which leads to the following table.

$v_i \setminus v_j$	1	2	3	4
1	0.16	–0.02	0.03	0.30
2	–0.02	–0.09	0.06	0.43
3	–0.30	–0.27	–0.01	0.47
4	–0.68	–0.54	–0.19	0.39

Table 5: Selection table for friendship with respect to drug use.

The corresponding figure is in Figure 7. Here we see a totally different pattern. Since the sum of the alter squared and the difference squared effects is positive, the parabolas have a minimum rather than a maximum. Pay attention to the vertical scale: in Figure 6 the difference between the highest and lowest values is about 2.5, in Figure 7 it is about 1.2, half as much. This signifies that overall, drug use has smaller effects on friendship than smoking. For ego's who do not use cannabis ($v_i = 1$), drug use of alters hardly matters; egos who use cannabis regularly ($v_i = 4$) have a somewhat lower attraction to alters with lower cannabis use. For this actor variable, the specification with the effects of ego, alter, and the product interaction ego $\times$ alter might fit just as well, or perhaps better.

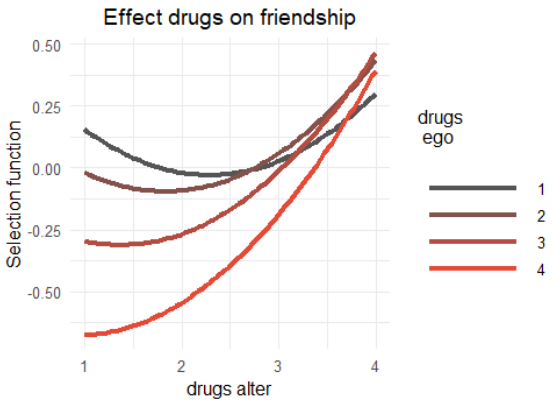


Figure 7: Plot of selection table for friendship with respect to drug use.

14.4 Ego – alter influence tables

In quite a similar way as in the preceding section, from the parameter estimates as presented in the output tables, combined with the formulae for the effects, we can construct tables indicating how likely are changes to various different values of the behavior, depending on the behavior of the actor's friends. The functions used to define the effects can be found in Section 13.2.1, and it must not be forgotten that all covariates are by default internally centered in RSiena, and the subtracted means are reported in the initial output produced by write_report, but for the covariates they also can be obtained by requesting the 'mean' attributes of the covariates, see Section 5.2.2.

14.4.1 Average alter

In the model from the preceding section, the estimated coefficients in the behavior evaluation function are as follows.

Effect	par.	(s.e.)
Behaviour Dynamics		
rate drinking (period 1)	1.598	(0.302)
rate drinking (period 2)	2.431	(0.508)
drinking linear shape	0.452	(0.138)
drinking quadratic shape	–0.609	(0.181)
drinking average alter	1.281	(0.488)

The dependent behavior variable now is indicated Z. (In the preceding section the letter V was used, but this referred to any actor variable predicting network dynamics, irrespective of whether it was also a dependent behavior variable.)

The formulae in Section 13.2.1 show that the evaluation function for this model specification is

$$f_i^{\text{beh}} \;=\; \beta_{\text{linear}}\,(z_i - \bar{z}) \;+\; \beta_{\text{quad}}\,(z_i - \bar{z})^2 \;+\; \beta_{\text{avAlt}}\,(z_i - \bar{z})(\bar{z}_{(i)} - \bar{z})\;, \tag{53}$$

where $\bar{z}_{(i)}$ is the average Z value of i's friends,

$$\bar{z}_{(i)} = \frac{1}{x_{i+}} \sum_j x_{ij}\, z_j\;,$$

and 0 if this formula would be 0/0.

The *Influence Table* is defined as the matrix with in the rows the average of Z for i's friends, in the columns the possible values of z_i, and in the cells the evaluation function (53). For the table above, the evaluation function is

$$\hat{f}_i^{\text{beh}} \;=\; 0.452\,(z_i - \bar{z}) \;-\; 0.609\,(z_i - \bar{z})^2 \;+\; 1.281\,(z_i - \bar{z})(\bar{z}_{(i)} - \bar{z})\;.$$

The influence table can be produced by the function `interpret_influence`, which can be plotted by package autograph (Hollway, 2026).

The influence table for drinking in this example is as follows.

$\bar{z}_{(i)}$ \ z_i	1	2	3	4	5
1	1.76	1.48	–0.02	–2.74	–6.67
2	–0.79	0.21	–0.01	–1.44	–4.10
3	–3.34	–1.06	0.00	–0.15	–1.53
4	–5.89	–2.33	0.02	1.14	1.05
5	–8.44	–3.59	0.03	2.43	3.62

The interpretation[24] is that each row corresponds to a given average behavior of the focal actor's friends; comparing the different values in the row shows the relative 'attractiveness' of the different potential values of ego's own behavior. The maximum in each row is assumed at the diagonal. This means that for each value for the average friends' behavior $\bar{z}_{(i)}$, the focal actor 'prefers' to have the same behavior as all these friends. The differences in the bottom rows are larger than in the top rows, indicating that in the case where the friends who do not drink at all, the 'preference' (or social pressure) toward imitating their behavior is somewhat less strong than in the case where all the friends drink a lot.

A figure of this table can be produced by package autograph, and is presented in Figure 8.

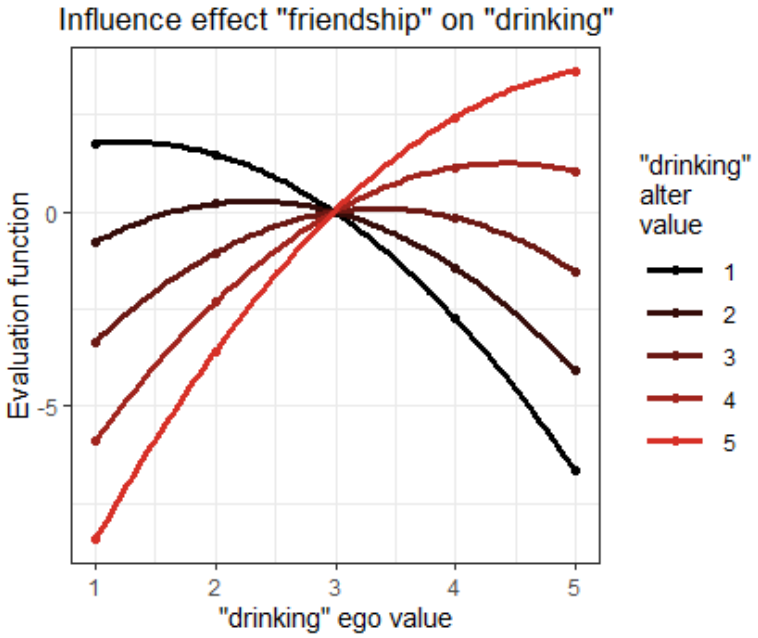


Figure 8: Plot of influence table for friendship with respect to drinking.

The five plotted curves intersect at the same point. This is indeed implied by formula (53), which is 0 if $z_i = \bar{z} = 2.99$.

Another way to look at the behavior evaluation function is to consider mathematically the

[24]The interpretation is given here in terms of attraction and preferences; this is for the sake of having a straightforward and easy way of expressing the interpretation, which strictly should be only about probabilities.

location of its maximum. This function here can be written also as

$$f_i^{\text{beh}} \;=\; \big(0.452 + 1.281(\bar{z}_{(i)} - \bar{z})\big)\,(z_i - \bar{z}) \;-\; 0.609\,(z_i - \bar{z})^2\,.$$

Differentiating with respect to z_i shows that this function is maximal for

$$z_i = \bar{z} \;+\; \frac{0.452 + 1.281(\bar{z}_{(i)} - \bar{z})}{2 \times 0.609} \;=\; 0.22 \;+\; 1.05\,\bar{z}_{(i)}\,,$$

just a little bit larger than $\bar{z}_{(i)}$. Indeed in the table and in the plot we see that, if $\bar{z}_{(i)}$ has integer values 1, 2, 3, 4, or 5, the highest values are obtained exactly for $z_i = \bar{z}_{(i)}$.

14.4.2 Average similarity

The formulae in Section 13.2.1 show that when average similarity is used as the specification of social influence, the evaluation function is given by

$$f_i^{\text{beh}}(z_i) \;=\; \beta_{\text{trend}}\,(z_i - \bar{z}) + \beta_{\text{drink}}\,(z_i - \bar{z})^2 + \beta_{\text{av. sim}}\,\frac{1}{x_{i+}}\,\sum_j x_{ij}\big(\text{sim}(z_i, z_j) - \widehat{\text{sim}^z}\big) \quad (54)$$

where the similarity score is

$$\text{sim}(z_i, z_j) \;=\; 1 \;-\; \frac{|z_i - z_j|}{\Delta}$$

in which Δ is the range (maximum minus minimum) of Z, while $\widehat{\text{sim}^z}$ is the mean of the similarity scores. The latter is just a constant that does not have any influence on the change probabilities of the behavior. Therefore we use the equivalent form

$$f_i^{\text{beh}}(z_i) \;=\; \beta_{\text{trend}}\,(z_i - \bar{z}) + \beta_{\text{drink}}\,(z_i - \bar{z})^2 + \beta_{\text{av. sim}}\,\frac{1}{x_{i+}}\,\sum_j x_{ij}\left(\frac{|\bar{z} - z_j| \;-\; |z_i - z_j|}{\Delta}\right), \quad (55)$$

which has the advantage that it is zero for $z_i = \bar{z}$, and therefore comparable across i.

With this specification for the effect of drinking (and a modified specification of the effect of drug use), the estimation results are as follows.

Effect	par.	(s.e.)
Network Dynamics		
constant friendship rate (period 1)	12.769	(1.331)
constant friendship rate (period 2)	10.262	(0.987)
outdegree (density)	–2.708	(0.320)
reciprocity	3.376	(0.247)
gwesp (transitivity)	1.837	(0.084)
indegree – popularity	–0.028	(0.027)
outdegree – popularity	–0.108	(0.053)
outdegree – activity	0.075	(0.037)
reciprocal degree – activity	–0.285	(0.052)
sex alter	–0.123	(0.092)
sex ego	0.051	(0.107)
same sex	0.661	(0.087)
drinking alter	0.037	(0.075)
drinking squared alter	–0.108	(0.095)
drinking ego	0.097	(0.091)
drinking squared ego	0.010	(0.098)
drinking (ego – alter) squared	–0.108	(0.052)
drugs alter	–0.020	(0.054)
drugs ego	–0.183	(0.066)
drugs ego × drugs alter	0.126	(0.052)
Behaviour Dynamics		
rate drinking (period 1)	1.696	(0.326)
rate drinking (period 2)	2.659	(0.565)
drinking linear shape	0.461	(0.136)
drinking quadratic shape	–0.003	(0.088)
drinking average similarity	6.190	(1.810)

convergence t ratios all < 0.08.

Overall maximum convergence ratio 0.17.

Equation (55) is less simple than equation (53), because (53) is a quadratic function of z_i, with coefficients depending on the Z values of i's friends as a function of their average, whereas (55) depends on the entire distribution of the Z values of i's friends.

Suppose that, in model (55), the similarity coefficient $\beta_{\text{av. sim}}$ is positive, and compare two focal actors, i_1 all of whose friends have $z_j = 3$ and i_2 who has four friends, two of whom with $z_j = 2$ and the other two with $z_j = 4$. Both actors are then drawn toward the preferred value of 3; but the difference between drinking behavior 3 on one hand and 2 and 4 on the other hand will be larger for i_1 than for i_2. In model (53), on the other hand, since the average is the same, both actors would be drawn equally strongly toward the average value of 3.

Since the objective function for model (55) depends on all behavior values of the actor's friends, not just on their average, here we present a table only for the special case of actors all whose friends have the same behavior z_j. For the parameters given above, the behavior evaluation function then reads

$$f_i^{\text{beh}}(z_i) \;=\; 0.461\,(z_i - \bar{z}) \;-\; 0.003\,(z_i - \bar{z})^2 \;+\; 6.190\left(\frac{|\bar{z} - z_j| \;-\; |z_i - z_j|}{\Delta}\right).$$

Function interpret_influence produces the following table, which again can be plotted by autograph.

z_j \ z_i	1	2	3	4	5
1	0.80	–0.28	–1.36	–2.45	–3.55
2	–0.75	1.27	0.18	–0.91	–2.00
3	–2.30	–0.28	1.73	0.64	–0.45
4	–3.84	–1.83	0.18	2.19	1.10
5	–5.39	–3.37	–1.36	0.64	2.64

The interpretation of this table is that each row corresponds to a given common behavior of the focal actor's friends; comparing the different values in the row shows the relative 'attractiveness' of the different potential values of ego's own behavior. The maximum in each row is assumed at the diagonal. This means that for each value for the common friends' behavior z_j, the focal actor prefers to have the same behavior as these friends. The differences in the bottom rows are larger than in the top rows, indicating that in the case where the friends who do not drink at all, the preference (or social pressure) toward imitating their behavior is less strong than in the case where all the friends drink a lot.

The values far away from the maximum contrast in this case less strongly than in the case of the model with the average alter effect, which is a consequence of the use of the absolute rather than squared deviation. However, neither of these models fits clearly better than the other. The fact that the t-ratio for testing the social influence effect is larger for average similarity (6.190/1.810 vs. 1.281/0.488) suggest that this specification has a slightly better fit.

These tables present only the contribution of some of the terms of the objective function, and the behavior dynamics will of course be compounded if the objective function contains more effects.

Figure 9 gives the same information as the table.

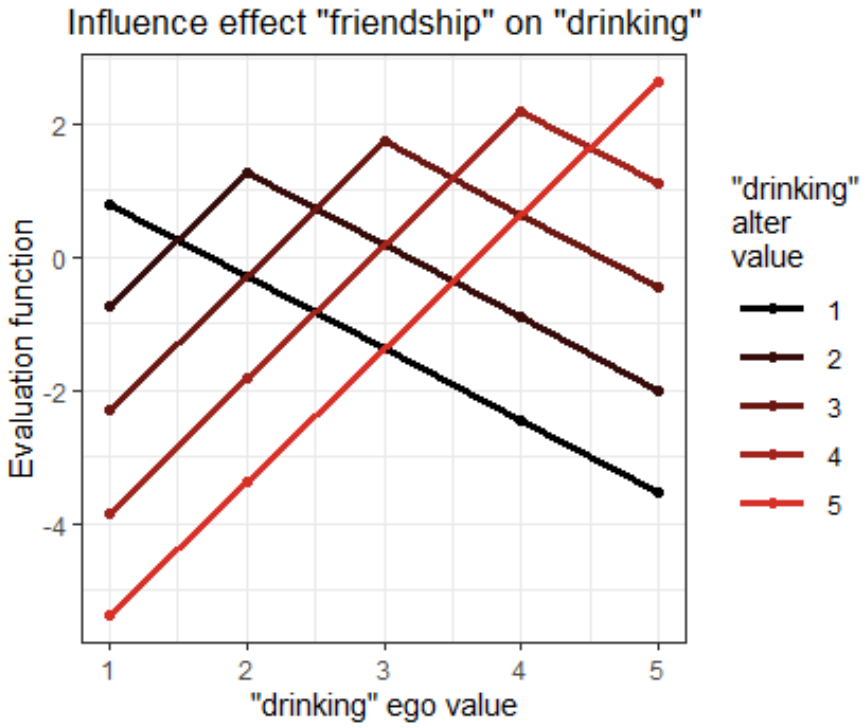


Figure 9: Plot of influence table for friendship with respect to drinking, for specification by average similarity.

14.5 Effect sizes and measures of fit

For linear regression models we are used to the coefficient of determination, usually denoted R^2, defined as the proportion of variance that is explained by the model. Such a measure is not easily available for most other statistical models; linear regression is special in allowing such a well-interpretable measure of 'fit'. The following measures that have something of the same purpose are available in RSiena through function interpret_size.

Some further material is at
https://www.stats.ox.ac.uk/~snijders/siena/EffectSizes_s.pdf.

There is a script at
https://www.stats.ox.ac.uk/~snijders/siena/vdB_effsize.R.

14.5.1 Entropy / relative certainty

The uncertainty in a random categorical variable can be measured by the *entropy* (Shannon, 1948). For a probability distribution with probabilities $p = (p_1, \ldots, p_c)$ for an outcome with categories $1, \ldots, c$, it is defined as

$$H(p) \;=\; -\sum_{k=1}^{c} p_k \, {}^2\log(p_k) \;. \tag{56}$$

This is 0 if the variable is deterministic, i.e., one of the categories has probability 1; the maximum value is ${}^2\log(c)$, attained if all categories have the same probability $1/c$ (maximum uncertainty).

Using a minus sign, the degree of certainty (as opposed to uncertainty) in the outcome of the ministep with probabilities $p_i(\beta, x) = \big(p_{i1}(\beta, x), \ldots, p_{in}(\beta, x)\big)$ can be expressed by

$$R_H(i, \beta, x) \;=\; 1 - \frac{H(p_i(\beta, x))}{{}^2\log(n)} \;. \tag{57}$$

For models with a constant rate function, this can be averaged over actors to obtain a measure for a ministep taken by a random actor:

$$R_H(\beta, x) \;=\; \frac{1}{n} \sum_i R_H(i, \beta, x) \;. \tag{58}$$

This is a measure between 0 and 1. It was proposed in Snijders (2004), where further discussion may be found.

The values of the degree of certainty will generally be low, because which tie to create or to drop normally will have a very large degree of 'chance variability'. They are calculated by interpret_size as `RHActors` (see the help page). The mean over actors is printed by the print method for interpret_size.

Note that the values given by interpret_size refer to each of the periods, including the first wave; for this purpose, x in (58) is taken as the observed network (and behaviour...) at the start of each period.

14.5.2 Standard deviation of change statistics

The *change statistics* $\delta_{ijk}(x)$ for an effect $s_{ik}(x)$ are the changes in the value of this effect, for a given actor i, a given current state of the network x, and a given alter j, when the tie variable x_{ij} would be toggled (and analogous for behavioral dependent variables). These are the basic 'variables' in the conditional multinomial logistic regression model that is at the heart of the Stochastic Actor-Oriented Model. Their *within-actor* standard deviations $\sigma_k(x)$ can be used to turn the parameters β_k into comparable quantities,

$$\sigma_k(x)\,\beta_k\ . \tag{59}$$

These express the parameters β_k, for different k and a common x, on a common scale.

These standard deviations are computed by interpret_size as `sigma` and printed by the print method for interpret_size if `printSigma = TRUE`. The change statistics themselves are potentially a lot of numbers ($n^2 \times M \times K$, where M is the number of waves and K the number of effects in the model), and are only retained in the object produced by interpret_size if `getChangeStats = TRUE`; see the help page for interpret_size.

14.5.3 Relative importance of effects

The measure for *relative importance of effects* in Stochastic Actor-Oriented Models, proposed by Indlekofer and Brandes (2013), together with a measure for the (non-relative) *importance of effects*, can be calculated by interpret_size.

Ministeps in the Stochastic Actor-Oriented Model are random events with c possible outcomes; for one-mode networks $c = n$, for two-mode networks c is the number of second-mode nodes +1 (for 'no change'), for behavior $c = 3$ (changes by –1, 0, or +1); there may be constraints diminishing the number of choices. Let us denote the probabilities with which actor i in a ministep may change one of the outgoing ties by $p_i(\beta, x) = \big(p_{i1}(\beta, x), \ldots, p_{ic}(\beta, x)\big)$. The Indlekofer-Brandes measures are numerical answers to the question, 'How strongly do these probabilities depend on each of the effects?'. The answer in this paper operates by replacing, in turn, each parameter value β_k by 0, and then considering the differences between the probability vectors.

In their paper, measures are proposed for expressing this per actor per wave; these measures can be relative (so that the total importance for each actor always is 1), or raw. In the current implementation, only the relative importances for the waves at the start of each period are calculated as the first ministep after the last observation is never estimated by interpret_size. The relative importances are calculated by interpret_size as `RIActors`, the raw importances as `IActors` (see the help page). The relative importances can be averaged over actors to give `expectedRI`, which is printed by the print method for interpret_size.

A weak point of this measure for importance of effects, or relative importance, is the difficulty of grasping the meaning of the model defined by probabilities where some parameter values are replaced by 0. Setting one of the parameters to zero, while leaving the others unaffected, leads to model specifications that are hard to interpret and often are misleading. For example, for a model including covariates, it is easy to see that the relative importances depend on whether the covariates are centered. More generally, simply setting a parameter to zero affects the variance of the values of the objective function in any ministep decision.

15 Error messages

This chapter contains some error messages with their explanations. Currently it is not very extensive; new error messages will be added as answerable questions about them arise. For understanding errors, general advice is to read in this manual, read the help pages for functions, and look at the News page of the Siena website.

For diagnosing errors in R in general, a useful function is

```
traceback()
```

This is to be called after an error was generated. It will give information, usually very helpful, about where exactly the error was caused. For example, it will give you the function generating the error message, with recursive information.

Note that it is not difficult to find the source of error messages in the code. The easiest way is searching for the text of the error message by some search machine such as google. You will probably find it points to GitHub or another repository of the code. There you can see what led to the error message, if you can understand some R. An alternative, if you know the function generating the error message (which you can find out by `traceback()`, see above) is to search directly in the code for this function. If the function is called, e.g., siena, you can write the code to a file called `listing.txt` by the commands

```
sink("listing.txt")
siena
sink()
```

Note that you should give the function name without parentheses. When the function is not an exported function, e.g., `robmon`, you can request it via

```
sink("listing_robmon.txt")
RSiena:::robmon
sink()
```

15.1 After updating

Updates of RSiena are not always completely backward compatible. When updating the version, you may have to create effects objects and algorithm objects again. If there is incomplete backward compatibility with respect to useability of scripts, this will be mentioned at the *News page* of the SIENA website.

Creating a new effect object given that you have an earlier effect object `old.effects` for an older RSiena version can become easier using the function update_specification, with commands such as the following.

```
eff0 <- make_specification(data_object)
new.effects <- update_specification(eff0, old.effects)
```

See the help file for update_specification; the RSiena version used to create `old.effects` does not matter.

15.2 During estimation

♢ *Unlikely to terminate this epoch: more than 1000000 steps.*

This can happen in function siena; in conditional estimation (`cond = FALSE` in set_algorithm_saom), when the rate parameter provisionally has hit a value such that the desired number of changes will probably never be reached; or in non-conditional estimation when the number of ministeps has become too large before arriving at the time for the next wave. See Section 7.12.1.

Potential solutions.

1. Check whether your model specification is reasonable; for example, there might be doubts about the specification of the rate function.

2. Use non-conditional estimation (Section 7.12.1) if you were not already doing so.

3. If the model includes the `outRate` effect, consider replacing it by the `outRateLog` effect; but note that this works properly only for non-conditional estimation (`cond = FALSE` in set_algorithm_saom).

