Manual for RSiena

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

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

May 28, 2016

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 Snijders (2001, 2005), Snijders et al. (2007), and Snijders et al. (2010a). 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 earlier, for the manual for SIENA version 3, by Mark Huisman, Michael Schweinberger, and Christian Steglich.

This manual is frequently updated, mostly only in a minor way. This version was renewed for RSiena version 1.1-294.

Contents

1 General information **6**

I Minimal Intro **8**

1.1 Giving references . . . 8

2 Getting started with SIENA **9**

2.1 The logic of Stochastic Actor-Oriented Models . . . 9
2.1.1 Types of Stochastic Actor-Oriented Models . . . 10
2.1.2 Data, variables and effects . . . 11
2.1.3 Outline of estimation procedure . . . 15
2.1.4 Further useful options in RSiena . . . 16
2.2 Installing R and SIENA . . . 16
2.3 Using SIENA within R . . . 17
2.4 Example R scripts for getting started . . . 19
2.5 Steps for looking at results: Executing SIENA. . . . 19
2.6 Getting help with problems . . . 20

II Users' manual **22**

3 Steps of modelling **22**

4 Input data **23**

4.1 Data types . . . 23
4.1.1 Network data . . . 23
4.1.2 Transformation between matrix and edge list formats . . . 25
4.1.3 Behavioral data . . . 26
4.1.4 Individual covariates . . . 26
4.1.5 Dyadic covariates . . . 27
4.2 Internal data treatment . . . 28
4.2.1 Interactions and dyadic transformations of covariates . . . 28
4.2.2 Centering . . . 28
4.2.3 Monotonic dependent variables . . . 30
4.3 Further data specification options . . . 30
4.3.1 Structurally determined values . . . 30
4.3.2 Missing data . . . 32
4.3.3 Composition change: joiners and leavers . . . 33

5 Model specification **36**

5.1 Definition of the model . . . 36
5.1.1 Elementary effects . . . 38
5.1.2 Specification in SIENA . . . 38
5.1.3 Mathematical specification . . . 39
5.2 Important structural effects for network dynamics: one-mode networks . . . 40
5.3 Important structural effects for network dynamics: two-mode networks . . . 43

5.4 Effects for network dynamics associated with covariates . . . 44
5.5 Cross-network effects for dynamics of multiple networks . . . 46
5.6 Effects on behavior evolution . . . 47
5.7 Model Type: non-directed networks . . . 48
5.8 Additional interaction effects . . . 49
5.8.1 Interaction effects for network dynamics . . . 50
5.8.2 Interaction effects for behavior dynamics . . . 51
5.9 Time heterogeneity in model parameters . . . 52
5.10 Limiting the maximum outdegree . . . 52
5.11 Goodness of fit with auxiliary statistics . . . 53
5.11.1 Treatment of missing data and structural values in sienaGOF . . . 54

6 Estimation **55**
6.1 The estimation function siena07 . . . 55
6.1.1 Initial Values . . . 57
6.1.2 Convergence Check . . . 58
6.1.3 Continued estimation to obtain convergence . . . 59
6.2 Use of the algorithm parameters . . . 61