4. If you are working with a data set with more than two waves, there might be unmodeled heterogeneity between the periods. Try modeling the periods separately.

5. Set the parameter `firstg` in set_algorithm_saom to a lower value than the default 0.2. This will make the algorithm move more slowly, hopefully avoiding the problematic region in the parameter space.

   The advice would be to set `firstg` to 0.02 and expect the necessity to do a second estimation using the `prevAns` parameter in siena. If the problem still occurs for `firstg` = 0.02, use a smaller value (but less than 0.001 probably makes no sense). `firstg` determines the step sizes in the stochastic approximation algorithm; it is mentioned in some places earlier in this manual. Especially for models with additional rate effects the default value of 0.2 might be too large. `firstg` is the initial value of parameter a_N mentioned on p. 393 of Snijders (2001).

6. Use multiple processes (`useCluster = TRUE, nbrNodes = ...`).

7. In combination with trying out a lower value for `firstg`, you might try setting a higher value for `doubleAveraging` in set_algorithm_saom. Double averaging is a different definition of the Robbins-Monro update step; see Section 7.3.3. You could try setting `doubleAveraging` to 1 or 2, which uses single averaging in the first 1 or 2 subphases; or you could set it to 4, avoiding double averaging altogether (if you have 4 subphases).

♢ *Error in solve.default(z$dfra) :*
system is computationally singular: reciprocal condition number = 2.34809e–35
(or some other very small number – note that '*e–35*' means 10 to the power –35.)

or

♢ *Error for inversion of d11*

This can happen at the end of estimation in function siena, when the covariance matrix is singular. It means that some effects in the model are linearly related, or are always 0.

Solutions.
Check your data (look at the description of the variables given as the output of write_report).
Check your model specification.
Find out which effect might be especially correlated with some other effects, and treat it by a score-type test without estimating (i.e., specify it with `fix = TRUE, test = TRUE`). (Also see the discussion of the Hauck-Donner phenomenon in Section 7.12.2; this issue might be relevant here.)

♢ *Error in initializeFRAN(z, x, initC = FALSE, ...) :*
Unexpected effect name: someShortName
(where *someShortName* is a short name you supplied in the model specification).

Probably your effects object was made by an earlier or newer version of RSiena than the version you are using now.

Solution.
First check that the short name was correctly spelled.
If this is all right, then recreate the effects object.
You can check that an effect is available in your version of RSiena by requesting

```
effectsDocumentation(myeff)
```

(where you should substitute the name of your effects object for `myeff`).

♢ *Endowment effect not supported*
(for a specified endowment or creation effect).

This endowment or creation effect was not implemented.

Solution.
First check that the short name was correctly spelled and is available in your version of RSiena, by requesting

```
effectsDocumentation(myeff)
```

(where of course you should substitute the name of your effects object for `myeff`). If this is all right, the recreate the effects object.

♢ *total probability non-positive*

The calculation of probabilities for the next ministep failed, probably because no change was allowed for this actor in this period, given the current state of the model.

This may have to do with structurally determined values (see Section 5.3.1) and/or with joiners and leavers (see Section 5.3.3); or with a restriction that a variable (tie variable, behavioral variable) is monotonic (Section 5.2.4), i.e., can only increase or only decrease.

Perhaps the most likely possibility is that you are working with a multiple group data set (perhaps using multi_siena) for which some groups are only-increasing or only-decreasing, which then is taken by RSiena to apply also to the simulations carried out for this group, as mentioned in Section 5.2.4.

Solution.
Try to find out why it is possible in your data set that some actor has no options at all to choose from.

If you are working with a multi-group data set, use `allowOnly = FALSE` in the call of as_dependent_rsiena, as recommended in Section 5.2.4; also see the help file for as_dependent_rsiena.

Note that if you wish to forbid some actors from making any changes in some dependent variable, you can use a rate function dependent on a dummy actor covariate (effect `rateX`), with parameter fixed to a large negative value such as –100, which is sufficiently close to infinity (recall the exponentation in the calculation of the rate function) to effectively prevent this actor from taking any ministeps.

♢ *Error in matrix(0, nrow = sum(y[x,] != 0), ncol = 3) :*
invalid "nrow" value (too large or NA)

You are using a dyadic covariate containing only missing values.

Solution.
Correct the definition of this dyadic covariate.

15.3 As the result of a score-type test (including time test)

♢ *Error in solve.default(v9) : Lapack routine dgesv: system is exactly singular*
Error in if (cvalue < 0) cvalue < − 0 : missing value where TRUE/FALSE needed.

This can happen as the result of a score test requested in function siena; or the score test requested in function test_time. In the first case it indicates that there are linear dependencies in the list of effects (estimated and fixed) that are used for siena. If this error message occurs for test_time, it indicates linear dependencies in the list of effects estimated in the siena run analyzed by this test_time, together with the interactions with time dummies tested by test_time. See Sections 6.13 and 9.2.

Solutions.
For siena: drop some of the requested score-type test, retaining only a set of tested effects between which there are no linear dependencies.
For test_time: exclude some of the requested time heterogeneity tests by the excludeEffects parameter.

15.4 In test_gof

♢ *Error in if (attr(obsData[[groupName]]$depvars[[varName]], "sparse")) { :*
argument is of length zero

This can happen directly when calling test_gof. It indicates that you used a wrong name for groupName or varName. See the help file for test_gof.

Solutions.
Use a correct groupName and varName.

♢ *Error in if (isbipartite) { : argument is of length zero*

This can happen directly when calling test_gof. It indicates that you used a wrong name for groupName or varName. See the help file for test_gof.

Solutions.
Use a correct groupName and varName.

16 For programmers: Get the source code

To do something with the source code, first you must get access to it. In the first place, it is good to know that for any R function that can be called, the source code is listed by writing the function name. Thus, e.g., if RSiena is loaded, the command

```
set_algorithm_saom
```

will list the code for the function with this name.

To get insight into a package, and certainly to modify or personalize it, it is necessary, however, to get the source code of the whole package. This is available from GitHub at https://github.com/stocnet/rsiena/.

A lot of programmers' documentation is also in the file Siena_algorithms.pdf which can be downloaded from the SIENA website.

17 For programmers: Other tools you need

Windows

1. Download and install the appropriate (version number depending on which R you are using) Rtools.exe from http://www.murdoch-sutherland.com/Rtools/. (I think this is not the right place any more - should be downloaded from CRAN.)
2. Make sure you check the box to amend your path during installation.
3. Beware: if you later install other programs containing utilities such as tar (delphi is one offender), you may need to uninstall and reinstall Rtools, as you need Rtools at the start of your path.
4. Add the path-to-the file R.exe to your path. Right-click on My Computer icon, select Properties/Advanced/Environment variables...
   Restart your computer to put the new path into effect.

Mac

1. Make sure the Xcode tools are installed.
2. Add the path-to-the-file R.exe to your path.

linux Add the path-to-the-file R.exe to your path.

18 For programmers: Building, installing and checking the package

In a command prompt or terminal window, navigate to the directory immediately above the siena source tree. Here we assume the source tree is in a directory called RSiena. (You may have minor difficulties if it is not the same as the name of the package you are trying to build or install.)

For Windows computers, the following 'type' instructions are Dos commands. A convenient way to apply them are by including them in a batch file (extension name `.bat`) followed by a name with `pause` so that the Dos window – that will contain the error messages if there are any – will still be there when all is over.

Install Installing will recreate the binary and install in your normal R library path. Type
`R CMD INSTALL RSiena`

Build Building will create a tar ball. Type
`R CMD build RSiena`
This may give warning messages about the line endings if you run it on Windows. Do not worry, unless you have created any new source files, when it might remind you to set the property of eol-style on them when you add them to the repository.

Check Checking is a process designed to ensure that packages are likely to work correctly when installed. Type `R CMD check RSiena_1.6.14.tar.gz`
(where the version number is adjusted to match the tar ball name.)

zip file To make a zip file that can be used in Windows for 'installing from a local zip file', and therefore is easy for distribution to others, type
`R CMD INSTALL --build RSiena_1.6.14.tar.gz`
(where again the version number is adjusted to match the tar ball name.)

If you make a change you need to INSTALL the new version in order to test it, and before you commit any changes to a repository you should check your new version. Make sure you get *no* warnings or errors from the check.

You can also INSTALL from a tar ball, and check a source tree.

If you have permission problems on Linux or Mac, you may need to do the first install from within R, so that the necessary personal library directories will be created. Use
`install.packages(tarballname, repos = NULL)`
after creating a tar ball. (Or possibly just try to install some other package within R which will create the directories for you.)

You can unpack a tar ball by using
`tar xf tar-ball-name`

19 For programmers: Understanding and adding an effect

If you wish to check the definition of an effect, you can locate it in the source code and study it. You may also add effects to your personalized version of RSiena. If you think the effect could be useful for others, too, it will be appreciated if you propose it for inclusion through one of the discussion lists or directly to the maintainer of the package. This section gives the outline of the procedure for adding an effect, and then presents an elaborate example.

If you only wish to understand an effect without creating a new one, then you may follow the appropriate steps of this section. The main things then are to go to the Effectfactory.cpp file, find the name of the effect you are interested in and from there the function that implements it and read the code of this function.

The explanations here are not yet given for generic effects, which allow a more streamlined construction of effects. Looking at the code, starting with the file EffectFactory.cpp, may be helpful for understanding this construction of generic effects. For example, compare the construction of `sameXTransTrip` to that of `sameXInPop`.

1. Work out the definition of the effect and the contribution or change statistic. For network effects, the change statistic is

$$\Delta_{kij}(x) = s_{ki}(x^{+ij}) - s_{ki}(x^{-ij}) \tag{60}$$

   where x^{+ij} is the network with the tie $i \to j$ and x^{-ij} is the network without this tie.

2. The list of all defined effects can be obtained from effectsDocumentation, which produces a file effects.html or effects.pdf. All effects are also listed, with their definitions, in the manual (Section 13).
   Determine an existing effect that is most similar to this effect (or perhaps more than one). In the file effects.html or effects.pdf the effects are grouped by effectGroup.

3. Open the file allEffects.csv located in "RSiena\data". The default program for opening a .csv file usually is Excel, but other editors may be more helpful for opening this particular file; e.g., NotePad or NotePad++. It must not be saved as a Excel file!

   - You will see that the first column is called effectGroup. These groups define combinations of dependent variable, effect type, and covariates (if any) (e.g. nonSymmetricObjective, bipartiteSymmetricObjective). Identify the effectGroup where this effect belongs. Determine which is which by considering some examples in this file or in the result of effectsDocumentation().
     For covariate-related effects for two-mode networks, see extra remarks in Section 19.3.
   - Insert a new row in this group. Copy a row that corresponds best to your new effect and modify effectName, functionName, shortName, and more if this seems necessary. Perhaps your new effect is suitable in more than one group; then a new row can be made for all these groups, differing only in the name of the effect group; e.g., check that there are three versions of `inPop`, for directed, non-directed, and bipartite (two-mode) networks. Assume our new effect has shortName *newEf*.
     In some cases, the new function will have extra parameters, as you can see from other examples; this is mostly the case, if one function is being used to define more than one effect.
     For how to deal with internal effect parameters, look up a function defining an effect that has such a parameter.
     It is advisable to use the hash sign (`#`) in the effect and function names. See subsection 19.1.
   - Build the package and install it. Check from R that the new effect (which has only been created nominally) appears now in the effects object in RSiena.
     If this is not the case, there may be further changes necessary in the file effects.r; also see Section 19.3.

4. Open the folder "RSiena\src\model\effects". In an editor open the files AllEffects.h and EffectFactory.cpp.
   These are C++ files; using a C++ editor is convenient but not necessary. Note however that you must save the files as ASCII (raw text) files without changing their names. Let us use the name *NewEffect* as the function name to be used (replace this by whatever is appropriate).

- In the file AllEffects.h you need to add the line
  `#include "NewEffect.h"`
  where it is alphabetically appropriate.
- In the file EffectFactory.cpp, at the appropriate place, add the lines

```
    else if (effectName == "newEf")
{
 pEffect = new NewEffect(pEffectInfo);
}
```

- Now you will need to create two files (namely header and source files for C++) that should be called NewEffect.h and NewEffect.cpp. If there is a similar effect to the one you want to add it is usually easier to use it as a template.
  We recommend opening any effect file to see how the syntax works, but creating a new effect will be hard without knowing at least a bit of C++.
- Add the name NewEffect.cpp to the file sources.list. This is a long file without any hard returns. Separation of filenames is by blanks. It does not matter where you put it .

5. Once you are done editing you should build the package again and install it (from the command prompt) and then go to R to see if it is available to you.
   It is a good idea to check the target statistics computed for a simple two-wave data set such as s50. Examples are in the script https://github.com/stocnet/rsiena/blob/main/checkEffects.R.

As examples, start with simple effects. For example, a network effect depending on a nonlinear transformation of outdegree, or a behavior effect depending nonlinearly on the behavior and nothing else. After having obtained experience with such a simple effect, continue with the effect that you are interested in.

If you want to use the effect in interactions, consult the section in Siena_algorithms.pdf about interaction effects. If you wish to construct a dyadic or ego effect, note that you should define a tieStatistic rather than an egoStatistic.

Note that if your new effect could usefully be used as part of a multiple network effect you should use the generic effect approach and not the following.

19.1 The use of the hash (#) sign in names of effects and estimation statistics

If an effect contains an internal effect parameter, it is advisable to report the value of the internal effect parameter in the names of the effect and the estimation statistics reported when printing the effects object and the estimation results. These names are the columns `effectName` and `functionName` in the effects object, and their origins are the corresponding columns in file `allEffects.csv`. The value of the internal effect parameter can be included by the use of

the hash sign (#). In function set_effect, occurrences of "#" in columns effectName and functionName are replaced by the value of the internal effect parameter (given explicitly by the user, or else the default value).

Thus, the hash sign (#) is a placeholder for the value of the internal effect parameter. When you look in the file allEffects.csv, you will see many occurrences of this symbol.

A difficulty occurs when the replacement has been done (so there is no more "#"), but a new call of set_effect is made with a new value of the internal effect parameter. To make the correct replacement then, the following rule is followed for defining the names in allEffects.csv:

- The symbol # should occur in the effectName and functionName in allEffects.csv in only the following substrings:
  - "1/#"
  - "(#)"
  - "#-"
  - "= #"
  - "+ #"
  - "- #"

  where "␣" denotes one blank space (used here for the sake of clarity, not for use in allEffects.csv).
- No confusion should be possible, i.e., no substrings in allEffects.csv of these kinds are permitted where # is any number.

19.2 Example: adding the truncated out-degree effect

As an example, we show how the truncated out-degree effect (short name outTrunc) was added. It is defined by

$$s_i^{\text{net}}(x) = \min(x_{i+}\,,\, p) \tag{61}$$

where p is an internal effect parameter.

1. The change statistic (60) is

$$\Delta_{ij}(x) \;=\; \begin{cases} 1 & \text{if } \{x_{i+} < p, x_{ij} = 0\} \text{ or } \{x_{i+} \le p, x_{ij} = 1\} \\ 0 & \text{else.} \end{cases} \tag{62}$$

   Note that for this effect the case for going up ($x_{ij} = 0$) must be distinguished from the case for going down ($x_{ij} = 1$).

2. In the file allEffects.csv the name of the effect group nonSymmetricObjective seems to cover the type of effect we are considering, and also contains other effects such as out-degree activity which are very similar to this effect.
   The row for the out-degree activity (sqrt) effect was copied and inserted below this row.

The 'effectName' was changed to `outdegree-trunc(#)`, the 'functionName' to `Sum of outdegrees trunc(#)`, and the 'shortname" to `outTrunc`. The hash sign (#) in these names will be replaced by the value of the internal effect parameter in the written output. The `parm` column, which defines the default value of the internal effect parameter, was set to 5.
The package was built. Loading it in R and creating an RSiena data set showed that indeed the effect was there.

3. The name TruncatedOutdegreeEffect was chosen for the new function.
In the file AllEffects.h the line

```
#include 'TruncatedOutdegreeEffect.h'
```

was included at the appropriate alphabetic place.

In the file EffectFactory.cpp, after the piece referring to `effectName == "outActSqrt"`, the lines

```
else if (effectName == "outTrunc")
{
     pEffect = new TruncatedOutdegreeEffect(pEffectInfo);
}
```

were inserted. This refers the program, when it encounters short name outTrunc, to the function TruncatedOutdegreeEffect. The next step was to construct this function.

4. To choose a template for TruncatedOutdegreeEffect, we could make various different choices; here it is important to have a look at the various effects defined in Chapter 13 that depend only on the outdegree. Consider the effects Outdegree activity - sqrt (short name `outActSqrt`) and sum of (1/(out-degree + p) (short name `outInv`) as possible examples. A look in EffectFactory.cpp shows that these are implemented using the functions OutdegreeActivitySqrtEffect and InverseOutdegreeEffect, respectively. Therefore look at the files OutdegreeActivitySqrtEffect.cpp and InverseOutdegreeEffect.cpp where these functions are defined.
The former defines the effect through a 'calculateContribution' function, which defines the tie flip contribution (the function called $\Delta_{ij}(x)$ above) and tieStatistic, which is the function $r_{ij}(x)$ when the effect can be defined as

$$s_i^{\text{net}}(x) = \sum_j x_{ij}\, r_{ij}(x)\ . \tag{63}$$

The latter defines the effect through a calculateContribution function and an egoStatistic function, which is the effect as defined in (61). It should be noted that generally effects can be defined either by the one or the other combination.

Since our new effect cannot be expressed in a straightforward way by an equation of the type (63), we chose to use the files InverseOutdegreeEffect.h and InverseOutdegreeEffect.cpp as templates. This has a second advantage: the `outInv` effect has an effect parameter, which we also need to represent the parameter p in (61).

As a first step, the files InverseOutdegreeEffect.h and InverseOutdegreeEffect.cpp were saved under the new names TruncatedOutdegreeEffect.h and TruncatedOutdegreeEffect.cpp.
For the header file TruncatedOutdegreeEffect.h, all strings 'inverseoutdegreeeffect' were changed into 'truncatedoutdegreeeffect' while retaining the original use of upper and lower case. The explanation also was adapted. The header file now implies that for the function `TruncatedOutdegreeEffect` functions are needed of the types calculateContribution, endowmentStatistic and egoStatistic.
This was implemented in the file TruncatedOutdegreeEffect.cpp, which just was created by renaming InverseOutdegreeEffect.cpp. First all strings 'outdegreeactivitysqrteffect' were changed into 'truncatedoutdegreeeffect', again retaining the original use of upper and lower case.
To understand the C++ syntax, keep into account the object-oriented nature of C++. The argument this is a pointer referring to the object in which the current function is defined, and the arrow -> indicates a further pointer; thus, the variable
`this->pNetwork()->outDegree(this->ego())`
refers to the outdegree of ego (ego being denoted in our mathematical formulae by i) in the current network – in other words, x_{i+}.
The variable `this->lc` refers to the internal effect parameter, denoted in our formulae by p. The return statement defines the function value that is returned when the function is called.
Armed with this knowledge, we specified the change statistic, implementing (62), as follows.

```
double TruncatedOutdegreeEffect::calculateContribution(int alter) const
{
    double change = 0;

    // Current out-degree
    int d = this->pNetwork()->outDegree(this->ego());
    if (this->outTieExists(alter))
    {
        // After a tie withdrawal, the new out-degree would be d-1, and
        // the new effect value would have decreased by 1 if d <= this->lc

        if (d <= this->lc)
        {
            change = 1;
        }
    }
    else
    {
```

```
        // When introducing a new tie, the new out-degree would be d+1, and
        // the new effect value would have increased by 1 if d < this->lc

        if (d < this->lc)
        {
             change = 1;
        }
    }

    return change;
}
```

The effect statistic, implementing (62), was specified as follows.

```
double TruncatedOutdegreeEffect::egoStatistic(int ego,
    const Network * pNetwork)
{
    // Current out-degree
    int d = this->pNetwork()->outDegree(this->ego());

    if (d <= this->lc)
    {
         return d;
    }
    else
    {
         return this->lc;
    }
}
```

5. To the file src\sources.list, the item `model/effects/TruncatedOutdegreeEffect.cpp` was added.

6. Having done this, the package was built and installed again. (To be honest, there first were some errors; but the error messages from the compiler are quite clear and easily led to solving the errors.)
   Upon starting R and loading RSiena, indeed the new effect was available. For an easy check, the following commands were used.

```
mynet       <- as_dependent_rsiena(array(c(s501, s502), dim = c(50, 50, 2)))
mydata      <- make_data_rsiena(mynet)
myeff       <- make_specification(mydata)
myeff       <- set_effect(myeff, outTrunc, parameter=3)
ans          <- siena(data=mydata, effects=myeff)
summary(ans)
```

   The parameter was set at 3, because the maxima of the observed out-degrees in the two data sets s501 as well as 502 were 5, so the 'outdegree-trunc(#)' effect would be highly collinear with the outdegree effect if the default parameter of 5 were used. This led to good

convergence. To check the calculation of the statistics, it was noted that the output file mentioned the target values

```
Observed values of target statistics are
  1. Number of ties                              116.0000
  2. Number of reciprocated ties                  70.0000
  3. Sum of outdegrees trunc(3)                  105.0000
```

The value of the target statistic for the new effect should be

$$\sum_i s_i^{\text{net}}(x(t_2)) \;=\; \sum_i \min(x_{i+}(t_2), 3)\;.$$

This can be directly calculated in R by requesting

```
sum(pmin(rowSums(s502),3))
```

which indeed returns the value 105, confirming that the calculation of the ego statistic seems correct.

7. To complete the extension of the package by this effect, it also was added to the set of effects for symmetric and bipartite networks. This was done by inserting, at appropriate places in the file allEffects.csv, the same line but now with effectGroup changed to bipartiteObjective and symmetricObjective, respectively.

19.3 Notes on effectGroups and two-mode networks

For two-mode networks the difference between the two node sets implies some peculiarities. Recall that effectGroups define combinations of dependent variable, effect type, and covariates (if any). They are used in the file `allEffects.csv` and in function effects.r. In the output of effectsDocumentation() (mentioning no effects object), the effectGroups are given as categories of the effects.

In the function make_specification in file effects.r, some additional measures are taken for effects in effectGroup covarBipartiteObjective, because this effect group does not differentiate between effects of covariates defined on the first or on the second mode. This implies that for adding such effects, it will be necessary to see whether this function (i.e., make_specification) also must be modified; this will have to be done in function covarBipartiteEff in file effects.r. (Note: it would be preferable perhaps to have separate effect classes for covariates on the first and on the second mode, as done for the effect groups in the following paragraphs.)

A different approach was taken for effectGroups covarABehaviorBipartiteObjective and covarBBehaviorBipartiteObjective: the former is for covariates on the first node set, the second for covariates on the second node set.

EffectGroup dyadFirstSecondBipartiteObjective is for a dependent network and two one-mode dyadic covariates; if the network is two-mode, then the first should be for the first mode, the second on the second mode.

For effects defined for one dependent network and two actor covariates, the following effectGroup is defined:

- doubleCovarNetObjective is for effects where the first covariate is for the actors and, if the network is two-mode, the second covariate is for the second mode; moreover, both covariates should have no more than 20 distinct values. (This restriction is currently coded in effects.r, make_data_rsiena.r, and three C++ functions. That is perhaps overdone.)

For effects defined for one dependent two-mode network X and one dyadic covariate, the following three effectGroups are defined:

- dyadBipartiteObjective is for effects where the dyadic covariate is also two-mode and matching the node sets of X;
- dyadFirstBipartiteObjective is for effects where the dyadic covariate is one-mode and defined for the first mode of X;
- dyadSecondBipartiteObjective is for effects where the dyadic covariate is one-mode and defined for the second mode of X.

Further some of the complex effectGroups are the following. For co-evolution of multiple networks:

- nonSymmetricNonSymmetricObjective is for combination of two directed networks, one in the role of dependent network and the other in the role of explanatory network;
- nonSymmetricSymmetricObjective is for the combination of a directed network in the role of dependent network and a non-directed network in the role of explanatory network;
- nonSymmetricSymmetricSObjective is for the combination of a directed network in the role of dependent network and a non-directed network in the role of explanatory network, where also the primary setting can have the role of explanatory network (and is treated with cross-lagged statistics as usual for co-evolution models);
- tripleNetworkObjective is for combinations of three networks where for the two in the role of explanatory networks,
- bipartiteNonSymmetricObjective is for combination of a two-mode dependent network and a directed network in the role of explanatory network;
- bipartiteSymmetricObjective is for combination of a two-mode dependent network and a non-directed network in the role of explanatory network;
- bipartiteBipartiteObjective is for combination of two two-mode networks, one in the role of dependent network and the other in the role of explanatory network.

For effects defined for two dependent networks, X in the role of dependent network and W in the role of explanatory network, and one actor covariate V, the following effectGroups are defined:

- covarNetNetObjective is for effects where both networks are one-mode;

- covarANetNetObjective is for effects where X is one-mode, and if W is two-mode, V is defined for the first mode;
- covarAWNetNetObjective is for effects where W is one-mode, and V is defined for the first mode;
- covarADNetNetObjective is for effects where, if W is two-mode, V is defined for the first mode, and both networks should have the same second mode;
- covarBNetNetObjective is for effects where, if W is two-mode, V is defined for the second mode;
- covarBXNetNetObjective is for effects where, if W is two-mode, X should be one-mode, and V is defined for the second mode.

For effects defined for two dependent networks, X in the role of dependent network and W in the role of explanatory network, and a dyadic covariate, the following effectGroups are defined:

- dyadANetNetObjective is for combinations of two networks and a dyadic covariate, where the network in the role of dependent network (X) should be one-mode, and the dyadic covariate should be on the node set ('squared') of the dependent network;
- dyadBNetNetObjective is for combinations of two networks and a dyadic covariate, where the dyadic covariate should have the same first and second node sets as the explanatory network.

The data structures for which effects in dyadANetNetObjective is suitable is a subset of those for which dyadBNetNetObjective is suitable.
For dependent behavioral variables:

- behaviorOneOneModeObjective is for dependent behavioral variables and two explanatory one-mode directed networks;
- behaviorSymSymObjective is for dependent behavioral variables and two explanatory one-mode non-directed networks;
- behaviorOneModeSymObjective is for dependent behavioral variables and two explanatory one-mode networks, the first directed, the second non-directed.
- behaviorBipBipObjective is for dependent behavioral variables and two explanatory two-mode networks with identical actor sets;
- dyadBehaviorNetObjective is for dependent behavioral variables, an explanatory one-mode network and an explanatory dyadic covariate.

If you wish to add an effect with a combination of variables that is not yet implemented, you have also to treat it in effects.r (which will require some figuring out) and mention the effectGroup in effectsDocumentation.r.

A Appendix: Changes compared to earlier versions

This presents the main changes, especially those directly visible to the user, in reverse time order from the current version until version 1.0.6 of end October 2009. For more complete information about changes, consult the `NEWS.md` file in the source code, and the `ONWS` ('old news') and `ONWS_gh` ('old news in GitHub') files, which contain more complete listings. The `NEWS.md` file can also be consulted during an R session if you installed the package RSiena, provided you have also attached the packages `commonmark` and `xml2` to render the Markdown format.

After installing these two packages, to get the NEWS about any package that you installed (such as RSiena), give the command

```
news(package="packageName")
```

(where you can fill in the package name).

- 2026-10-22 GitHub, package version 1.6.14.
  Changes in RSiena:
  - Functionality:
    - The multi-group option (make_group_rsiena) is now available for models with continuous behavior variables, with parameter starting values based on one fit to all groups.
    - The option `splitDepvars` in set_algorithm_saom, introduced in version 1.6.3, was rolled back and does not exist any more.
  - Bug corrections:
    - Analytic Phase 1 derivatives of the feedback statistic for continuous behavior variables in `updateDerivativeSde` corrected (it used the behavior at the end instead of at the start of the period).
    - There was an error when using multiple processors and maximum likelihood estimation. This was corrected.
- 2026-10-01 GitHub, package version 1.6.13.
  Changes in RSiena:
  - Effects:
    - Effect `WXX` corrected for two-mode networks.
    - New effect `WXZX` (contributed by Harald Waxenecker).
    - Effects `outPop.c`, `outPopSqrt`, `outPopMore`, `outPopSqrtMore`, `outPopThreshold`, `inAct.c`, and `inActSqrt` now have internal effect parameter 1.
  - Methods:
    - New method print.coDyadCovar.
    - New method print.varDyadCovar.
- 2026-09-14 GitHub, package version 1.6.12.
  Changes in RSiena:
  - Effects:

    - Effect `toU` corrected.
    - Effects `toU` and `WXX` now also available for two-mode networks.
      Effect group `dyadANetNetObjective` is empty now. (Retained for possible later use.)
    - New effectGroup `dyadFirstBipartiteObjective`.
    - Change to `make_specification` to allow empty effect groups.
  - Bug corrections:
    - Effect `toU` corrected. (Also mentioned above.)
    - Adherence to positivity constraints during parameter updates in Phase 2 corrected for the `doubleAveraging=TRUE` option.
    - Correction of use of `splitDepVars` in `initializeFRAN`.
    - Changes in `CalculateDerivative` in `phase1.r` (in a part used only by multi_siena to handle estimation difficulties).
    - In transformScript, deleted the last `s` in `as_covariate_rsienas`.
    - Error message in as_covariate_rsiena for square dyadic covariates corrected.
  - Functionality:
    - Changes in interpret_size making the result a list of objects, starting with one object for each dependent variable; and dropping interpret_size.sienaEffects, i.e., the possibility to run interpret_size without a sienaFit object.
    - `toggleProbabilities` are returned by interpret_size only if `getChangeStatistics` is `TRUE`.
    - Changed the class name of the object created by interpret_size from sienaRI to siena_sizes.
    - New method print.siena_sizes giving the significances/p-values of the separate estimated parameters, the semi-standardized parameters, and the degree of certainty.
    - New method print.summary.siena_sizes giving the standard deviations of the change statistics and the expected relative importance of the effects.
    - Better error messages for make_data_rsiena in the case of incorrect node sets for bipartite networks.
    - Component `initial_theta` added to sienaFit objects.
  - Documentation:
    - Explanation of effectGroups `dyadBipartiteObjective`, `dyadSecondBipartiteObjective`, and `dyadFirstBipartiteObjective`, involving a dyadic covariate and a dependent two-mode network, added to Section 19.3 of this manual.

- 2026-07-16 GitHub, package version 1.6.11.
  Changes in RSiena:
  - Functionality:
    - New auxiliary functions AC_Moran and AC_Geary.
    - In transformScript, the replacement for `sienaCompositionChangeFromFile` was put before the replacement for `sienaCompositionChange`.
    - The component `theta` of results of function siena, the result of functions coef, and estimate_onestep have got a class coef_siena, which has its own print method (to allow vertical printing).

        * Function coef now has default `shortenNames=FALSE`.
        * Alias coefficients for coef.
        * Remark to console about output file for siena only when `silent=FALSE`.
        * Undo sorting of returned data frames for `returnChains=TRUE` (in `siena07models.cpp`).
    - Bug corrections:
        * Running siena with `returnChains = TRUE, returnDataFrame = TRUE` corrected.

- 2026-06-10 GitHub, package version 1.6.9.
  Changes in RSiena:
    - Effects:
        * Selection of effects for combinations of two dependent networks and an actor covariate improved, which leads to including more effects for such combinations where one of the dependent networks is bipartite (see 'coding' below).
    - Improvements of functionality:
        * New function estimate_onestep.
        * Option `onestep` for update_theta.sienaEffects changed.
        * Hidden option to use the one-step estimator with `prevAns` in siena (experimental).
        * print.sienadata and print.sienaGroup now specify the node sets for covariates and restricts printed line lengths to 64.
        * More informative error message in make_data_rsiena if the number of elements of the second node set for bipartite networks is not correct.
        * In transformScript, the replacement for descriptives.sienaGOF was put before the replacement for sienaGOF.
    - Coding:
        * Reorganization and clarification of `effectGroups` for combinations of two dependent networks and an actor covariate: these now are `covarNetNetObjective`, `covarANetNetObjective`, `covarAWNetNetObjective`, `covarADNetNetObjective`, `covarBNetNetObjective`, `covarBXNetNetObjective`. See Section 19.3.

- 2026-05-15 GitHub, package version 1.6.7.
  Changes in RSiena:
    - Improvements of functionality:
        * Option `onestep` for test_parameter(..., method="score", ...) .
        * Option `onestep` for update_theta.
        * Name `sienaNet` added to transformScript as one of the names to be replaced.
        * make_effects now also works for `sienadata` objects older than 1.4.10.

- 2026-04-20 GitHub and CRAN, package version 1.6.6.
  Changes in RSiena:
    - Corrections related to sanity checks:
        * Stack handling improved in `siena07utilities.cpp` for compliance with CRAN.

- 2026-04-08 GitHub, package version 1.6.5.
  Changes in RSiena:
  - Corrections related to sanity checks:
    * Keyword handling corrected in help files for interpret_influence, interpret_selection, make_specification, set_effect, test_gof_auxiliary and test_time.
    * In help file for interpret_size_dynamics, part of example code disabled for testing due to time exceeding 5 seconds in checks.
- 2026-04-07 GitHub, package version 1.6.4.
  Changes in RSiena:
  - Bug corrections:
    * For test_gof, a bug in the handling of auxiliary functions was fixed.
- 2026-03-21 GitHub, package version 1.6.3.
  Changes in RSiena:
  - New effects:
    * New effects `outdegMixedPop` and `indegMixedPop`.
    * `endowment` allowed for `XWX1`, `XWX2`, and `cl.XWX1`.
  - Improvements of functionality:
    * Auxiliary function egoAlterCovarComb for test_gof.
    * Option `splitDepvars` for set_algorithm_saom for making zero blocks in D for different-dependent variable effects.
    * The quasi-Newton Raphson update step at the end of Phase 1 used now uses the partially diagonalized (and possibly zero blocks) version of D, unless `splitDepvars`=-1.
    * Column names `inter1` and `inter2` of the result of effectsDocumentation changed to `covar1` and `covar2`.
    * Better error message for set_interaction in case `shortNames` is not a list.
    * In sienaTest objects, if the test is one-dimensional, the standard error of the linear combination is given as an additional component `sterror` and included in print.sienaTest.
  - Bug corrections:
    * For transformScript, processing of Multipar.RSiena added.
    * For test_gof, all arguments are transferred to the call of sienaGOF.
    * For interpret_selection, all arguments are transferred to the call of selectionTable.
    * For interpret_influence, all arguments are transferred to the call of influenceTable.
  - Documentation:
    * Manual updated to new function names and use of autograph for plotting influence and selection tables.
    * In help file for set_interaction, treatment of `shortNames` corrected.
    * In help file for set_effect, treatment of `covar1` and `covar2` corrected.
- 2026-25-02 GitHub, package version 1.6.2.
  Changes in RSiena:
  - New function transformScript.

  - New method write_report for sienadata objects, replacing print01Report (which still is retained).

- 2026-01-20 GitHub, package version 1.6.1

  Changes in RSiena:

  - New effects:
    * Effects `sameXOutAct`, `diffXOutAct`, `crossXOutAct` also implemented for two-mode networks.

- 2026-01-02 GitHub, package version 1.6.0

  Changes in RSiena:

  - New function names:
    * Overhaul of naming system of exported functions. Old function names still are available. See Chapter 3.
    * Chapter 3 was inserted in the manual to explain the new function names.
    * Class name `siena` for siena data sets replaced by `sienadata`.
  - Coding:
    * Completely refactored diffusion rate effects.
    * For sienaRI: corrected contributions for the first potential ministep in each period, and dropped calculation of the first ministep after the last observation wave, and the RI measure at the last observation moment.
      Further see the `NEWS.md` file.
  - New effects:
    * New distance 2 exposure diffusion rate effects `totInExposureDist2`, `totAInExposureDist2`, `anyInExposureDist2`, behavior effects `totPopAlt`, `totGroup`, `indegAvGroup`, `indegtotGroup`, `totGwdspFFAlt`, `totGwdspFBAlt_nc` (Daniel Gotthardt).
    * New behavior effects without centering `quad_nc`, `avInAltDist2_nc`, `totInAltDist2_nc`, `avTinaltDist2_nc`, `totAInAltDist2_nc`, `totGwdspFFAlt_nc`, `totGwdspFBAlt_nc`.
    * New effects `avSameXAlt`, `totSameXAlt`, `avSameXInAlt`, `totSameXInalt`, `avXRecAlt`, `totXRecAlt`, `avSameXRecAlt`, `totSameXRecAlt`, `avAltSameX`, `totAltSameX`, `avRecAltSameX`, `totRecAltSameX`.
    * New effects `avInAltAltX`, `totInAltAltX`, `avRecAltAltX`, `totRecAltAltX`, `avInAltSameX`, `totInAltSameX`.
  - Improvements of functionality:
    * Distinguished between three algorithm objects. See Chapter 3.
  - Bug corrections:
    * `checkImpossibleChanges` restricted to networks.
    * `checkZeroChanges` adapted to also cover behavioral variables.
  - Documentation:
    * A new subsection was added to `Siena_algorithms` to explain the new construction of diffusion rate effects (Daniel Gotthardt).
    * Appendix A of this manual, "List of Functions in Order of Execution", was dropped and the author of this appendix, Paulina Preciado Lopez, was accordingly dropped from the list of authors of the manual. Greetings, Paulina!

- 2025-10-28 GitHub, package version 1.5.6.
  Changes in RSiena:
  - Improvements of functionality:
    * Auxiliary function egoAlterCombi for Goodness of Fit now omits values where the behavioral variable is missing.
  - Bug corrections:
    * Bug corrected that may have affected effects `sameXInPop`, `diffXInPop`, `sameXOutAct`, `diffXOutAct`, and `sameXVInPop`, leading to divergence or crashes.
- 2025-10-17 GitHub, package version 1.5.3.
  Changes in RSiena:
  - New effects:
    * `quad_cc`, `avAlt_cc`, `totAlt_cc`.
- 2025-09-06 GitHub, package version 1.5.2.
  Changes in RSiena:
  - Improvements of functionality:
    * Option `returnChangeContributions=TRUE` added to siena07, allowing users to extract the `changeContributions` to the sienaFit object.
    * interpret_size_dynamics reinstated based on the above new option, replacing the original code.
    * selectionTable got an attribute `quad` indicating whether the plot is a quadratic function.
    * Users can now extract now extract the changeContributions when running siena07 by setting an argument `returnChangeContributions=TRUE`. If used together with `nsub=0` and `prevAns` or modified initial values in the effects object, especially useful for post-estimation, e.g. in interpret_size_dynamics.
  - New effects:
    * `outActMore_ego`, `outActSqrtMore_ego`, `outMore_ego`, `outPopMore`, `outPopSqrtMore`, `outPopThreshold`.
- 2025-07-12 GitHub, package version 1.5.1.
  Changes in RSiena:
  - Improvements of functionality:
    * sienaRI reinstated.
    * Option `prML=2` for maximum likelihood estimation using the `move` proposal step reinstated (set_algorithm_saom).
  - Effects:
    * New effects `divOut_ego`, `divIn_ego`.
- 2025-07-07 CRAN, New version 1.5.0.
  New maintainer: Christian Steglich.
- 2025-07-05 GitHub, package version 1.4.25.
  Changes in RSiena:

  - Effects:
    * Some effects available for constant dyadic covariates were by accident not yet available for changing dyadic covariates. Now they are. The effects in question are `XWX`, `XWX1`, and `XWX2`.
    * Coding:
    * Memory leaks closed.

- 2025-05-03 GitHub, package version 1.4.23.

  Changes in RSiena:

  - Bug corrections
    * Repaired (hopefully) memory leak in `getTargetActorStatistics` and `getTargetsChangeContributions` in `siena07setup.cpp`.
    * Repaired (hopefully) memory leak in `move` in `MLSimulation.cpp`.
    * Repaired memory leak in `DoubleCovariateCatFunction`.
  - Effects
    * New effects `fromAny`, `sameInXCycle4`.
    * Effect shortName `cycle4ND` replaced by `cycle4`.
    * Better treatment of missing covariate values in effect `sameXCycle4`.
    * Internal effect parameter values 3 and 4 for `sameXInPop`, `diffXInPop`, `sameXInPopIntn`, `sameXInActIntn`, `homXOutAct2`.
    * New C++ class `CatCovariateDependentNetworkEffect` (for `homXOutAct2`).
    * Added `(#)` to the `effectName` of `from`, `sameXInPop`, `diffXInPop`, `sameXVInPop`, `sameXVInPop2` in `allEffects.csv`.
  - New functionality

    In print.sienaEffects(..., includeShortNames=TRUE), the `effectNumber` is also printed.
  - Changes in documentation:

    Definition of effects `outRateLog`, `inRateLog`, and `recipRateLog` corrected in the manual.

- 2025-02-08 GitHub, package version 1.4.22.

  Changes in RSiena:

  - New functionality:
    * New auxiliary GOF function egoAlterCombi.
    * Parameter `showAll` added to plot.sienaGOF.
  - Effects
    * Parameter 0 (for logarithm) in `outActIntn`.
  - Documentation
    * Description of *sienacpp* omitted from manual.

- 2024-12-18 GitHub, package version 1.4.21; but the package has date 2024-12-12 and the Github release is from 2025-01-14.

  Changes in RSiena:

  - New functionality
    * New functions selectionTable and influenceTable.
  - Effects
    * New effect `altHigherEgoX`.
    * Effect `higher` also implemented for symmetric networks.
    * Used # in name of threshold effects.
  - Bug corrections
    * In siena.table, fixed parameter values are not reported as `NA`, but as their fixed values.
  - Messages
    * If includeInteraction is called with argument parameter, an error message appears that this keyword should not be given.
    * Extension of message in sienaTimeTest given in the case of collinearities.
  - Help pages
    * Improved explanation of the use of `interaction1` and `interaction2` in the help page for includeInteraction.

- 2024-11-10 GitHub, package version 1.4.20.
  Changes in RSiena:
  - New effects:
    * New effects `varAlt` and `avSimVarAlt`.
  - Bug corrections:
    * Interaction effects for continuous behavior turned off. The R side of includeInteraction for continuous behavior had been implemented but not the C++ side, so now an error message is given.

- 2024-09-03 GitHub, package version 1.4.19.
  Changes in RSiena:
  - New effects:
    * New effects `outThreshold` and `outThreshold2`.
    * `outActIntn` is an ego effect.
  - New functionality:
    * New network option 11 (*Contemporaneous evaluation statistics model*) for use of contemporaneous statistics for estimating all evaluation effects of the network variable.
    * If setEffect is called for an effect with `type=gmm`, an error message is given that the function called should be includeGMoMStatistics.
  - Bug corrections:
    * Error corrected that occurred for estimation by the GMoM if some effects are fixed.
    * Correction of list of effect names given when `thetaBound` is exceeded.

- 2024-07-31 GitHub, package version 1.4.18
  Changes in RSiena:
  - New effects:

    - New gmm effects for co-evolution of multiple networks: `crprod_gmm`, `to_gmm`, `from_gmm`.
  - Bug corrections:
    - Bug corrected in sienaGOF which occurred for models with tested effects if `iterations` is less than `sienaFitObject$n3`.

- 2024-06-06 GitHub, package version 1.4.13

  Changes in RSiena:

  - New effects:
    - New effect `homXOutAct2`.
  - Improvements of functionality:
    - The handling of internal effect parameters in the columns `effectName` and `functionName` of `sienaEffects` objects is improved. This was achieved by changes in functions setEffect and includeInteraction. Effectnames as reproduced by `sienaFit` objects should now contain the correct values of internal effect parameters.
    - New parameter `thetaBound` for siena07, which has the effect of stopping the estimation process if some parameters become too large (which would signal divergence).
    - Display of deviations in `siena07gui` modified so that numbers larger than or equal to `1e5` in absolute value are displayed in exponential format (and use only one line in the gui).
  - Bug corrections:
    - Function setEffect corrected (it did not give the proper internal parameters in the effectName).
    - Function updateSpecification corrected (it did not work properly for including interactions).

- 2024-04-23 GitHub, package version 1.4.11

  Changes in RSiena:

  - Effects:
    - Effect name `outOutDist2AvIntn` changed to `avAlt.2M.tot`.
    - New effects `avAlt.2M.tie`, `avAlt.2M.tot`, `avAltU.2M.tie`, `dist2OutInActIntn`, `nDist2ActIntn`, `sharedToU`.
    - Changed sqrt treatment in `outOutDist2ActIntn` and `outOutDist2AvIntn` / `avAlt.2M.tot`.
  - Improvements of functionality:
    - Method print.sienaEffects has an extra parameter `includeShortNames` to do what the name of this parameter suggests.

- 2024-03-25 GitHub, package version 1.4.10

  Changes in RSiena:

  - New effects:
    - New effects `crossXOutAct`, `outOutDist2ActIntn`, `outOutDist2AvIntn`, `inPopOutW`.
    - New effect group `doubleCovarNetObjective`.
    - New effects `sameXV` and `sameXVInPop` for bipartite and symmetric networks.
    - `sameXCycle4` added for one-mode and symmetric networks.

    - * sharedTo gets default internal effect parameter p=3 .
  - Improvements of functionality:
    - * Function updateSpecification now also updates interaction effects and initialValues.
    - * New parameter silent in sienaAlgorithmCreate activated.

- 2024-02-29 GitHub, package version 1.4.8.
  Changes in RSiena:
  - Bug corrections:
    - * Correction of memory leak in siena07setup.ccp for ML estimation.
  - New functionality:
    - * New parameter silent in sienaAlgorithmCreate.

- 2024-02-20 GitHub and CRAN (2024-02-22), package version 1.4.7
  Changes in RSiena:
  - Bug corrections:
    - * An error in a .cpp file was corrected.

- 2024-02-19 GitHub, package version 1.4.6
  Changes in RSiena:
  - Improvements of functionality:
    - * New parameter targets in siena07 , used to supersede the targets calculated from the data (not for use in estimation for regular data sets, see the help file for siena07).
    - * effectsDocumentation() reports to the console the name of the file that was written.
    - * sienaDataCreate stops with an error message if there is a bipartite network before a one-mode network.
    - * If the data contains a continuous dependent behavioral variable and the algorithm specifies conditional estimation, the estimation stops with a clear error message.
  - Bug corrections:
    - * Various corrections in the cpp code.
    - * Allow sienaDataCreate to work with a single variable defined as a dependent network given as a list of sparse matrices.
    - * Allow getEffects to construct effects of more than one dependent network on continuous behavior dependent variables.
    - * Changed the check for constant dyadic covariates for sparse matrices (GitHub issue #88).
    - * Some of the recently added effect groups were missing from effectsDocumentation. This led to an incomplete listing of the effects. They are now included.
  - Dropped functionality:
    - * sienaRI temporarily disabled because of a memory leak.
    - * The option prML=2 temporarily disabled because of a memory leak in the move proposal distribution (sienaAlgorithmCreate).
  - Other changes:
    - * List of changes in the code before 2022, but since the source code was hosted at GitHub (i.e., since September, 2020) moved from NEWS.md to ONEWS_gh .

- 2023-12-14 GitHub, package version 1.4.2.
  Changes in RSiena:
  - New effects:
    * New effects `outXMore`, `outMore3`.
    * Corrected effect `outMore`.
    * `Interactiontype` of `altLThresholdX` and `altRThresholdX` is dyadic. `Interactiontype` of `degAbsDiffX`, `degPosDiffX`, and `degNegDiffX` is "" (blank).
  - Improvements of functionality:
    * New parameter `iterations` in sienaGOF to allow shorter computations.
  - Bug corrections:
    * `bxeffects` initialized to 0 in `ContinuousVariable::accumulateScores`.
    * Improved initialization, may have corrected some unknown bugs.
- 2023-11-01 CRAN and GitHub, package version 1.4.1.
  Changes in RSiena:
  - Meta-data:
    * citation for RSiena improved.
    * Authors of RSiena now are listed as Tom A.B. Snijders, Ruth M. Ripley, Krist Boitmanis, Christian Steglich, Nynke M.D. Niezink, Felix Schoenenberger, and Viviana Amati.
  - Corrections of documentation:
    * Various small improvements in help files, in accordance with "Guidelines for Rd files" at CRAN.
- 2023-10-11 GitHub, package version 1.3.28
  - Changed effect:
    * `threshold`, `threshold2`, `threshold3` and `threshold4` changed to work with non-centered parameter (not backward-compatible).
  - Improved documentation:
    * Descriptions of effects `altInDist2W` and `totInDist2W` added to the manual (the effects had been there since a long time, but not documented).
- 2023-09-29 GitHub, package version 1.3.27
  Changes in RSiena:
  - New effects:
    * `avInSimDist2`, `totInSimDist2`, `sameEgoDist2`, `sameEgoInDist2`, `outMore2`, `divOutEgoIntn`, `divInEgoIntn`, `divOutAltIntn`, `divInAltIntn`.
    * `avTAltDist2` and `totTAltDist2` also implemented for behavior co-evolving with symmetric networks.
  - Improvements of functionality:
    * The slowness of siena07 that occurred since version 1.3.18 was ended (by correcting one line in siena07models.cpp).

  - Improved documentation:
    * Some explanation is given in the manual about internal effect parameters for interactions created by `includeInteraction`.

- 2023-08-15 GitHub, package version 1.3.26

  Changes in RSiena:

  - Coding:
    * Improved Phase 1 derivative matrix computation for basic SDE parameters (for continuous behavior).
    * Added continuous behavior to returned simulated data.
  - Corrections:
    * Period/groupwise tests in sienaTimeTest corrected for the case of non-saturated sets of dummy variables.
    * plot.sienaTimeTest for `pairwise=TRUE` changed so that the warning is avoided.
    * sienaGOF corrected so that again it can handle auxiliary functions referring to more than one varName (such as in mixedTriadCensus).

- 2023-08-01 GitHub, package version 1.3.24

  Changes in RSiena:

  - Corrections:
    * In getEffects, the effects object was given an attribute `version` (was not correct in version 1.3.23).
      (This led in that version to always giving a warning if any interaction effects were specified.)
    * Corrections of implementation of acceptance by sienaGOF of a list of sienaFit objects (was not correct in version 1.3.23).

- 2023-06-29 GitHub, package version 1.3.23.

  Changes in RSiena:

  - New effects:
    * New effects `diffWXClosure`, `sameWWClosure`, `diffWWClosure`, `diffXWClosure`, `sameXWClosure`, `unequalX`.
    * `JoutMix` made available for bipartite dependent networks.
    * For continuous behavior variables depending on a bipartite dependent network, the effect group continuousBipartiteObjective was created, with effects `outdeg`, `outdegSqrt`, and `isolateOut`.
    * `sameXOutAct` and `diffXOutAct` now have a parameter 2 for $\sqrt{}$.
  - Improvements of functionality:
    * The model for continuous behavior variables seems to work now.
    * sienaGOF now also accepts a list of sienaFit objects.
    * In sienaDataCreate, the warning message that there is at least one "upOnly" period now is made for each dependent variable instead of only the last.
    * A warning is given if the used version of the effects object is not current and it contains interaction effects (then it is possible that the interacting effects are chosen incorrectly, even though the `effectName` of the interaction seems OK).

    - Various coding improvements and better texts of warnings and error messages.
    - In the violin plot for sienaGOF, the default bandwidth selection method is used (the use of "nrd" sometimes led to absent plots because of negative bandwidth).
  - Bug corrections:
    - The error was corrected that occurred if `useStdInits = TRUE` in sienaAlgorithmCreate and the effects object includes interaction effects.

- 2023-05-11 GitHub, package version 1.3.22.

  Changes in RSiena:

  - New functionality:
    - For one-mode networks, new options 7, 8, 9, 10 for `modelType` set in sienaAlgorithmCreate; see Section 6.10.
  - Bug corrections:
    - Coding of some "generic effects" was cleaned up and corrected. This led to changes in coding of several distance-2 covariate-related network effects such as `altDist2`, `totDist2` and `altInDist2`; some of these may have been incorrect in earlier versions.
    - In sienaAlgorithmCreate, changed default `prML=2` back to `prML=1`; stop if Maximum Likelihood estimation is attempted for a data set containing more than one dependent variable with `prML=2`.

- 2023-04-22 GitHub, package version 1.3.20.

  Changes in RSiena:

  - Improvements of functionality:
    - The keyword `parameter` in includeInteraction was dropped because it did not have any consequences. The help page for includeInteraction now explains how internal effect parameters for user-defined interactions are determined.
    - The column `dimnames` of the `Simulations` array returned by sienaGOF are set to the names of the elements of the auxiliary function.
    - Standard deviations added to output of descriptives.sienaGOF.
    - Improved error message in `initializeFRAN` in the case of mismatch between effects objects.
    - Warning in sienaAlgorithmCreate if `maxlike` is chosen and `MaxDegree` is specified. This is now also mentioned in the help page for sienaAlgorithmCreate.
  - Bug corrections:
    - updateSpecification (in `effectsMethods`) now also updates internal parameter values.
    - In TriadCensus, the empty network will not lead to an error but be reported with the correct triad census.
    - For `reciAct`, check whether internal parameter is equal to 2 replaced by check whether absolute difference from 2 is less than 0.001.
    - In `phase2.r`, `z$sd` is calculated using `sqrt(pmax(..., 0))` to avoid the extremely rare case of a negative calculated variance.
    - In sienaDataCreate, handling of structurally determined values in `checkConstraints` corrected (thanks to issue raised by Jos Elkink).

  - Improvements of documentation:
    * In the help page for sienaDependent, it is mentioned that if there are one-mode as well as two-mode dependent networks, the one-mode networks should come first.

- 2023-02-07 GitHub, package version 1.3.19.
  Changes in RSiena:
  - New effects:
    * New effect `inPop_dya` .
    * Parameter 2 allowed for `sameXInPop` , `diffXInPop`.
  - Improvements of functionality:
    * coCovar and varCovar now accept vectors with at least one non-missing value, but will stop with an error message if all values are missing.
  - Coding:
    * `siena07internals.cpp` adapted to be compatible with new `clang 16` C++ compiler (thanks to Brian Ripley).
  - Corrections of documentation:
    * Help page for siena07 corrected with respect to `x$lessMem`.

- 2023-01-29 GitHub, package version 1.3.18.
  Changes in RSiena:
  - Improvements of functionality:
    * Additional step type `move` for MH proposal distribution for likelihood estimation (thanks to Charlotte Greenan).
    * Accordingly, parameters changed that are used in sienaAlgorithmCreate for probabilities of moves, now summarized in `prML`; with a new default.
    * List elements `accepts`, `rejects`, `aborts` for sienaFit objects produced by ML estimation improved/corrected by reorganizing them in C++.
    * List element `ac3` (auto-correlations of statistics during Phase 3) added to sienaFit object produced by siena07 for ML estimation.
  - Corrections of documentation:
    * Extended explanation of maximum likelihood estimation.

- 2023-01-06 GitHub, package version 1.3.17
  Changes in RSiena (including those for version 1.3.16):
  - Effects:
    * `inPopIntnX`, `inActIntnX`, `outPopIntnX`, `outActIntnX`, `sameXInPopIntn`, `sameXOutPopIntn`, `sameXInActIntn`, `sameXOutActIntn` restored (these had got lost in some way...).
  - Improvements of functionality:
    * sienaGOF now accepts simulated auxiliary statistics containing missing values. If there are any, this will be reported with a warning if `giveNAWarning` is `TRUE`.
    * sienaDataCreate now also accepts, as "...", a list of such objects.

- 2022-11-27 GitHub, package version 1.3.15
  Changes in RSiena:
  - Correction:
    * Correction: siena08: correct p-value `pTsq` for overall test statistic `Tsq`.
  - Improvements of functionality:
    * Minor improvements in meta.table.
- 2022-10-06 CRAN / GitHub, package version 1.3.14
  Changes in RSiena:
  - Some changes for increasing the compatibility with various hardware platforms.
- 2022-10-06 GitHub, package version 1.3.12
  Changes in RSiena:
  - Updates:
    * Changes to comply with new version of Matrix package.
  - New effects:
    * Several new effects related to primary setting: `nonPCompress`, `primCompress`, `primary`, `primDegAct`, `primDegActDiff`, `primDegActDiffSqrt`, `primDegActSqrt`, `primDegActLog`, `primDegActInv`.
    * `gwdspFB` effect added for two-mode networks.
    * New effects `outAct_ego`, `inAct_ego`,`reciAct_ego`, `toAny`.
    * For effects `to`, `toBack`, `toRecip`, `mixedInXW`, internal effect parameter 3 now specifies truncation of the number of twosteps (change to `MixedTwoStepFunction`).
  - Improvements of functionality:
    * `sigmas` and `meansigmas` added to sienaRI object.
    * Print of standard deviations in the sienaRI object for `printSigma=TRUE` changed to using averages at the variance level.
    * If `returnThetas` in the call of siena07, also simulated estimation statistics during Phase 2 (deviations from targets) are returned.
  - Bug corrections:
    * `universalOffset` initialized as 0; it was earlier initialized as the maximum real number (`NetworkLongitudinalData.cpp`).
  - Corrections of documentation:
    * Modified help page for sienaRI.
    * Small modifications of help page for sienaGOF.
- 2022-05-30 GitHub, package version 1.3.11
  Changes in RSiena:
  - Corrections:
    * Correction in `effects.r` of error that led to warning in R 4.2.0 for multivariate networks.
    * Correction of help page for sienaGOF (`groupName`).

    - Correction of `igraphNetworkExtraction` in the help page for `sienaGOF-auxiliary`.
  - Improvements of functionality:
    - Further explanation of `mixedTriadCensus` in the help page for `sienaGOF-auxiliary`.

- 2022-04-28 GitHub, package version 1.3.10

  Changes in RSiena:

  - Effects:
    - New effects `avInAltW`, `avWInAlt`, `totInAltW`, `totWInAlt` (help from Robert Krause).
    - Corrected implementation of `sharedTo`.
  - Corrections:
    - Bug corrected that occurred when several two-mode networks were included in the dependent variables, with an order restriction between them (`HigherFilter`, `DisjointFilter`).

- 2022-03-18 GitHub, package version 1.3.9.

  Changes in RSiena:

  - Effects:
    - Corrected implementation of `simAllNear` and `simAllFar`.
  - Corrections:
    - Small correction of summary.sienaGOF.
    - Small correction of sienaTimeTest.

- 2022-03-07 GitHub, package version 1.3.8.

  Changes in RSiena:

  - Effects:
    - New effect `avDeg`.
    - Changed default internal effect parameter for `simAllNear` to 2 and for `simAllFar` to 4.
  - Improvements of functionality:
    - Function sienaRI was made available also for two-mode networks (provided the number of second-mode nodes is smaller than the number of actors = first-mode nodes), and corrected for behavioral dependent variables.
    - In sienaTimeTest, added `warn = FALSE` to `varCovar` to avoid warnings.
    - Small improvements to help pages for sienaGroupCreate and sienaGOF.

- 2022-02-16 GitHub, package version 1.3.6.

  Changes in RSiena:

  - Effects:
    - New effects `simAllNear`, `simAllFar`, `absOutDiffIntn`, `avDegIntn`.
    - New effects `recipRateInv`, `recipRateLog` (contributed by Steffen Triebel).
    - Default internal effect parameter for `outOutActIntn`, `outOutAvIntn`, and `both` changed from 2 to 1.

  - Improvements of functionality:
    * Function includeInteraction now also can modify the `initialValue` of an effect; and the order of parameters for this function was changed, bringing it in line with setEffect.
    * Small clarifications of help pages for includeInteraction and setEffect.

- 2021-12-15 GitHub, package version 1.3.5.

  Changes in RSiena:

  - Bug corrections:
    * Corrected the check for effects in `initializeFRAN.r` which led to errors if interaction effects were included, because of the changes to includeInteraction in version 1.3.4.

- 2021-12-08 GitHub, package version 1.3.4.

  Changes in RSiena:

  - New effects:
    * New effects `inRateInv`, `inRateLog` (thanks to Steffen Triebel).
  - Improvements of functionality:
    * When an effects object with interaction effects is printed, the names of the interacting effects are mentioned, and prefixes `int.` and `i3.` were dropped.
    * The check of whether an interaction effect is allowed now is done immediately when creating the interaction effect instead of waiting for its use in siena07.
    * For function sienaGroupCreate some changes were made: if it is applied to a list of length 1, attributes of the single group are not recomputed; if it is applied to a list of length larger than 1, the attributes `range` and `range2` of behavioral variables of individual groups are computed as the range of the unions of the ranges of all the groups. The same is done for covariates.
    * print.sienaGroup slightly extended.
    * Creation of covariates gives a warning (optional) if all values are missing, and also if all non-missing values are the same.
  - Bug corrections:
    * Some bug corrections referring to the use of sienaBayes, which will be integrated later with RSiena.

- 2021-10-08 GitHub, package version 1.3.3.

  Changes in RSiena:

  - New effects:
    * internal effect parameter for diffusion rate effects ('at least p').
    * New effect `outOutAvIntn`.
    * `outOutActIntn` also made available for non-directed explanatory networks and for two-mode dependent networks.
  - Improvements of functionality:
    * `toggleProbabilities` added to output of sienaRI.
    * Trial values of `theta` used during Phase 2 of siena07 added to the object `ans` produced by `sienaFit` as `ans$thetas`.

    - warnings if a data object contains only missings, or only the same value.
  - Bug corrections:
    - if a data set contains a constant covariate, the `simX` effect will not run into an error any more; this is obtained by defining the `range` attribute of the constant covariate as 1 to prevent division by zero, and the `simMean` attribute as 0 instead of `NaN` (sienaDataCreate). This is relevant especially for `sienaGroup` data sets, where covariates might be constant for some of the groups.
  - Corrections of documentation:
    - In `sienaAlgorithmCreate`, the default values of `diagonalize` for MoM is 0.2; this was corrected in the help file.

- 2021-07-29 GitHub, package version 1.3.2.

  Changes in RSiena:

  - New effects: `crprodInActIntn` (thanks to Nynke Niezink), `XXW`.
  - Improvements of functionality:
    - updateTheta also accepts `sienaBayesFit` objects as `prevAns`.
    - Effects of type `creation` or `endow` represented in the output of siena.table by `creation` and `maintenance`, respectively.
  - Bug corrections:
    - If `upOnly` or `downOnly`, the `(out)degree (density)` effect is also excluded for symmetric networks (this was reported by print01Report, but not carried out).
    - Message corrected in sienaDataCreate if there is an attribute `higher`.

- 2021-07-13 R-Forge, RSienaTest version 1.2-30.

  Changes in RSienaTest:

  - sienaBayes: adaptation of `nwarm` and `nmain` at the end of `initializeBayes` dropped.

- 2021-05-02 CRAN/GitHub, package version 1.3.0.

  The cumulative importance of all changes in the past two years (including Generalized Method of Moments and continuous behavioral variables) justifies that the version number now proceeds from 1.2.x to 1.3.0.

  Changes in RSiena:

  - Drop `testthat` in tests.

- 2021-04-30 GitHub, package version 1.2.34.

  Changes in RSiena:

  - New function testSame.RSiena to test equality of parameters.
  - New effects: `avInSim` (thanks to Steffen Triebel), `totInSim, avInSimPopAlt, totInSimPopAlt, constant, avAttHigher, avAttLower, totAttHigher, totAttLower`.
  - Changed effects: endowment and creation types for `avInSim` (brought in line with these types for `avSim`).
  - Improvements of functionality:

    - funnelPlot adapted to lists of sienaFit objects containing some missing estimates or standard errors.
    - plot.sienaGOF: new parameter `position`.
    - Small improvements (constant length of effect names) in meta.table and siena.table.
  - Bug corrections:
    - Restore backward compatibility with respect to checks of `x$gmm`.
    - Correct names reported for tested effects (avoid wrong names being given if there are interactions without main effects).

- 2021-04-14 R-Forge, RSienaTest version 1.2-29.
  Changes in RSienaTest:
  - The pre-warming phase of sienaBayes is interrupted (and followed by the warming phase) if no proposals were accepted during two batches of iterations.

- 2021-03-19 GitHub, package version 1.2.33.
  Changes in RSiena:
  - `configure.ac` changed (thanks to Alvaro Uzaheta). This gives wider possibilities for installation at a variety of platforms.

- 2021-03-18 R-Forge, RSienaTest version 1.2-28.
  Changes in RSienaTest:
  - glueBayes corrected.

- 2021-03-16 GitHub, package version 1.2.32.
  Changes in RSiena:
  - Generalized Method of Moments implemented (Viviana Amati): see https://github.com/stocnet/rsiena/blob/main/docs/manual/Changes_RSiena_GMoM.pdf; new function includeGMoMStatistics, extended functionality of siena07.
  - effectsDocumentation now also includes `gmm` effects (at the bottom).
  - dyadicCov made to accept also changing dyadic covariates.
  - new arguments `plotAboveThreshold` and `verbose` for funnelPlot.
  - Resolved issue with continuous dependent behavior variables (Nynke Niezink).
  - New effects: `homXTransRecTrip`, `toU`.
  - This implied creation of a new effect class dyadANetNetObjective.
  - `sqrt` versions for parameter 2 for the effects `to`, `toBack`, `toRecip`, `from`, `fromMutual`.
  - Reinstated effect `MixedInXW`, also with `sqrt` version for parameter 2.
  - Dropped effect `to.2` (identical to '`to`') and `MixedInWX` (identical to '`toBack`').
  - Description of `toBack` and `toRecip` in manual.
  - Correction in `phase3.2` of a bug that sometimes led to an error message if `simOnly`.
  - Display of deviations from targets changed to after subtraction of targets.
  - Stop if no parameters are estimated and `simOnly` is FALSE.
  - Vignette dropped; it is available on the website.

  - Require `R >= 3.5.0`.
  - xtable added to `Imports` (used to be in `Suggests`).

- 2020-12-10 GitHub, package version 1.2.29.
  Integration of GitHub and what was earlier at R-Forge.

  Changes in RSiena:

  - New effects (due to Christoph Stadtfeld): `transtrip.FR, transtrip.FE, transtrip.EE, WWX.EE, WWX.FR, WXX.FE, WXX.ER, XWX.ER, XWX.FE, to.2, toBack, toRecip`.
  - New functions meta.table and funnelPlot.
  - In sienaAlgorithmCreate, use the definitions for `projname = NULL` also if any environment variable `_R_CHECK*` is set.
  - Adapted filter `disjoint` so that it operates correctly also when the network is symmetric. Consequence: constraint that two networks are disjoint operates correctly also when one of the networks is symmetric and the other is not.
  - Adapted filter `higher` so that it operates correctly also when the other network is symmetric. Consequence: constraint that one network is at least as high as another network operates correctly also when the higher networks is symmetric and the other is not.
  - In CheckConstraints, used in sienaDataCreate, the requirement was dropped that the two networks have the same symmetry property; and for `higher` it is required that if the lower network is symmetric, the higher network is also symmetric.
  - In sienaDataConstraint, if type is `disjoint` or `atLeastOne`, the constraint is also implemented for the pair `(net2, net1)`.
  - New parameter `tested` in sienaGOF.

- 2020-09-16 R-Forge Revision 350, package version 1.2-27.

  Changes in RSiena and RSienaTest:

  - `tcltk` no longer a requirement; if not available, batch mode is followed for siena07 (with help by James Hollway).
  - New effect `transTripX`.
  - For effect `from.w.ind`, option `parameter = -1` added.
  - The `to` effect is an ego effect.
  - For siena.table, some of the `effectNames` changed to nice strings, so that LaTeX can run without errors on the table produced if `type = "tex"`.
  - The sienaMeta object produced by siena08 now has IWLS estimates more easily accessible, as `object$muhat` and `object$se.muhat`.
  - Error message in sienaTimeTest for sienaFit objects produced with `lessMem=TRUE`.
  - More extensive error message for error in named vectors in algorithm object (`checkNames` in initializeFRAN).
  - For sienaDataCreate: more extensive error message, and `class(...)` replaced by `class(...)[1]`.
  - multiplication factor `mult` added to print.sienaAlgorithm if `maxlike`.
  - sienaAlgorithmCreate: requirements for `mult` corrected in help page.

  Changes in RSiena:

- Vignette basicRSiena added (was earlier available as a script); thanks to James Hollway.

Changes in RSienaTest:

- Improved initialisation in sienaBayes: initial scale factors (squared) and initial parameter estimate ('Kelley').
- Allow parameter targetMHProb of sienaBayes to have length 2, applying separately to the random parameters and the fixed parameters.
- Test on class()== replaced by inherits in sienaBayes.
- Another attempt to discard the objects zn and zsmall remaining after operation of sienaBayes.

- 2020-04-08 R-Forge Revision 348, package version 1.2-25.

  Changes in RSiena and RSienaTest:

  - Correction of sienaDataCreate for actor covariates (gave a warning).
  - Auxiliary function dyadicCov for sienaGOF now can handle missings, does not return a named value for frequency of zeros, and returns the frequencies without any division. The help page indicates how this function can be used to check fit for ego-alter combinations of monadic variables.
  - descriptives.sienaGOF also shows the exceedance probabilities of the observed statistics.
  - The NULL default values for modelType and behModelType are implemented already at the R side instead of waiting until the information comes to C++.
  - Some extra information in help page for siena07.

  Changes in RSienaTest:

  - Error corrected that occurred in sienaBayes, extract.sienaBayes, and extract.posteriorMeans for data with multiple periods.

- 2020-02-16 R-Forge Revision 347, package version 1.2-24.

  Changes in RSiena and RSienaTest:

  - New auxiliary function dyadicCov for sienaGOF.
  - Correction of error in modelType when one-mode as well as two-mode networks are used.
  - Added startingDate in siena08 to the object produced.

  Changes in RSienaTest:

  - Corrected siena.table for sienaBayesFit objects; added startingDate in sienaBayes to the object produced.

- 2020-01-12 CRAN Siena version 1.2-23.

  Changes in RSiena and RSienaTest:

  - Changed extension names of output files from .out to .txt.
  - Option projname = NULL in sienaAlgorithmCreate. This option is used for all examples in RSiena where siena07 is called, to restrict file writing for examples to the temporary directory.

- 2020-01-16 R-Forge Revision 345, package version 1.2-22.
  Changes in RSiena and RSienaTest:
  - Tested effects reported in print of Multipar.RSiena and score.Test.
  - Reordered effects object so that `reciAct` comes after instead of before `inAct` and `inAct.c`.
- 2019-12-15 R-Forge Revision 344, package version 1.2-19.
  Changes in RSiena and RSienaTest:
  - New effects `avGroupEgoX`, `outIso`.
  - Starting date of estimation reported in `siena.table(..., type = "tex", ...)`.
  - Handling of missings in changing covariates corrected (?).

  Changes in RSiena:
  - Changes of RSienaTest version 1.2-18 (see below) ported to RSiena. The main changes are the introduction of the option of continuous dependent behavior variables and allowing `imputationValues` also in sienaDependent (both by Nynke Niezink).

  Changes in RSienaTest:
  - In sienaBayes: more precise check that `prevAns` has same specification as `effects`. `priorSigEta` added to `sienaBayesFit` object. `dimnames(ThinParameters)[[3]]` defined as effect names; ridge for prior variance of rate parameters if `priorRatesFromData=1` or 2 decreased from 0.5 to 0.01.
  - sienaBayes: `zm` and `zsmall` are removed at the end.
  - plotPostMeansMDS also shows a title, and coordinates show the dimensions of the MDS solution.
  - The value returned by plotPostMeansMDS contains not only the plot coordinates, but also the similarities (correlations) between the groups.
  - glueBayes: added `z$varyingInEstimated`, `z$varyingNonRateInEstimated`, `z$ratesInVarying` to the object produced. (Else extract.sienaBayes is not going to work properly.)
  - glueBayes: if one of the sampling parameters is different, give a warning instead of a stop.
  - Corrected error in names of array returned by extract.posteriorMeans.
  - New parameter `excludeRates` in extract.posteriorMeans, plotPostMeansMDS.
  - Use parameter `pmonly` also in plotPostMeansMDS.
- 2019-10-16 R-Forge Revision 341, RSienaTest version 1.2-18.
  Changes in RSienaTest:
  - Continuous dependent behavior variables implemented (Nynke Niezink). This implies new effect types `continuousFeedback`, `continuousIntercept`, `continuousOneModeObjective`, `continuousRate`, `continuousWiener`, `unspecifiedContinuousInteraction`.
  - `imputationValues` allowed in sienaDependent (Nynke Niezink).
  - New effect `outMore`.

  - component `startingDate` added to sienaFit object; this date is reported in siena.table(..., type = "tex", ...).
  - Object names are given in sienaFit.print if `simOnly`.
  - Speeded up calculation of IndegreeDistribution and OutdegreeDistribution for sienaGOF if there are no missings or structurals.
  - `regrCoef` and `regrCor` added to the sienaFit object also when not `dolby`.
  - Immediate stop if `useCluster` and `returnChains` both are used (in this case, no chains would be returned anyway).
  - sienaDataCreate: more informative message in case of constraints.
  - Small improvements in many help pages.
  - Corrected error in names of array returned by extract.posteriorMeans.
  - New parameter `excludeRates` in extract.posteriorMeans, plotPostMeansMDS.
  - Use parameter `pmonly` also in plotPostMeansMDS.

- 2019-05-20 R-Forge Revision 340, package version 1.2-17.

  Changes in RSiena and RSienaTest:

  - New effects `outAct.c`, `inAct.c`, `outPop.c`, `inPop.c`, `degPlus.c`. These are centered versions, which might have better convergence properties than their namesakes without the `.c`.
  - Effects `antiInIso` and `antiInIso2` implemented also for non-directed networks.
  - Effects `outTrunc`, `outTrunc2`, `outSqInv`, `isolateNet`, and `degPlus` got endowment effects.
  - For `inPopIntn`, `outPopIntn`, `inActIntn`, and `outActIntn`, added 'centered' to the statistic name.
  - plot.sienaGOF has a new parameter `fontsize`; the dots `...` will also accept parameters `cex`, `cex.main`, `cex.lab`, `cex.axis`, as used more generally for plotting in R.
  - In case of simulation as indicated by `x$simOnly = TRUE`, the object produced by siena07 now also contains the standard errors of the mean, called `estMeans.sem` (see the help page for siena07).
  - On various places diagnostic output messages were relabeled to 'message' or 'warning', which implies that they can be turned off by commands such as `suppressMessages` or `suppressWarnings`.

  Changes in RSienaTest:

  - In sienaBayes:
    `prevAns` now is allowed to have a different specification of random effects than `effects`;
    it is not allowed to use `prevAns` and `prevBayes` objects simultaneously;
    some further small algorithm changes (see CHANGELOG).

- 2019-02-26 R-Forge Revision 339, package version 1.2-16.

  Changes in RSiena and RSienaTest:

  - New effects `maxAAlt, minXAlt` (thanks to Per Block, Marion Hoffman, Isabel Raabe, and Kieran Mepham), `outIsolate`.
  - Effect name of `isolate` effect (not its `shortName`) changed to `in-isolate`.

- New parameter `thetaValues` in siena07, allowing simulations in Phase 3 with varying parameters according to matrix input. Corresponding changes in print.sienaFit.
- sienaRI: earlier check for bipartite dependent variables; if so, stop.
- Argument `verbose` added to includeEffects and setEffect. (It existed already for includeInteraction.)
- Wald.RSiena, Multipar.RSiena and score.Test also produce one-sided test statistic if `df = 1`.
- objects produced by Wald.RSiena, Multipar.RSiena and score.Test have class sienaTest. These now also have a print method.

Changes in RSienaTest:

- Allow arbitrary number of interactions in sienaBayes (via getEffects).
- simpleBayesTest and multipleBayesTest corrected; they were wrong for models with user-defined interactions.
- glueBayes extended (to allow good use for models with fixed parameters) and check for prior rates relaxed (to allow `priorRatesFromData=2` leading to different prior rates.)
- New default `prevBayes$nwarm >= 1` for `newProposalFromPrev` in sienaBayes.
- siena.table for sienaBayes results extended with between-groups s.d.
- Argument `verbose` added to extract.posteriorMeans.
- sienaBayes: check that `prevAns` is a sienaFit object, and not sienaBayes; totally skip initial multi-group estimation if `prevAns` is given with `((prevAns$n3 >= 500) & (!usePrevOnly))`.

- 2019-01-15 R-Forge Revision 338, package version 1.2-15.

  Changes in RSienaTest:

  - New effects `sameXReciAct` and `diffXReciAct`.
  - Increased burn-in for ML simulations in siena07.
  - siena.table adapted for sienaBayes results.
  - New functions shortBayesResults and plotPostMeansMDS.
  - extract.posteriorMeans works also for an object saved as `PartialBayesResult` (which has `NA` results for as yet unfinished runs).
  - Examples added to help page for `print.sienaBayesFit` (in `dontrun`...).

- 2018-12-05 R-Forge Revision 337, packages version 1.2-14.

  Changes in RSiena and RSienaTest:

  - In the help page of sienaAlgorithmCreate, the possibility is mentioned that the multiplication factor (parameter `mult`) can be specified as a vector. This allows the possibility for ML estimation to specify the multiplication factor separately for periods $\times$ waves.

  Changes in RSienaTest:

- sienaBayes:
  the possibility now exists to specify prior variances for non-varying parameters, which can be meaningful especially for effects of group-level variables;
  a new, hopefully better, initial value is used, defined as a precision weighted mean of the multi-group MoM estimate and the prior mean;
  number of subphases for initial global parameter estimation by MoM decreased to 2;
  prewarming phase introduced before the first improveMH; checks for dimensions of prior mean and covariance matrix put earlier in the initialization phase.
- extract.sienaBayes corrected.

- 2018-10-30 R-Forge Revision 336, packages version 1.2-13.
  Changes in RSiena and RSienaTest:
  - Correct error in sienaGroupCreate for non-centered actor covariates.
  - Correct error in print01Report that occurred for changing dyadic covariates given as lists of sparse matrices.
  - Also get simulated dependent behavior variables for `siena07(..., returnDeps = TRUE, ...)` for ML estimation (see help page).
  - siena.table corrected for data sets with several dependent variables.
  - For siena08: new parameters `which` and `useBound` in plot.sienaMeta; new parameter `reportEstimates`, allowing to reduce the output produced.
  - updateSpecification: also update `randomEffects` column.
  - More explanation of sparse matrix input in the help page for sienaDependent.

  Changes in RSienaTest:
  - sienaBayes: new parameters `proposalFromPrev`, which allows taking proposal distributions from `prevBayes` object;
    `incidentalBasicRates` resuscitated;
    allow fixed rate parameters, with `priorRatesFromData = -1`;
    changed initial values of scale factors for proposal distributions;
    new parameters target and `usePrevOnly`;
    in case `prevBayes` is used, parameters `nrunMHBatches, nSampVarying, nSampConst,` and `nSampRates` in the function call of sienaBayes supersede those in the `prevBayes` object;
    correct bug for three-effect interactions;
    copy parameters `modelType, behModelType, MaxDegree, Offset, initML`, from parameter `algo` to the algorithms created within sienaBayes;
    more extensive checking of smallest eigenvalue of covariance matrix;
    for `priorRatesFromData = 1` or 2, the resulting prior Covariance matrix for `priorKappa != 1` was incorrect; this was corrected;
    reported timing changed to elapsed system time.

- 2018-05-13 CRAN, RSiena version 1.2-12.
- 2018-05-06 R-Forge Revision 335, packages version 1.2-11.
  Changes in RSiena and RSienaTest:
  - New effects `gwdspFF` and `gwdspFB`.

- Effects simEgoInDist2 and simEgoInDist2W for two-mode networks were corrected. Perhaps simEgoDist2 and simEgoDist2W were broken in a previous version; if so, that was now corrected.
- Added sqrt version for reciAct, obtained for parameter = 2.
- Allowed the value parameter = NULL for setEffect and includeInteraction, meaning that no change is made for the internal effect parameter. For setEffect this is the new default, implying that when starting the default values from allEffects.csv are used, just like in includeEffects (where no parameter can be given).
- New auxiliary functions for sienaGOF: Triad Census and mixedTriadCensus (contribution by Christoph Stadtfeld).
- triad census from igraph added to help page of sienaGOF-auxiliary.
- improved sienatable output for type = "html": added rules = none, frame = void to general options; changed minus to – ; added column for asterisks to have better alignment for estimates.
- Stop cluster also for a user interrupt in siena07.
- Extension of help page for siena07 by mentioning functions for accessing simulated networks for ML.

Change in RSienaTest:

- extract.sienaBayes: error corrected that occurred if called with extracted = "all" but there are no varying, or no non-varying parameters.

- 2018-03-24 R-Forge Revision 334, packages version 1.2-10.
  Change in RSiena and RSienaTest:
  - Example in help file siena07 for accessing generated networks in case of ML estimation/simulation.
- 2018-03-21 R-Forge Revision 332, packages version 1.2-9.
  Changes in RSiena and RSienaTest:
  - Added effects avExposure, totExposure, infectDeg, susceptAvCovar, infectCovar for symmetric networks, in new effect group covarBehaviorSymmetricRate.
  - New effects degAbsDiffX, degPosDiffX, degNegDiffX, degAbsContrX, degPosContrX, degNegContrX.
  - Effects XWX, XWX1, and XWX2 enabled for bipartite networks, in new effect group dyadBipartiteObjectiv
  - Added parameter = 2 for FFDeg, BBDeg, FBDeg, FRDeg, BRDeg.
  - Corrected altInDist2, totInDist2 for two-mode networks.
  - Dropped some duplications in effectsDocumentation.
  - Modification of effect outTrunc2. Perhaps this was superfluous.
  - Export of last simulated state from ML simulations if returnDeps.
  - Added endowment and creation effects for maxAlt and minAlt.
  - Error messages for non-character nodeSets in functions sienaDependent, coCovar, varCovar.
  - Improved way of handling missings for distance-2 network effects.

- Added components `requestedEffects`, `theta`, and `se` to `siena08` (`theta` and `se` are the ML estimates).
- Error in summary of sienaMeta object corrected: for the estimated mean parameter the two-sided p was announced but a one-sided p was given. Also the ML results under normality assumptions were copied from print.sienaMeta to print.summary.sienaMeta.
- For non-directed networks, the initial model now contains only basic rates and degree effect.
- Added reference to `DESCRIPTION` and citation info in `\inst`, so `citation("RSiena")` now gives useful information.
- The dropping of the exclusion of effects if the variance of a covariate is 0, or a covariate has only two values, of version 1.1-306, was taken further (was incomplete).
- Message if attributes `higher, disjoint`, or `atLeastOne` are `TRUE` at the end of sienaDataCreate.

Changes in RSienaTest:

- new function extract.posteriorMeans for sienaBayes results.
- Restrict check in sienaBayes of maximum estimated parameter value after initialization to non-fixed effects.
- Correct construction of groupwise effects object in sienaBayes so that this will work also when evaluation effects for density and/or reciprocity are not included; and when there are interaction effects for which the main effects are not included.
- Corrections for print and summary of sienaBayes objects. In print.sienaBayesFit, include fixed parameters and give credibility intervals for rate parameters; include variance parameters; allow shorter `ThinParameters`; print objects returned through `partialBayesResult.RData` (by adding `na.rm = TRUE` to quantile).
- multipleBayesTest corrected (there was an error for testing 2 or more linear combinations simultaneously) and adapted for cases with fixed parameters; adapted help file text.
- All remaining parts of rsiena01gui removed.
- As a compensation of this, sienaDataCreateFromSession exported, and a more informative help page written.

- 2017-09-09 R-Forge Revision 318, packages version 1.2-4. Changes in RSiena and RSienaTest:
  - Longer Description field in `DESCRIPTION`, as per CRAN suggestions.
  - Correction of print.sienaFit, repairing the option `include = FALSE` in setEffect and includeEffects.

  Changes in RSienaTest:
  - Extra line in example for `sienaBayes.Rd`.
- 2017-09-08 R-Forge Revision 316, CRAN package version 1.2-3.
  Changes in RSiena and RSienaTest:
  - totAlt also for non-directed networks.
  - Further corrections to makeRSienaacceptable for CRAN.
- 2017-09-04 R-Forge Revision 312.
  Changes in RSiena and RSienaTest:

- Score test and sienaGOF corrected for the case that some parameters are fixed and not tested.
- New function score.Test. This allows, for a sienaFit object in which some effects were tested with `test = TRUE`, to get the results of the score-type test for some or all of the parameters tested. When some effects are tested with `test = TRUE`, the results of the score test are also presented when printing the estimation result.
- Argument `varName` added to updateTheta. This allows the update to be restricted to one or more of the dependent variables.
- Operation of option '`absorb`' (`behModelType` = 2) corrected.
- In sienaAlgorithmCreate, for parameters `Offset, MaxDegree, modelType`, and `behModelType`, require that these are named vectors with the names of the dependent variables, or NULL; this is checked in initializeFRAN.
  The use of non-named vectors for these, in the case of using multiple processes may have led to errors in earlier versions.
  Note that the default for `MaxDegree` now is not `MaxDegree = 0` but `MaxDegree = NULL`.
- New effects `sameXInPopIntn, sameXOutPopIntn, sameXInActIntn, sameXOutActIntn`.
- For `networkModelType` 3 (Initiative, `AAGREE`), an offset is added to the confirmation model; this is taken from `Offset` in the algorithm object.
- For effects `inPopX, outPopX, inActX, outActX, sameXInPop, sameXOutAct, diffXInPop, diffXOutAct, homXInPop, homXOutAct, altXOutAct, diffXOutAct`, changed `interaction2` to `''` (was `'1'`, erroneously) and default parameter to 1 (AllEffects.csv).
- Additional parameter `dropRates` in print.sienaEffects (useful for sienaBayes with many groups).
- In print.sienaEffects, omit last remark about random effects if only one line is printed.
- Corrected use of `modeltype` (initializeFRAN.r, CInterface.r),
- Change in print.sienaAlgorithm for `modelType`.
- Added an example for multiple processes in sienaGOF.Rd, and took out verbose reporting from sienaGOF in case of multiple processes.

Changes in RSiena:

- New parameter `OffSet` in sienaAlgorithmCreate.
- Registration of native routines (requirement for CRAN).

Changes in RSienaTest:

- Parameter `UniversalOffset` of sienaAlgorithmCreate renamed to `Offset`.
- Added posterior variances to print.sienaBayesFit.
- Correction of error in sienaBayes that could lead to errors in results of estimations with data sets for multiple dependent variables (networks and/or behavior) with model specifications that contain an interaction effect for a dependent variable that is not the last of the dependent variables.
- Various changes in sienaBayes for less memory use and avoiding crashes in some situations.

- 2017-05-08 Revision 306.

  Changes in RSiena and RSienaTest:

- New function updateSpecification to includes in an effects object a set of effects that are included in another effects object.
- New effects `inPopX`, `outPopX`, `inActX`, `outActX`, `sameWXClosure`, `degPlus`, `absDiffX`, `avAltPop`, `totAltPop`, `egoPlusAltX`, `egoPlusAltSqX`, `egoRThresholdX`, `egoLThresholdX`, `altRThresholdX`, `altLThresholdX`.
- `outOutAss` dropped for symmetric networks; only `outInAss` remains.
- `egoX` effect has `interactionType = "ego"` also for symmetric networks.
- The exclusion of effects if the variance of a covariate is 0, or a covariate has only two values, is dropped (these effects have no meaning, but their exclusion was a potential nuisance for meta-analyses).
- Description of `gwespFB` and `gwespBF` corrected.
- includeInteraction has an additional parameter `random`.
- siena08 now also accepts a list for ...
- Argument `behModelType` added to sienaAlgorithmCreate. For `behmodelType=2`, the 'absorbing option' is chosen in the model for behavioral dependent variables; see Section 6.11 of this manual.
- Indication of effect parameters dropped in names for `altInDist2` and `totInDist2` (they have no effect parameters).
- `ModelType` now is specific to the dependent variable: given as a named integer vector in sienaAlgorithmCreate.
- sienaAlgorithmCreate,siena07: new option `lessMem`, reducing storage in siena07 by leaving out `z$ssc` and `z$sf2` from the object produced; but these are used by sienaTimeTest and sienaGOF, so running those functions will be impossible for sienaFit object obtained with `lessMem = TRUE`.
- extended information in print.sienaAlgorithm.
- Modified check for singular covariance matrix after Phase 3.
- Warning if includeEffects is used with parameter `random`.
- Warning for impossible or zero changes if maximum likelihood (see Section 7.11, and News page of Siena website).
- Some parts dropped from the object produced by siena07 to reduce memory use.

Changes in RSienaTest:

- sienaBayes: various changes to save memory (thanks to Ruth Ripley); improved reporting of groups with no changes; warning for impossible changes, see Section 12.4.2; if `priorRatesFromData = 2`, change to different robust covariance matrix estimator when this is necessary (i.e., for small number of groups); in print.summary, also report `nImproveMH`; a few lines added to help file.
- print.sienaEffects gives dimensions of `priorMu` and `priorSigma` if `includeRandoms`.

- 2016-10-13, 14 Revision 303, 304.

  Changes in RSiena and RSienaTest (George Vega Yon):

  - New parallellization option -cl- for siena07.

- 2016-10-09 Revision 301.
  Changes in RSiena and RSienaTest:
  - New effects: `sameXCycle4, homCovNetNet, contrastCovNetNet, covNetNetIn, homCovNetNetIn, contrastCovNetNetIn, inPopIntnX, inActIntnX, outPopIntnX, outActIntnX`.
  - Changes permitting the 4-cycles effects for larger and denser networks.
  - Dropped `cl.XWX` effect from two-mode – one-mode coevolution (did not belong).
  - `egoSqX` is an ego effect.
  - Added `cycle4` for one-mode networks.
  - Added `outAct, outInAss` for symmetric networks.
  - SienaRI: Structural zeros and ones are excluded from the calculations; added option `getChangeStats`; row names given to matrices that have rows corresponding to effects; adapted so that it runs for models with only 1 parameter; adapted so that for a bipartite dependent variable it does not crash.
  - Warning if includeEffects is used with parameter `parameter`.
  - Small additions to print.sienaAlgorithm.
  - Clearer output for `MaxDegree` in print.sienaFit.
  - Correction of how effect parameter for `outInAss` for 2-mode networks is reported.
- 2016-08-17 Revision 296.
  Changes in RSiena and RSienaTest:
  - Warning if includeInteraction is used for more interactions than available given parameters `nintn` and `behNintn`.
  - Deleted session parameter from print01Report.
  - Corrected `cycle4` effect for parameter = 2 (sqrt version).
  - Additional auxiliary function CliqueCensus in help page sienaGOF-auxiliary.
  - Error corrected in `DyadicCovariateAvAltEffect.cpp`.
- 2016-05-28 Revision 294.
  Changes in RSiena and RSienaTest:
  - The manual was taken out of the installation; it still is in the source code distribution as RSienaTest\doc\RSiena_Manual.tex. It still is publicly it can be downloaded from `http://www.stats.ox.ac.uk/~snijders/siena/RSiena_Manual.pdf`.
  - New effects `totAltEgoX`, `totAltAltX`, `egoSqX`, `diffX`, `diffSqX`, `egoDiffX`, `avAltW`, `totAltW`, `avSimW`, `totSimW`, `jumpFrom`, `jumpSharedIn`, `mixedInXW`, `mixedInWX`, `avWalt`, `totWAlt`.
  - `inActIntn` also implemented for two-mode dependent networks.
  - Endowment and creation effects were added for `inAct`, `inActSqrt`, `outAct`, `outActSqrt`.

  Changes in RSiena:
  - (siena01Gui) and sienaDataCreateFromSession dropped.

  Changes in RSienaTest:

  - Change in endowment effect estimation for `avAlt` effect.
  - New effects `simmelian`, `simmelianAltX`, `avSimmelianAlt`, `totSimmelianAlt`.

- 2016-02-03 Revision 291.

  Identical to 290, but now there is a Mac version.

- 2016-02-01 Revision 290.

  Changes in RSiena and RSienaTest:

  - New effects `FBDeg`, `FRDeg`, `BRDeg` (`RFDeg` was mentioned earlier but was not implemented; its place is now taken by `FRDeg`), `gwespFFMix`, `gwespBBMix`, `gwespBFMix`, `gwespFBMix`, `gwespRRMix`, `gwespMix`.
  - Corrected the omission of the check for positive derivative matrix at the end of Phase 1 for effects with fixed parameters.
  - Added parameters `fix`, `test`, `parameter` to includeInteraction.
  - More helpful error message for incorrect nodesets in sienaDataCreate; extended help pages for sienaDataCreate and sienaDependent.
  - Correction to allow estimation for a one-dimensional parameter.
  - `fromBayes` bug corrected.

  Changes in RSienaTest:

  - sienacpp added (programmed by Felix Schönenberger); this includes gmm estimation (Amati - Snijders).
  - sienaBayes: option `initML` added; check for zero distances; corrected function `improveMH` in initialization.
  - print.multipleBayesTest: option `descriptives` added.
  - glueBayes: added `p1` and `p2` to created object.

- 2015-09-10 Revision 289.

  Changes in RSiena and RSienaTest:

  - New defaults for siena07 in sienaAlgorithmCreate: doubleAveraging = 0, diagonalize = 0.2 (for MoM).
  - Improved one-step approximations to expected Mahalanobis distances in sienaGOF (control variates for score function).
  - Permit 3-way interactions with one ego and two dyadic effects (`initializeFRAN.r`) (this was erroneously not allowed).
  - New effects `Jin, Jout, JinMix, JoutMix, altXOutAct, doubleInPop, doubleOutAct.`
  - print01Report now reports in-degrees also for two-mode networks.
  - Better error handling for sienaTimeTest and scoreTest.
  - `inOutAss` is dyadic.
  - Corrected `effectName` and `functionName` of `inPopIntn, outPopIntn, inActIntn,` and `outActIntn` (‘`in`’ and ‘`out`’ were missing).
  - Check for positive derivative matrix at the end of Phase 1 (non-positive estimated derivatives lead to repeating a prolonged Phase 1) omitted for effects with fixed parameters.

Changes in RSiena:

- New effects `homXOutAct, FFDeg, BBDeg, RFDeg, diffXTransTrip` (ported from RSienaTest).
- `sameXInPop` and `diffXInPop` also added for two-mode networks; but they are not dyadic!
- In names of behavior effects and statistics dropped the (redundant) parts "behavior" and "beh.".

Changes in RSienaTest:

- New function extract.sienaBayes.
- sienaBayes: options diagonalize = 0.2, doubleAveraging = 0 for estimation of initial models in initialization phase; save initial results in case of divergence during initialization phase; check for large initialEstimates done only for non-rate parameters.

- 2015-07-18 Revision 288.

  Changes in RSiena and RSienaTest:

  - plot.sienaRI: new parameter `actors`; proportions with piechart improved (hopefully); effect of parameter `radius` changed.
  - siena.table does no more produce the double minus sign in html output.

  Changes in RSiena:

  - Correction of error for two-mode networks in sienaGOF.

  Changes in RSienaTest:

  - New effects `homXOutAct`, `FFDeg`, `BBDeg`, `RFDeg`, `diffXTransTrip`.
  - `sameXInPop` and `diffXInPop` also added for two-mode networks; but they are not dyadic!
  - In names of behavior effects and statistics dropped the (redundant) parts `behavior` and `beh.`.
  - sienaBayes: new parameter `nSampRates`; correction in use of `prevBayes`; more efficient calculation of multivariate normal density.
  - Small changes in HowToCommit.tex.

- 2015-05-21 and 2015-06-02 Revision 286.

  Changes in RSiena and RSienaTest:

  - SienaRIDynamics dropped from RSiena, it still has an error (retained in RSienaTest).

  Changes in RSienaTest:

  - Correction of error for two-mode networks in sienaGOF.

- 2015-05-20 Revision 285.

  Changes in RSiena and RSienaTest:

  - `tmax` added to sienaFit objects and `tconv.max` mentioned in print.sienaFit.
  - sienaAlgorithmCreate has new arguments `n2start`, `truncation`, `doubleAveraging`, `standardizeVar`; this leads to various changes in `phase2.r`.
  - Diagonalization corrected (matrix transpose) (`phase2.r`).

- When there are missings in constant or changing monadic covariates, and centered=FALSE for their creation by coCovar or varCovar, the mean will be imputed (used to be 0, which was an error). For changing covariates, this is the global mean.
- In coCovar and varCovar there is a new argument `imputationValues`, which are used (if given) for imputation of missing values. Like all missings, they are not used for the calculation of the target statistics in the Method of Moments.
- New effects: `outOutActIntn, toDist2, from.w.ind`.
- In the target statistic for the `higher` effect, contributions for value(ego)=value(alter) are now set appropriately at 0.5 (was 0).
- New effectGroup tripleNetworkObjective (allEffects.csv, effects.r) (see earlier in this manual for its characteristics).
- Decent error message when there are (almost) all NA in the dependent behavioural variable (effects.r, function getBehaviorStartingVals).
- The centering within effects for similarity variables at distance 2 now is done by the same similarity means as for the `simX` effect (`attr(mydata$cCovars$mycov, "simMean")`, etc.).

- 2015-04-02 Revision 284.

  Changes in RSiena and RSienaTest:

  - New effects: `simEgoDist2, simEgoInDist2, simEgoDist2W, simEgoInDist2W, sameXOutAct, diffXInPop, diffXOutAct`. The centering for the similarity measure in effects such as simEgoDist2 and simDist2 is not yet clear (this affects only the outdegree parameter).
  - Relevant for creating new effects: New effect groups covarABNetNetObjective, covarANetNetObjective, and covarBNetNetObjective. See SienaSpec.tex/pdf, section 4.9: covarNetNet.
  - Bug corrected that occurred in print01Report for a sienaGroup object where the component objects have constant dyadic covariates.
  - When a statistic is not plotted in plot.sienaGOF because its variance is 0, a note about this is printed to the screen.
  - Minimum and maximum of plotted region in plot.sienaGOF is calculated without taking into account non-plotted statistics.
  - Bug corrected with includeTimeDummy for `timeDummy` greater than or equal to 10 (sienaTimeTest.r).
  - In case of collinear parameter estimates, standard errors are reported as NA.
  - Arguments `main` and `ylab` dropped from plot.sienaGOF; they did not work, and their functionality now is covered by the ... argument (so using `main` and `ylab` as arguments now should work). (Thanks to David Kavaler.)

  Changes in RSienaTest:

  - sienaBayes: correction in initialization of truncation rate parameters based on prior; error corrected for sampling constant parameters.

  Changes in RSiena:

  - Parameter `reduceg` added in sienaAlgorithmCreate for use in siena07, like in RSienaTest.

- 2014-12-11 R-Forge Revision 282.
  Changes in RSiena and RSienaTest:
  - Effects `cl.XWX` and `cl2.XWX` corrected (thanks to Christoph Stadtfeld).
  - `interactionType` of `gwesp..` effects was made `dyadic`.
  - Some layout changes in warning message in siena07.
  - New effects `reciPop, reciAct, in3Plus, maxAlt, minAlt, transTrip1, transTrip2`.
  - Effect `antiInIsolate2` got alias `in2Plus`.
  - `inPop` is dyadic effect (except for non-directed networks) (it is $\sum_j x_{ij}\{\sum_{h\neq i} x_{hj} + 1\}$).
  - `egoX` added as effect for non-directed networks (can be important for representing effects of group-level covariates in multi-group analyses).
  - Components `IActors` and `expectedI` added to sienaRI and print.sienaRI.
  - The check for `MaxDegree` when running siena07 now works properly also for sienaGroup objects.
  - Manual introduces the term *elementary effects*.

  Changes in RSienaTest:
  - sienaBayes: the stop caused by singularity of the precision matrix after the multi-group estimation now is circumvented; still a warning is printed to the screen.
  - sienaBayes: option `priorRatesFromData` changed to values 0-1-2, with 0 = former `FALSE`, 1 = former `TRUE`, 2 = robust estimation of prior for rate parameters from estimates at the end of initialization phase.
  - Correction of print.summary.sienaBayesFit for models with more than one dependent variable.
- 2014-11-19 R-Forge Revision 280. Changes in RSienaTest:
  - Small changes in help pages for sienaGOF and for sienaCompositionChange.
  - New parameters `nSampVarying` and `nSampConst` in sienaBayes.
- 2014-11-13 R-Forge Revision 279.
  Changes in RSiena and RSienaTest:
  - Effect `AltsAvAlt` renamed to `avXAlt`.
  - Effects object no longer used as argument for print01Report.
  - A lot of new effects: `sameXInPop, transRecTrip2, totAlt, avInAlt, totInAlt, totRecAlt, totXAlt, avXInAlt, totXInAlt, avAltDist2, totAltDist2, avTAltDist2, totAAltDist2, avXAltDist, totXAltDist2, avTXAltDist2, totAXAltDist2, avInAltDist2, totInAltDist2, avTInAltDist2, totAInAltDist2, avXInAltDist2, totXInAltDist2, avTXInAltDist2, totAXInAltDist2, XWX1, XWX2, cl.XWX1, cl.XWX2`.
  - Endowment and creation effects added for `gwesp...` effects.
  - Some meaningless effects for two-mode networks dropped.
  - For non-invertible covariance matrices at the end of siena07, give diagnostic for the linear combination that gives trouble.

- Correction of igraphNetworkExtraction in the help page for sienaGOF-auxiliary (the earlier version dropped isolated nodes from simulated networks).
- In the help page for sienaGOF-auxiliary.Rd, the example of constraint is replaced by the example of eigenvector centrality (because constraint is undefined for isolated nodes, leading to computational problems).
- Set diagonal of observed networks to 0 in sparseMatrixExtraction.
- sienaRIDynamics restored, after corrections.
- 'file" parameter of sienaRI dropped (implied platform dependence).
- Section in manual about user-defined interaction effects updated.
- Parameter `showAll` added to descriptives.sienaGOF.
- Small correction of print.siena (reporting uponly/downonly).
- Some changes in print.sienaAlgorithm.
- Check in siena07 for incorrect `MaxDegree` specification.
- Correction of printing errors arising when result of score-type test is `NA`.
- `maxRatio` checked for `NA` or `NaN` in phase2.r.
- `Siena_algorithms4.tex` renamed `Siena_algorithms.tex`; this document now is made available as a pdf file at the SIENA website.
- Some improvement of error messages for sienaTimeTest.
- p-value for goodness of fit (sienaGOF) rounded to 3 decimal places.
- File effects.pdf dropped from `\inst\doc` (it can be created by effectsDocumentation).

Changes in RSienaTest:

- sienaBayes: new parameters `nImproveMH` and `priorRatesFromData`; these give the possibility to truncate initial rate parameters depending on prior.
- glueBayes corrected so that it can be applied sequentially.
- multipleBayesTest now allows matrix parameter to test linear combinations.
- Improved plot.multipleBayesTest (to show truncation at 0).

- 2014-07-08 Revision 278.

  Changes in RSiena and RSienaTest:

  - Added `s50s` to data set.
  - Corrected `se` component of sienaFit objects (should be standard error, was its square).
  - new effects `totDist2, altInDist2, totInDist2, totDist2W, altInDist2W, totInDist2W`.
  - Some warnings for calculations of `z$regrCor` and `z$regrCoef` avoided in siena07.
  - print01Report errors corrected, and slightly improved, for descriptives for changing dyadic covariates and for upOnly / downOnly cases.

  Changes in RSienaTest:

  - sienaBayes: Internally multiplied the data-dependent choice of `priorSigma` for rate parameters by `priorKappa`; changed `z$nwarm` to 0 if `prevBayes` is used; dropped `plotit` functionality.

  - sienaBayes, glueBayes, and print.sienaBayes: adapted to allow inclusion of interaction effects without the corresponding main effects.
  - Added parameter `nwarm2` to glueBayes. Checks of identical prior parameters in this function restricted to non-rate parameters.

- 2014-06-22 R-Forge Revision 277.

  Changes in RSiena and RSienaTest:

  - Higher write-to-screen frequency for batch operation of siena07.
  - Function includeEffects now includes parameters `fix` and `test`.
  - Various small bug corrections (see the ChangeLog, 1.1-275).

  Changes in RSienaTest:

  - Changes in sienaBayes and its print and summary methods.
  - Corrected starting printing sienaBayesFit at `nfirst`.
  - New function glueBayes for combining sienaBayesFit objects.
  - Added functions simpleBayesTest and multipleBayesTest.

- 2014-04-26 R-Forge Revision 274.

  Changes in RSiena and RSienaTest:

  - Correction of effect `homWXClosure`.
  - Small change in print01Report to improve reporting two files of composition change.
  - sienaRI: require that `file` argument is not NULL for non-Windows operating systems.

- 2014-04-13 R-Forge Revisions 267-271.

  Changes in RSienaTest:

  - Updates to let sienaBayes accept a wider range of data and models (e.g., user-defined interactions); and various corrections to sienaBayes.

  Changes in RSiena and RSienaTest:

  - Added function sienaRI for assessment of relative importance of effects, with print and plot methods.
  - In coDyadCovar and varDyadCovar, centering now also is optional by the new option `centered` (like it was done for coCovar and varCovar in revision 1.1-251).
  - Corrected bug when printing siena object with a symmetric network; and in varDyadCovar corrected a bug occurring when calling it with a named list.
  - Added standard errors as component `se` to `sienaFit` objects.

- Internal changes in the code version 1.1-255 to 1.1-267 (see ChangeLog).
- No noticeable changes from version 1.1-251 to 1.1-254.

- 2014-02-13 R-Forge Revision 251

  It should be noted that two changes were made that potentially have an influence on some results obtained.

  First, the effects `gwespFF, gwespBB, gwespFB, gwespBF, gwespRR` were modified in RSienaTest to bring them in accordance with the literature. This means that the 'old' parameter α' was effectively replaced by $\alpha = -\log\left(1 - \exp(\alpha')\right)$; here α is the internal effect parameter divided by 100. For the default $\alpha = \log(2)$ this means no difference.

  Second, in the help page for sienaGOF-auxiliary, geodesic distances were changed to non-directed. This makes more sense usually and was done to avoid runtime errors that occurred very rarely.

  Changes in RSiena and RSienaTest:

  - New effects `cl.XWX`, `homXTransTrip`, `homWXClosure`, and `sharedPop`.
  - Effect `cycle4` extended to non-directed one-mode networks (for directed one-mode networks this is `sharedPop`).
  - Effect `jumpXTransTrip` ported to non-directed networks.
  - `gwesp..` effects modified and ported to non-directed networks.
  - Also take account of behavior user-specified interactions in includeInteraction and setEffect.
  - Correction: Effect `to` is not a dyadic effect.
  - Manual: added paragraph about how to import results from xtable and siena.table into MS-Word.
  - sienaGOF: added the name of the sienaFit object as attribute sienaFitName to each of the sienaGofTest objects.
  - Correction in sparseNetworkExtraction to avoid errors occurring when the extracted network has no edges.
  - In the help page for sienaGOF-auxiliary, geodesic distances changed to non-directed; which avoids a further error when the extracted network has no edges.
  - Correction of an error in print.siena for data sets including other types than `oneMode`.
  - Changed bandwidth selector for violin plots in plot.sienaGOF to 'nrd", to avoid long violins in cases where all simulations have the same outcome.

  Changes in RSienaTest:

  - Further work on SienaBayes.

  Changes in RSiena:

  - Ported effects `outRateLog` and `outTrunc2` from RSienaTest.

- 2013-12-04 R-Forge Revision 250

  Changes in RSiena and RSienaTest:

  - New option `centered` in coCovar and varCovar.
  - setEffect, updateTheta, and prevAns in siena07 now also work properly for user-specified interactions.
  - Tests Wald.RSiena and Multipar.RSiena added.
  - Error occurrence with message about `cvalue` in EvaluateTestStatistic corrected.

  - Divergent parameters in siena07 get NA for their rows and columns in the resulting covariance matrix.

  The following changes in revision 244 were ported from RSienaTest to RSiena:

  - In siena08, also report Bonferroni combination of the two Fisher combinations.
  - In siena07, rolled back change in truncation from version 1.1-227 to the earlier procedure.
  - descriptives.sienaGOF added.
  - Minor changes of output in siena.table, print.siena, siena07, and in error message for includeEffects.
  - Change artificial results from 999 to NA in siena07.
  - For ML estimation: added autocorrelations during phase 3 to print.summary.sienaFit.

- 2013-10-31 R-Forge Revision 246

  Changes in RSiena and RSienaTest:

  - New behavior objective function effects avSimAltX, totSimAltX and avAltAltX to differentiate sources of peer influence in directed networks.
  - Added effect class covarBehaviorNetObjective to effectsDocumentation.R.
  - Fix of a bug that occurred in the case of on average decreasing behavior variables.

- 2013-16-09 R-Forge Revision 245

  Changes in RSienaTest:

  - New structural rate effect outRateLog.
  - Duplication of outTrunc effect: outTrunc2, allowing use with two different parameters.
  - In siena08, also report Bonferroni combination of the two Fisher combinations.
  - In phase2 of siena07, rolled back change in truncation from version 1.1-227 to the earlier procedure.
  - Added function descriptives.sienaGOF with numerical results of plot.sienaGOF.
  - Minor changes of output in siena.table and various reports, and in error message for includeEffects.
  - Change artificial ('missing') results of siena07 from 999 to NA.
  - Added autocorrelations during phase 3 to print.summary.sienaFit for ML estimation.
  - Start of the manual reorganized and partially rewritten (with help from Zsófia Boda and András Vörös); instructions for siena01Gui separated in siena01gui.pdf.

- 2013-10-09 R-Forge Revision 244

  Changes in RSienaTest:

  - Repair bug that prevented compilation for Mac.

- 2013-09-17 R-forge revision 243

  Changes in RSiena and RSienaTest:

  - Correct bug in EffectFactory for `isolatePop` effect.

  - Improved plotting of sienaGOF objects so that observed values outside of the range of simulated values don't run off the chart.
  - Improve treatment of structural values in sienaGOF.

  Changes in RSiena:

  - Add functions AntiIsolateEffect.h and AntiIsolateEffect.cpp which were forgotten to include in revision 242.

- 2013-08-27 R-forge revision 242

  Changes in RSiena as well as RSienaTest:

  - Correction to Dolby option for the case of more than 2 waves: in phase1.r and phase3.r, scores are added (instead of averaged) over waves. (Averaging was wrong, because in phase 2 they are added.)
  - New effects: `anti isolates`, `anti in-isolates`, and `anti in-near-isolates`.
  - Effect `inIsolatePop` dropped (it was shortlived).
  - Improved printing of results of siena07 in the case simOnly.
  - Prettier response printed to console for includeEffects and setEffect.
  - z$estMeans added to sienaFit objects z: vector of estimated expected values of statistics; this is colMeans(z$sf) + z$targets but if dolby, the regression on the scores is subtracted.

- 2013-08-23 R-forge revision 241

  Changes in RSiena as well as RSienaTest:

  - Corrected bug (leading to error message) that occurred if there was only one option for choice of alter. It appeared mainly in cases where all changes were upward only (or downward only); in practice, it only was observed yet for two-mode networks with upward changes only.
  - Drop the unintended multiplication of the target statistic for the `inPop` effect by n.
  - Trapped some execution errors (mainly associated with inversion of singular matrices) and allowed the functions to end properly with a warning message.
  - Added various degree-related effects to bipartite networks.
  - New effect `inIsolatePop`.

- 2013-08-08 R-forge revision 240

  Changes in RSienaTest:

  - Added the parameter reduceg to siena07.

  Changes in RSiena and RSienaTest:

  - Added effects crprod and inPopIntn for two-mode networks.

- 2013-06-18 R-forge revision 232

  Changes in RSiena:

  - The possibility to use the obsolete packages snow and rlecuyer for R versions older than 2.14.0 was dropped; their functionality was replaced by package parallel.
  - The DESCRIPTION file was corrected to satisfy CRAN requirements.

- 2013-06-15 R-forge revision 231

  Changes in RSiena as well as RSienaTest:

  - Make the 'cumulative' option operational in BehaviorDistribution for sienaGOF.
  - Correct bug in treatment of missing values in sparseMatrixExtraction for sienaGOF.
  - Allow sparse observed data matrices, and structural zeros and ones, in sparseMatrixExtraction and networkExtraction, and bipartite networks in networkExtraction for sienaGOF.
  - Report correct centering (by overall means) of individual covariates for multi-group objects in print01Report.
  - If there is a composition change object, MoM estimation is forced to be non-conditional. This is reported in the help file for sienaCompositionChange.

- 2013-05-10 R-forge revision 230

  Changes in RSiena as well as RSienaTest:

  - Check whether the maximum observed degree is not higher than maxDegree.
  - Fix error in implementation of maxDegree.
  - Fix bug in print.siena and extend print.siena.
  - Make print method for class sienaDependent.

- 2013-04-19 R-forge revision 227

  Changes in RSiena as well as RSienaTest; both now are very similar; sienaBayes, algorithms, and profileLikelihoods are the only functions in RSienaTest not in RSiena. Available effects now are the same in both packages.

  Main changes visible to users:

  For Siena only:

  - function bayes was removed (still under development in RSienaTest).
  - Attributes allowOnly and simOnly ported from RSienaTest.
  - Improved error messages in includeEffects ported from RSienaTest.
  - sienaGOF ported from RSienaTest.
  - siena.table ported from RSienaTest.

  For RSienaTest only:

  - bayes renamed to sienaBayes and considerably changed, with print option.

  ForRSienaand RSienaTest:

  - Changes to sienaGOF: new use structure with extraction functions sparseMatrixExtraction, networkExtraction, behaviorExtraction, allowing the testing of any dependent variable; commented out some superfluous lines.
  - The function sienaModelCreate is now called sienaAlgorithmCreate, but the earlier name is still retained as an alias; the class name of the object created by this function is now called sienaAlgorithm.
  - The function sienaNet is now called sienaDependent, but the earlier name is still retained as an alias; the class name of the object created by this function is now sienaDependent.

– The function effectsDocumentation now has an extra argument effects; if this points to an effects object, all available effects in this effects object are listed with shortName, with a variety of other often used characteristics.
– Added effects (some existed already in RSienaTest):
average exposure effect on rate xxxxxx, avExposure
susceptibility to av. exp. by indegree effect on rate xxxxxx, susceptAvIn
total exposure effect on rate xxxxxx, totExposure,
infection by indegree effect on rate xxxxxx, infectIn,
infection by outdegree effect on rate xxxxxx, infectOut,
susceptibility to av. exp. by zzzzzz effect on rate xxxxxx, susceptAvCovar
infection by zzzzzz effect on rate xxxxxx, infectCovar,
WW=>X cyclic closure of xxxxxx, cyWWX
WW=>X shared incoming xxxxxx, InWWX
WW=>X shared outgoing xxxxxx, OutWWX
xxxxxx alter at distance 2 (#), altDist2
xxxxxx similarity at distance 2, simDist2
transitive triplets xxxxxx similarity, simXTransTrip
transitive triplets same xxxxxx, sameXTransTrip
transitive triplets jumping xxxxxx, jumpXTransTrip
transitive reciprocated triplets, transRecTrip
GWESP I $->$ K $->$ J (#), gwespFF
GWESP I $<-$ K $<-$ J (#), gwespBB
GWESP I $<-$ K $->$ J (#), gwespFB
GWESP I $->$ K $<-$ J (#), gwespBF
GWESP I <> K <> J (#), gwespRR
isolate - popularity, isolatePop
in-isolate Outdegree, inIsDegree
network-isolate, isolateNet
outdegree$\wedge$(1/#) xxxxxx popularity, outPopIntn
closure jumping yyyyyy, jumpWWClosure
mixed xxxxxx closure jumping yyyyyy, jumpWXClosure
cyclic closure of xxxxxx, cyClosure
shared incoming xxxxxx, sharedIn
– Outdegree-popularity effect: multiplication by n dropped.
– GWESP effects: default parameter changed from 25 to 69 (corresponding to $\alpha = \log(2)$.) See earlier in this manual.
– Added to siena07 (defined by sienaAlgorithmCreate): option Dolby for variance reduction. Correlations between scores and statistics are reported in output file; this is a measure for the amount of variance reduction.
– Added to siena07: option diagonalize for having more possibilities for tuning the algorithm (extent of diagonalization of matrix D in Robbins-Monro update).
– sienaTimeTest updated; now also contains effect-wise tests, groupwise tests (for group objects), automatic exclusion of collinear effects, and has prettier output and improved summary.

  - Overall maximum convergence ratio, x$tconv.max (maximum value of t-ratio for convergence, for any linear combination) added to result of siena07.
    This is a very severe convergence criterion, and not meant as a default criterion to judge convergence in practical cases.
  - The printmethod for objects of class siena (created by sienaDataCreate) has been extended with printing uponly and downly attributes, if these are TRUE.
  - A bug in the starting values for two-mode networks was corrected.
  - Small bug fixed in print01Report for reporting of uponly and downonly, in the case where this does not affect all periods.
  - Changed almost all .Rd documentation files: sometimes to make them better understandable or complete, sometimes to make more appropriate examples, sometimes only minor prettifications.
  - Updated scripts:
    Rscript01DataFormat.R, Rscript02VariableFormat.R, Rscript03SienaRunModel.R, Rscript04SienaBehaviour.R. (of RSienaDescriptives only the date was changed.)

- 2012-12-24 R-forge revision 222
  - Changed example on sienaNet help page to stop it masking data files in the package.
  - Example on sienaGOF page now runs. (RSienaTest only)
  - profileLikelihood (RSienaTest only) returns its object invisibly (i.e. it does not print when not assigned but can be assigned).

- 2012-12-23 R-forge revision 221,RSienaand RSienaTest:
  changed version check to cope with R version 3.0.0.

- 2012-07-05 R-forge revision 219, for RSienaTest only:
  - Further changes to bayes.
  - Additional effects connected with triadic closure interacting with covariates.
  - Some networks from Chris Baerveldt's data set added as data objects (N34* and HN34*).

- 2012-06-11 R-forge revision 217, for RSienaTest only:
  - Preliminary updates to sienaGOF to put more power in the hands of the user. A user may now extract more than one dependent variable from the dataset.

- 2012-06-07 R-forge revision 216, for RSienaTest only:
  - New effects connected with isolates; and with mixed WWX triadic closure (in various patterns) for dyadic covariates as well as multiple dependent networks.
  - Modifications to bayes (seems to run OK now, except that multiple groups option does not work for dyadic covariates; still not documented for general use).

- 2012-05-18 R-forge revision 213, for RSienaTest only:
  - Allow observed networks to have density 0 or 1 (not that it is generally advisable to use such data sets).
  - Incorporated argument simOnly in sienaModelCreate to facilitate simulation without estimation.

  - Incorporated argument allowOnly in sienaNet to permit ignoring monotonicity in data and its consequences for upOnly and downOnly.
  - Some new effects: interactions between reciprocity and transitivity.

- 2012-03-29 R-forge revision 211
  Fixed bug in effectsDocumentation.
- 2012-03-29 R-forge revision 210
  Altered ML code in hope of fixing intermittent ML bug. Just might cause different answers.
- 2012-03-25 R-forge revision 208
  - fix bug in bipartite network endowment and creation effect scores.
  - Rationalise behavior/network effects for symmetric networks in allEffects.csv, effects.r (partially).
  - Fix bug which caused crash creating starting values for sparse matrices with movements only in one direction.
- 2012-03-16, 2012-03-21, R-forge revisions 206/207: new behavior rate effects.
- 2012-03-07 R-forge revision 205
  - Bug fix for effects AvSimEgoX, totSimEgoX, avAltEgoX with changing covariates.
  - Minor alterations to altDist2, simDist2 and the multi network versions.
  - Bug fix in probability in chain for symmetric networks type b models.
- 2012-02-29 R-forge revision 204
  - Fixed bugs in endowment and creation effect statistics for behavior Similarity effects.
  - Fixed bug causing occasional failure in bayes routine
  - Fixed bug causing occasional failure in maximum likelihood with constraints.
  - Added error message if try to use maximum likelihood with composition change.
  - Fixed bug in endowment and creation effect score calculation for symmetric network pairwise models.
  - File cluster.out is now removed before recreation.
  - Meta analysis summary now does not contain a list of NULLs at the end.
  - Minor changes to print and messages formats.
- 2012-02-19 R-forge revision 203
  - Fixed minor bug in ML initialisation: will alter results slightly.
  - New check for updated version when using multiple processes.
  - Amended code in manual for making sparse matrices.
- 2012-02-07 R-forge revision 200
  - Bug fix to scores for behavior variable rate effects.
  - siena07 now stops if cannot get a derivative matrix in phase 1.
- 2012-01-29 R-forge revision 198 Fix bug in initializing rate parameters for bipartite and behavior variables in maximum likelihood. Will change results slightly.

- 2012-01-29 R-forge revision 197
  - Fix bug in effFrom with changing covariates. The targets depended on the compiler.
  - Fix bug in creation effects in Maximum likelihood.
- 2012-01-20 R-forge revision 195
  - NaN's in covariates were causing problems: now treated as though NA in C++, as they always were in R.
  - New file 'arclistdata.dat' added to examples directory
  - New maintainers address: rsiena@stats.ox.ac.uk
  - Print method for sienaFit objects now includes values of fixed parameters, rather than NA.
- 2012-01-17 R-forge revision 194
  - fix to prtOutMat to stop crash with null matrix
  - relaxed restrictions on behavior interactions in line with the manual
  - changes to validation of bipartite networks: should now be consistent
- 2012-01-17 R-forge revision 192.
  - minor but extensive changes to manual
  - minor changes to scripts
- 2011-12-15 R-forge revision 191. Some of these may alter results slightly.
  - Altered calculations of probabilities to avoid overflows
  - Fixed bug in storage of MII in bayes
  - Removed endowment effect for IndTies for symmetric networks.
  - Removed code for storing change contributions on ministeps: not functioning
  - Set random number type to 'default' at start of siena07.
- 2011-12-14 R-forge revision 190 Fixed bug in Bayes left over from R 189.
- 2011-12-14 R-forge revision 189 Fixed minor bugs in reports and error messages.
- 2011-12-04 R-forge revision 186 Fixed some bugs in ML estimation procedure which will alter the results slightly. Added algorithm functions to RSienaTest package.
- 2011-11-27 R-forge revision 185
  - Bayes and algorithm code now uses parallel package
  - Fixed memory leaks in ML estimation. Less space needed!
  - Other minor changes to ML with missing values (still incomplete)
- 2011-11-14 R-forge revision 184: Fix memory leaks in calculation of rate statistics.
- 2011-11-11 R-forge revision 183:
  - Fix bug stopping interruption in phases 1 and 3 (since recent change)
  - Check whether dfra from maxlike or not when using prevAns
- 2011-11-11 R-forge revision 182:

  - fix bug in ML/Bayes returning acceptances.
  - fix bug in sienaTimeFix with multi groups and differing actor set sizes
- 2011-11-04 R-forge revision 181: reset random number type after using parallel package.
- 2011-10-28 R-forge revision 179: fix bug in forking processes
- 2011-10-27 R-forge revision 177/8:
  - Change to covariance matrix for effects which have been fixed
  - Added new package for parallel running to be used from R 2.14.0. New option to use forking processes on non-Windows platforms.
  - Changes from revision 175 copied to RSiena
  - Updates to maximum likelihood estimation: NB this is still under development, and should not be used with missing data.
  - Added bayes, updateTheta functions to RSiena
  - sienaTimeTest for finite differences or ML now in RSiena
  - Space saving matrices used for derivatives in RSiena now, and optional by wave in ML.
- 2011-10-14 R-forge revision 176 (RSienaTest only) bug fix in diffusion effects, altered scripts in manual a little
- 2011-10-06 R-forge revision 175
  - Fix bug with multiple symmetric networks.
  - Limit constraints to be between both symmetric or non-symmetric networks
  - Added scripts to package (RSienaTest only)
  - siena07 called with batch = FALSE no longer crashes if called on mac or linux with no X11 available. (RSienaTest only)
- 2011-09-19 R-forge revision 172: (RSienaTest only) Diffusion rate effects.
- 2011-09-07 R-forge revision 171
  - Fix bug in siena08: crashed if underlying effects for interaction were not selected. (Or possibly with time dummies!).
  - Fix bug where print from siena08 was not produced if a previous display to the screen had occurred.
  - New parameter in siena08 to control number of iterations.
  - RSienaTest only: added validation to updateTheta
- 2011-08-08 RSienaTest only R-forge revision 168/9
  - More work on maximum likelihood.
  - When using finite difference derivative estimation or maxmimum likelihood estimation, return of derivatives by wave is optional, controlled by parameter byWave to siena07.
  - Format of derivatives by wave has altered: sienaTimeTest will be incompatible with older objects which used finite differences or maximum likelihood.
  - New function updateTheta to copy theta values from a fit to an effects object.

  - Time dummies in siena07 are created before the initial values are updated from any prevAns, so values may be copied to time dummies also.
  - Amended headings in print and summary for siena fit objects.
  - New function bayes is now fully available with a help page and no need to use RSiena::: when calling it.

- 2011-08-03 R-forge revision 167
  - Fix another display of manual in siena01Gui
  - Added network names to relevant behavior effects if there is more than one network.
  - Altered names of interaction effects to remove duplicate network names
  - Renamed avSimX, totSimX, avAltX to avSimEgoX, totSimEgoX, avAltEgoX
  - Trapped error caused by omitting Actors node set when specifying others.
- 2011-07-27 R-forge revision 164:
  - Include quadratic shape effect by default unless range is less than 2.
  - Shorten behavior interaction effectnames by removing the repeated variable name.
  - Fix bug when displaying manual from siena01Gui.
- 2011-07-23 R-forge revision 163: Fix bug in effectsDocumentation, reduce memory size needed for non-ML, non-finite-difference models.
- 2011-07-02 R-forge revision 161:
  - Fix problem with bayes routine with single data object only.
  - Fix problems with getRSienaRDocumentation: internal functions within internal functions now work (but still not automatically) and function now runs on non-Windows too.
- 2011-06-24. R-forge revision 160: behavior endowment effects are now all defined consistently as current value less previous one, as in the manual.
- 2011-06-22, 2011-06-23. R-forge revision 158/159: behavior interactions. Minor bug fixes to correct effects object and inclusion of non-requested underlying effects. Replace influence interaction effects by the three options. Time dummies for behavior effects.
- 2011-06-18 R-forge revision 157: Fixed minor bug in siena07: code controlling maximum size of move was incorrect if using prevAns for an exactly equivalent fit.
- 2011-06-13 R-forge revision 156: fixed bug removing density effect for bipartite networks with some only waves up or down only.
- 2011-06-12 R-forge revision 155:
  - Fixed bug with behavior variables with values 10 or 11: the 10th value in the matrix had 10 subtracted from it.
  - Fix for short name for egoXaltX effect for non-directed networks. Now matches the name for directed networks
- 2011-06-04 R-forge revision 153:
  - Maximum likelihood: this is still under development. Correction for bipartite networks and networks with constraints. Variable length of permutations will change results (slightly) compared with previous versions.

  - Creation effects: not yet complete.
  - Requested time dummies should now appear on the effects object print.
  - Bayesian routine (still very much under development) has altered.
- 2011-05-27 R-forge revision 150: Removed effect for an absolutely constant covariate with two networks.
- 2011-05-26 R-forge revision 148: Improvements to sienaGOF, added script to manual.
- 2011-05-16 R-forge revision 146:
  - Documentation improvements
  - Can read (some) non-directed network from Pajek files
- 2011-04-19 R-forge revision 144:
  - New effects: out trunc effect
  - Enhancements to siena08
- 2011-03-13 R-forge revision 142/3: GWESP effects (RSienaTest only)
- 2011-02-24 R-forge revision 140: adds functionality to sienaGOF for plotting image matrices of the simulations, cumulative tests based on the Kolmogorov- Smirnov test statistic, and conforms to coding standards.
- 2011-02-24 R-forge revision 139: fixes for bipartite networks with ML. ML is still incomplete, and will not work correctly with missing data or endowment effects.
- 2011-02-22 R-forge revision 137: (RSienaTest only) Additional work on the goodness of fit functionality (see ?sienaGOF)
- 2011-02-21 R-forge revision 136:
  - Fixed bug in bipartite network processing. Diagonal (up to number of senders) was being zeroed.
  - siena01Gui: corrected test for maximum degree in display.
- 2011-02-05 R-forge revision 134:
  - Enhanced features (and minor bug fixes) in siena08 report. (in revision 133 in RSienaTest)
  - Bug fix in iwlsm
  - sienaDataCreateFromSession, sienaTimeTest: improved error messages.
  - ML support for bipartite networks (still work in progress, particularly for missing data)
- 2011-01-17 R-forge revision 131/2: (RSienaTest only) New goodness of fit functions: work in progress.
- 2011-01-16 R-forge revision 130:
  - Fix bug for bipartite networks which usually crashed, but could have given incorrect answers.
  - Fix bug with multiple processes and sienaTimeTest.
  - Default value ofsiena07 argument initC is now `TRUE`.
- 2011-01-08 R-forge revision 129:

  - fix to sienaTimeFix for time dummies on covariate effects etc.
  - Suppressed warning message when loading snow package.

- 2010-12-02 R-forge revision 128:
  - Corrections to scores for symmetric pairwise models
  - ML now runs with missing data. Not yet sure it is correct!
  - New multiple network effects: To, altDist2W, simDist2W.
  - Can now use setEffects to update basic rate initial values
  - multiplication factor for ML now a parameter in sienaModelCreate.
  - Fixed bug meaning that covariate multiple network effects did not appear if the covariates was a behavior variable.
  - Can now run sienaTimeTest on fits from finite differences and maximum likelihood.
  - User defined interactions (and time dummies) can be expanded when printing the effects object, use parameter `expandDummies = TRUE`.
- 2010-11-25 R-forge revision 126:
  - Changed version of RSiena to be 1.0.12 and copied all new features which were only in RSienaTest to RSiena. RSiena and RSienaTest are functionally the same at this time.
  - New version of sienaTimeTest and sienaTimeFix.
  - Bayesian routine (experimental) can be used with multiple dependent variables.
- 2010-11-05 R-forge revision 125:
  - Corrected bug in report from siena07 about detailing network types.
  - Networks appear in data object before behavior variables regardless of order of submission to sienaDataCreate
  - RSienaTest only: new effect: in structural equivalence
  - RSienaTest only: new models for symmetric networks
  - bug fixed to sienaTimeTest: non included underlying effects for user defined interactions, multiple dependent networks and multiple groups.
- 2010-10-22 R-forge revision 124:
  - Fixed bug in sienaTimeTest when only one effect
  - Removed standalone siena01Gui. Still available within R.
- 2010-10-09 R-forge revision 122:
  - Distance two effects: added parameter
  - Bug in calculation of starting values for behavior variables. (RSiena only)
- 2010-09-20 R-forge revision 120: Bug fixes:
  - Multiple groups with 2 dyadic covariates had incorrect names
  - Multiple processes failed (RSiena only)
  - Minor print format corrections

  - Bug in calculation of starting values for behavior variables. (RSienaTest only)

- 2010-08-20 R-forge revision 117: RSienaTest only. Documentation updates, algorithms may work again!
- 2010-08-20 R-forge revision 116: forgotten part of change for print of sienaFit (RSiena only)
- 2010-08-20 R-forge revision 115: fixed bug in siena08 p-values on report, and minor corrections to layout of print of sienaFit.
- 2010-07-19 R-forge revision 114: fix a bug in initial report: names of multiple behavior variables were incorrect.
- 2010-07-10 R-forge revision 113: fix bugs
  1. endowment effect unless using finite differences failed
  2. could not return bipartite simulations
- 2010-07-04 R-forge revision 112: fix bug in groups with constant dyadic covariates and only 2 waves. (Introduced in revision 109).
- 2010-07-03 R-forge revision 111: bipartite networks now have a no-change option at each ministep of simulation.
- 2010-06-25 R-forge revision 110: updated manual pages for dyadic covariates.
- 2010-06-25 R-forge revision 109:
  - Dyadic covariates may have missing values and sparse input format.
  - Removed some inappropriate dyadic covariate effects for bipartite networks.
  - Score test output now available via summary on a fit.
  - Corrected conditional estimation for symmetric networks.
  - Now do not need to specify the variable to condition on if it is the first in sienaModelCreate
- 2010-06-21 R-forge revision 108:
  - effects print method with no lines selected no longer gives error, new argument includeOnly so you can print lines which are not currently included.
  - effectsDocumentation was failing due to timeDummy column
  - New average alter effects
  - Corrected format of error message if unlikely to finish epoch/
  - Corrected print report for multiple groups via the GUI, and for 8 waves.
  - Fixed names for used defined dyadic interactions.
  - Fixed bug where sienaTimeTest dummies with RateX would not work with changing covariates.
- 2010-06-21 R-forge revision 107: RSienaTest only: reinstated includeTimeDummy.
- 2010-06-18 R-forge revision 106: new version numbers: 1.0.11.105 and 1.0.12.105 forRSienaand RSienaTest respectively.
- 2010-06-18 R-forge revision 105: Fixed siena01Gui bug when trying to edit the effects. Problem was introduced in revision 81.

- 2010-06-10 Updated time heterogeneity script for Tom
- 2010-06-08 R-forge revision 102: RSienaTest only. Removed includeTimeDummy.
- 2010-06-08 R-forge revision 101: RSienaTest only. Fixed RateX so that it works with changing actor covariates as well.
- 2010-06-08 R-forge revision 100: corrected revision numbers in ChangeLog.
- 2010-06-08 R-forge revision 99 Fix to bug introduced in revision 98: bipartite networks could not have 'loops'
- 2010-06-08 R-forge revision 98
  - Fix to bug in constant dyadic covariates with missing values.
  - Changes to treament of bipartite networks. The processing of these is still under development: we need to add the possibility of 'no change' to the ministeps. Code to deal with composition change has been added, and the treatment of missing values in sparse matrix format networks has been corrected further (the change in revision 96 was not quite correct).
- 2010-06-04 R-forge revision 97RSiena includeTimeDummy not exported so not available to the user.
- 2010-06-04 R-forge revision 96 RSiena
  - bug fixes as in revisions 92, 93.
  - Changes and bug fixes to sienaTimeTest etc. as in revisions 85–89,
  - includeInteractions now will unInclude too.
- 2010-06-04 R-forge revision 93 (RSienaTest only)
  - New algorithms function (not in package: in the examples directory).
  - Progress on maximum likelihood code.
  - Bug fixes: print empty effects object, misaligned print.sienaFits, crash in print.sienaEffects with included interactions.
  - silent parameter now supresses more.
  - Added time dummy field to setEffects and removed from includeEffects.
  - includeInteractions now will unInclude too.
  - includeTimeDummy now sets or unsets the include flag, and prints the changed lines.
  - Using composition change with bipartite networks will give an error message – until this is corrected.
  - Separate help files for sienaTimeTest, plot.sienaTimeTest, includeTimeDummy.
  - Bug fix to treatment of missing data in sparse format bipartite networks.
  - Change to error message if an epoch is unlikely to terminate.
- 2010-06-04 R-forge revision 92 (RSienaTest only) New average alter effects. Bug fix to effects object for more than two groups.
- 2010-05-29 R-forge revision 89 (RSienaTest only) New option to control orthogonalization in sienaTimeTest, changes to includeEffects and sienaDataCreate (NB changes reverted in revision 93).
- 2010-05-28 R-forge revision 88 (RSienaTest only) Time dummies for RateX effects

- 2010-05-27 R-forge revision 87 (RSienaTest only) bug fix to plot.sienaTimeTest
- 2010-05-23 R-forge revision 86 (RSienaTest only) Bug fix to plot.sienaTimeTest, new function includeTimeDummy
- 2010-05-22 R-forge revision 85 (RsienaTest only) fixed bug in siena**TimeTest** with unconditional simulation.
- 2010-04-24 R-forge revision 81 New print, summary and edit methods for Siena effects objects
- 2010-04-24 R-forge revision 80
  - fixed bug causing crash with rate effects and bipartite networks.
  - added trap to stop conditional estimation hanging
  - new functions (INCOMPLETE) for maximum likelihood and Bayesian estimation (one period (two waves) only, no missing data, one dependent variable only for Bayesian model).
- 2010-04-13 R-forge revision 79 new function: sienaTimeTest.
- 2010-04-12 R-forge revision 78 fix minor bugs in reports, allow character input to effect utility functions, include effect1-3 etc on display of included effects in siena01Gui.
- 2010-04-12 R-forge revision 77 (RSiena only) As for RSienaTest revision 76
  - Report of 0 missings corrected
  - display of effect1-effect3 in siena01Gui
  - allow entry of character strings or not in includeEffects etc.
- 2010-04-12 R-forge revision 76 (RSienaTest only) Various bug fixes
  - Memory problems when calculating derivatives with many iterations and parameters.
  - Occasional effects not being included correctly due to trailing blanks
  - Some minor details of reports corrected.
- 2010-03-31 R-forge revision 75 fixed bug with dyadic covariates and bipartite networks.
- 2010-03-27 R-forge revision 71 (RSienaTest only)
  - Fixes as for RSiena in revision 68/69/70 for RSiena
  - New version number 1.0.12
- 2010-03-27 R-forge revision 70 (RSiena only)
  - Fix to crash at end of phase 3 with multiple processes and conditional estimation
  - Correct carry forward/backward/use mode for behavior variables
  - Fix bug causing crash in Finite Differences with only one effect
- 2010-03-24 R-forge revision 69 (RSiena only)
  - New features and bug fixes as for revision 63 in RSienaTest.
  - 4-cycles effect has new shortName: cycle4.
  - some percentages on reports were proportions not percentages
  - Sped up treatment of missing values in sparse format networks.

  - Fix: now allows more than one value to indicate missing in covariates.

- 2010-03-12 R-forge revision 68 new version number for RSiena.
  In siena01Gui, allow waves for SienaNet inputs to be numbered arbitrarily, rather than insisting on 1-n. Change simply allows this, the actual wave numbers are not yet used on reports etc.
- 2010-03-17 R-forge revision 66 Corrected processing of user-specified interaction effects with multiple processes. This had originally worked but failed when one no longer had to include the underlying effects.
- 2010-03-16 R-forge revision 64 covarBipartite ego effect had been given type dyadic rather than ego.
- 2010-03-16 R-forge revision 63 (RSienaTest only)
  - new functions siena08 and iwlsm, for meta analysis
  - can now use different processes for each wave. Not recommended: usually slower than by iteration, but will be useful with ML routines when they are completed.
  - No longer crashes with missing dyadic covariates.
- 2010-02-27 R-forge revision 61 (RSiena only) bug fix: random numbers used with multiple processes were the same in each run. Now seed is generated from the usual R random number seed. Also fixed a display bug if running phase 3 with few iterations.
- 2010-02-16 R-forge revision 60 (RSienaTest only) added average indegrees to reports. Also constraints.
- 2010-02-12 R-forge revision 59 (RSienaTest only) Fix to bugs in printing version numbers and in using multiple processes (would revert to RSiena package.) Added a skeleton MCMC routine.
- 2010-02-11 R-forge revision 57 Fix to bug in siena01Gui where in conditional estimation, the estimated values were not remembered for the next run.
- 2010-02-11 R-forge revision 56 (RSiena only) Multiple network effects, constraints between networks.
- 2010-02-11 R-forge revision 55 (RSienaTest only) New silent option forsiena07.
- 2010-02-11 R-forge revision 54 (RSienaTest only) Fix to covariate behavior effect bug.
- 2010-02-11 R-forge revision 53 Fixed bug in siena01 GUI which ignored changes to all effeccts
- 2010-02-07 R-forge revision 52 (RSiena only) New silent option forsiena07.
- 2010-02-04 R-forge revision 51 (RSiena only)
  - Fix to covariate behavior effect bug.
  - Fix to default effects with multiple networks.
- 2010-02-01 R-forge revision 49 (RSienaTest) only Fixes to bugs in constraints.
- 2010-01-28 R-forge revision 48 Fix to bug in sorting effects for multiple dependent variables.
- 2010-01-26 R-forge revision 47 (RSienaTest only)
  - New version: 1.0.10
  - Multiple networks
  - Constraints of higher, disjoint, atLeastOne between pairs of networks.

- 2010-01-19 R-forge revision 45 (RSiena), 46 (RSienaTest)
  New documentation for the effects object.
- 2010-01-18 R-forge revision 43 (RSiena)
  - new behavior effects
  - user specified interactions
  - new utilities to update the effects object
- 2010-01-15 R-forge revision 41 (RSienaTest only)
  - new effect: Popularity Alter, and altered effect1-3 to integers to correct bug in fix(myeff)
  - new utility functions to update effects object
  - no longer necessary to include underlying effects for interactions.
  - user parameter for number of unspecified behavior interactions
  - remove extra sqrt roots in standard error of rates for conditional estimation (see revision 31)
- 2010-01-15 R-forge revision 40: (RSiena only)
  remove extra sqrt roots in standard error of rates for conditional estimation (see revision 32)
- 2010-01-02 R-forge revision 34
  Corrected layout of printand xtable for SienaFit objects with both behavior and network variables.
- 2010-01-01 R-forge revision 33
  Updated change log and manual in RSiena and ChangeLog in RSienaTest.
- 2010-01-01 R-forge revision 32
  - print07report.r: corrected standard errors for rate estimate for conditional estimation: needed square roots. (RSiena)
- 2009-12-31 R-forge revision 31
  - print07report.r: corrected standard errors for rate estimate for conditional estimation: needed square roots. RSienaTest only
  - more behavior effects in RSienaTest.
- 2009-12-17 R-forge revision 30
  Fixed bug in dyadic interactions in RSienaTest
- 2009-12-17 R-forge revision 29
  Fixed bug in 3-way interactions in RSienaTest
- 2009-12-14 R-forge revision 28
  Fixed bug in use of multiple processes for RSiena.
- 2009-12-14 R-forge revision 27
  Fixed bug in use of multiple processes for RSienaTest.
- 2009-12-01 R-forge revision 26
  Created RSienaTest which includes user specified interactions.
- 2009-11-20 R-forge revision 25

  - version number 1.0.8
  - The default method for estimation is conditional if there is only one dependent variable.
  - Movement of behavior variable restricted if all observed changes are in one direction. In this case, linear change effects removed.
  - If all observed changes in a network are in one direction, density effects are removed.
  - If a behavior variable only takes two values the quadratic effects are not selected by default.
  - t-statistics appear on print of sienaFit object.
  - easier to use xtable method
  - warning if behavior variables are not integers
  - Fixed bug in editing all effects in the GUI.
  - Fixed a bug in effect creation for changing dyadic covariates
  - Fixed a bug in returning simulated dependent variables
  - Now fails if there are only two waves but you have a changing covariate. In the GUI, can just change the type.

- 2009-11-08 R-forge revision 24
  - version Number 1.0.7
- 2009-11-08 R-forge revision 23
  - corrected bug in creation of effects data frame for multi group projects and for changing covariates
  - added effect numbers to the Estimation screen
- 2009-11-08 R-forge revision 22
  - new option to edit effects for one dependent variable at a time. Model options screen layout altered slightly.
- 2009-11-08 R-forge revision 21
  - Fixed a bug causing crashes (but not on Windows!) due to bad calculation of derivative matrix.
- 2009-10-31 R-forge revision 17
  - version Number 1.0.6
  - xtable method to create LaTeXtables from the estimation results object.
  - added support for bipartite networks
  - structural zeros and 1's processing checked and amended
  - use more sophisticated random number generator unless parallel testing with siena3.

B References

Albert, A. and Anderson, J. A. (1984). On the existence of the maximum likelihood estimates in logistic regression models. *Biometrika*, 71:1–10.

Amati, V., Schönenberger, F., and Snijders, T. A. B. (2015). Estimation of stochastic actor-oriented models for the evolution of networks by generalized method of moments. *Journal de la Société Française de Statistique*, 156:140–165.

Amati, V., Schönenberger, F., and Snijders, T. A. B. (2019). Contemporaneous statistics for estimation in stochastic actor-oriented co-evolution models. *Psychometrika*, 84:1068–1096.

An, W. (2015). Multilevel meta network analysis with application to studying network dynamics of network interventions. *Social Networks*, 43:48–56.

Bather, J. (1989). Stochastic approximation: A generalisation of the Robbins-Monro procedure. In Mandl, P. and Hušková, M., editors, *Proceedings of the fourth Prague symposium on asymptotic statistics*, pages 13–27. Charles University, Prague.

Block, P. (2015). Reciprocity, transitivity, and the mysterious three-cycle. *Social Networks*, 40:163–173.

Butts, C. T. (2008). Social network analysis with `sna`. *Journal of Statistical Software*, 24(6).

Cheadle, J. E., Stevens, M., Williams, D. T., and Goosby, B. J. (2013). The differential contributions of teen drinking homophily to new and existing friendships: An empirical assessment of assortative and proximity selection mechanisms. *Social Science Research*, 42:1297–1310.

Cochran, W. G. (1954). The combination of estimates from different experiments. *Biometrics*, 10:101–129.

Davison, A. C. and Hinkley, D. V. (1997). *Bootstrap Methods and Their Application*. Cambridge University Press, Oxford.

de la Haye, K., Embree, J., Punkay, M., Espelage, D. L., Tucker, J. S., and Green, H. D. (2017). Analytic strategies for longitudinal networks with missing data. *Social Networks*, 50:17–25.

Donovan, T. M. and Mickey, R. M. (2019). *Bayesian statistics for beginners: A step-by-step approach*. Oxford University Press, Oxford.

Efron, B. (1987). Better bootstrap confidence intervals. *Journal of the American Statistical Association*, 82(397):171–185.

Fisher, R. A. (1932). *Statistical Methods for Research Workers*. Oliver & Boyd, 4th edition.

Gasparrini, A., Armstrong, B., and Kenward, M. G. (2012). Multivariate meta-analysis for non-linear and other multi-parameter associations. *Statistics in Medicine*, 31:3821–3839.

Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., and Rubin, D. B. (2014). *Bayesian Data Analysis*. Chapman & Hall / CRC, Boca Raton, FL, 3d edition.

Geyer, C. J. and Thompson, E. A. (1992). Constrained Monte Carlo maximum likelihood for dependent data. *Journal of the Royal Statistical Society, Series B*, 54:657–699.

Gill, J. (2015). *Bayesian Methods: A Social and Behavioral Sciences Approach*. Chapman & Hall / CRC, Boca Raton, FL, 3d edition.

Goldthorpe, J. H. (2001). Causation, statistics, and sociology. *European Sociological Review*, 17:1–20.

Greenan, C. C. (2015a). Diffusion of innovations in dynamic networks. *Journal of the Royal*

Statistical Society, Series A, 178:147–166.

Greenan, C. C. (2015b). *Evolving Social Network Analysis: developments in statistical methodology for dynamic stochastic actor-oriented models*. PhD thesis, University of Oxford.

Hauck, W.W., J. and Donner, A. (1977). Wald's test as applied to hypotheses in logit analysis. *Journal of the American Statistical Association*, 72:851–853.

Hedges, L. V. and Olkin, I. (1985). *Statistical Methods for Meta-analysis*. New York: Academic Press.

Hintze, J. L. and Nelson, R. D. (1998). Violin plots: A box plot-density trace synergism. *The American Statistician*, 52:181–184.

Holland, P. W. and Leinhardt, S. (1970). A method for detecting structure in sociometric data. *American Journal of Sociology*, 70:492 – 513.

Holland, P. W. and Leinhardt, S. (1973). The structural implications of measurement error in sociometry. *Journal of Mathematical Sociology*, 3:85–111.

Holland, P. W. and Leinhardt, S. (1976). Local structure in social networks. *Sociological Methodology*, 6:1–45.

Hollway, J. (2026). *autograph: Automatic Plotting of Many Graphs*.

Huisman, M. E. and Snijders, T. A. B. (2003). Statistical analysis of longitudinal network data with changing composition. *Sociological Methods & Research*, 32:253–287.

Huisman, M. E. and Steglich, C. (2008). Treatment of non-response in longitudinal network data. *Social Networks*, 30:297–308.

Hunter, D. R. (2007). Curved exponential family models for social networks. *Social Networks*, 29:216–230.

Indlekofer, N. and Brandes, U. (2013). Relative importance of effects in stochastic actor-oriented models. *Network Science*, 1:278–304.

Kalish, Y. (2020). Stochastic actor-oriented models for the co-evolution of networks and behavior: An introduction and tutorial. *Organizational Research Methods*, 23:511–534.

Kaplan, D. and Depaoli, S. (2013). Bayesian statistical methods. In Little, T. D., editor, *Oxford Handbook of Quantitative Methods. Volume 1: Foundations*, chapter 20, pages 407–437. Oxford University Press, Oxford.

Koskinen, J. and Snijders, T. A. B. (2013). Longitudinal models. In Lusher, D., Koskinen, J., and Robins, G., editors, *Exponential Random Graph Models*, pages 130–140. Cambridge University Press, New York.

Koskinen, J. H. (2004). *Essays on Bayesian Inference for Social Networks*. PhD thesis, Department of Statistics, Stockholm University.

Koskinen, J. H. and Edling, C. (2012). Modelling the evolution of a bipartite network – Peer referral in interlocking directorates. *Social Networks*, 34:309–322.

Koskinen, J. H. and Snijders, T. A. B. (2007). Bayesian inference for dynamic social network data. *Journal of Statistical Planning and Inference*, 13:3930–3938.

Koskinen, J. H. and Snijders, T. A. B. (2023). Multilevel longitudinal analysis of social networks. *Journal of the Royal Statistical Society, Series A*, 186:376–400.

Krause, R. W., Huisman, M. E., and Snijders, T. A. B. (2018). Multiple imputation for longitudinal

network data. *Italian Journal of Applied Statistics*, 30:33–57.

Kushner, H. J. and Yin, G. G. (2003). *Stochastic Approximation and Recursive Algorithms and Applications*. Springer, New York, second edition.

Lepkowski, J. M. (1989). Treatment of wave nonresponse in panel surveys. In Kasprzyk, D., Duncan, G., Kalton, G., and Singh, M. P., editors, *Panel Surveys*, pages 348–374. Wiley, New York.

Lomi, A., Snijders, T. A. B., Steglich, C., and Torló, V. J. (2011). Why are some more peer than others? Evidence from a longitudinal study of social networks and individual academic performance. *Social Science Research*, 40:1506–1520.

Lospinoso, J. A. (2012). *Statistical Models for Social Network Dynamics*. PhD thesis, University of Oxford, U.K.

Lospinoso, J. A., Schweinberger, M., Snijders, T. A. B., and Ripley, R. M. (2011). Assessing and accounting for time heterogeneity in stochastic actor oriented models. *Advances in Data Analysis and Computation*, 5:147–176.

Lospinoso, J. A. and Snijders, T. (2019). Goodness of fit for stochastic actor-oriented models. *Methodological Innovations*, 12:1–18.

Michell, L. and West, P. (1996). Peer pressure to smoke: the meaning depends on the method. *Health Education Research*, 11:39 –49.

Niezink, N. M. D. (2018). *Modeling the Dynamics of Networks and Continuous Behavior*. PhD thesis, University of Groningen, Groningen.

O'Hagan, A. and Forster, J. (2004). *Bayesian Inference*, volume 2B of *Kendall's Advanced Theory of Statistics*. Arnold, London.

Pearson, M. A. and West, P. (2003). Drifting smoke rings: Social network analysis and Markov processes in a longitudinal study of friendship groups and risk-taking. *Connections*, 25(2):59–76.

R Core Team (2025). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Rao, C. R. (1947). Large sample tests of statistical hypothesis concerning several parameters with applications to problems of estimation. *Proceedings of the Cambridge Philosophical Society*, 44:50–57.

Robbins, H. and Monro, S. (1951). A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407.

Robins, G. L. and Alexander, M. (2004). Small worlds among interlocking directors: network structure and distance in bipartite graphs. *Computational & Mathematical Organization Theory*, 10:69–94.

Robins, G. L., Pattison, P. E., and Wang, P. (2009). Closure, connectivity and degree distributions: Exponential random graph (p^*) models for directed social networks. *Social Networks*, 31:105–117.

Schwabe, R. and Walk, H. (1996). On a stochastic approximation procedure based on averaging. *Metrika*, 44:165–180.

Schweinberger, M. (2012). Statistical modeling of network panel data: Goodness-of-fit. *British Journal of Statistical and Mathematical Psychology*, 65:263–281.

Schweinberger, M. and Snijders, T. A. B. (2007a). Bayesian inference for longitudinal data on social networks and other outcome variables. Working Paper.

Schweinberger, M. and Snijders, T. A. B. (2007b). Markov models for digraph panel data: Monte Carlo-based derivative estimation. *Computational Statistics and Data Analysis*, 51(9):4465–4483.

Shannon, C. E. (1948). A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423, 623–656.

Snijders, T. A. B. (1996). Stochastic actor-oriented dynamic network analysis. *Journal of Mathematical Sociology*, 21:149–172.

Snijders, T. A. B. (2001). The statistical evaluation of social network dynamics. *Sociological Methodology*, 31:361–395.

Snijders, T. A. B. (2004). Explained variation in dynamic network models. *Mathématiques, Informatique et Sciences Humaines / Mathematics and Social Sciences*, 168(4).

Snijders, T. A. B. (2005). Models for longitudinal network data. In Carrington, P., Scott, J., and Wasserman, S., editors, *Models and Methods in Social Network Analysis*, chapter 11, pages 215–247. New York: Cambridge University Press.

Snijders, T. A. B. (2013). Variance reduction in the Robbins-Monro procedure in RSiena. http://www.stats.ox.ac.uk/~snijders/siena/Dolby_s.pdf. Paper presented at the 9th UKSNA Conference, June 27–28, 2013, Greenwich.

Snijders, T. A. B. (2017). Stochastic actor-oriented models for network dynamics. *Annual Review of Statistics and Its Application*, 4:343–363.

Snijders, T. A. B. and Baerveldt, C. (2003). A multilevel network study of the effects of delinquent behavior on friendship evolution. *Journal of Mathematical Sociology*, 27:123–151.

Snijders, T. A. B. and Bosker, R. J. (2012). *Multilevel Analysis: An Introduction to Basic and Advanced Multilevel Modeling*. Sage, London, 2nd edition.

Snijders, T. A. B., Koskinen, J. H., and Schweinberger, M. (2010a). Maximum likelihood estimation for social network dynamics. *Annals of Applied Statistics*, 4:567–588.

Snijders, T. A. B. and Lomi, A. (2019). Beyond homophily: Incorporating actor variables in statistical network models. *Network Science*, 7:1–19.

Snijders, T. A. B., Lomi, A., and Torló, V. (2013). A model for the multiplex dynamics of two-mode and one-mode networks, with an application to employment preference, friendship, and advice. *Social Networks*, 35:265–276.

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

Snijders, T. A. B. and Pickup, M. (2017). Stochastic actor-oriented models for network dynamics. In Victor, J. N., Lubell, M., and Montgomery, A. H., editors, *Oxford Handbook of Political Networks*, pages 221–247. Oxford University Press, Oxford.

Snijders, T. A. B. and Steglich, C. E. G., editors (2023). *Social Network Dynamics by Examples. In preparation*, Cambridge University Press, Cambridge.

Snijders, T. A. B. and Steglich, C. E. G. (2026). *Actor-based Models for Analyzing Network Dynamics. In preparation*, Cambridge University Press, Cambridge. In preparation.

Snijders, T. A. B., Steglich, C. E. G., and Schweinberger, M. (2007). Modeling the co-evolution

of networks and behavior. In van Montfort, K., Oud, H., and Satorra, A., editors, *Longitudinal models in the behavioral and related sciences*, pages 41–71. Mahwah, NJ: Lawrence Erlbaum.

Snijders, T. A. B., van de Bunt, G. G., and Steglich, C. (2010b). Introduction to actor-based models for network dynamics. *Social Networks*, 32:44–60.

Snijders, T. A. B. and van Duijn, M. A. J. (1997). Simulation for statistical inference in dynamic network models. In Conte, R., Hegselmann, R., and Terna, P., editors, *Simulating Social Phenomena*, pages 493–512. Springer, Berlin.

Stadtfeld, C., Snijders, T. A., Steglich, C. E., and van Duijn, M. A. (2020). Statistical power in longitudinal network studies. *Sociological Methods & Research*, 49:1103–1132.

Steglich, C. E. G., Snijders, T. A. B., and Pearson, M. A. (2010). Dynamic networks and behavior: Separating selection from influence. *Sociological Methodology*, 40:329–393.

van de Bunt, G. G. (1999). *Friends by choice; An actor-oriented statistical network model for friendship networks through time*. Amsterdam: Thesis Publishers.

van de Bunt, G. G., van Duijn, M. A. J., and Snijders, T. A. B. (1999). Friendship networks through time: An actor-oriented statistical network model. *Computational and Mathematical Organization Theory*, 5:167–192.

Veenstra, R., Dijkstra, J. K., Steglich, C., and Van Zalk, M. H. (2013). Network–behavior dynamics. *Journal of Research on Adolescence*, 23(3):399–412.

Viechtbauer, W. (2005). Bias and efficiency of meta-analytic variance estimators in the random-effects model. *Journal of Educational and Behavioral Statistics*, 30:261–293.

Viechtbauer, W. (2010). Conducting meta-analyses in R with the metafor package. *Journal of Statistical Software*, 36(3):1–48.

Wasserman, S. and Faust, K. (1994). *Social Network Analysis: Methods and Applications*. New York and Cambridge: Cambridge University Press.

Zandberg, T. and Huisman, M. E. (2019). Missing behavior data in longitudinal network studies: The impact of treatment methods on estimated effect parameters in stochastic actor oriented models. *Social Network Analysis and Mining*, 9.8.

Zeggelink, E. P. (1994). Dynamics of structure: an individual oriented approach. *Social Networks*, 16:295–333.

Žnidaršič, A. (2012). Impact of fixed choice design on blockmodeling outcomes. *Advances in Methodology & Statistics/Metodološki zvezki*, 9(2):139–153.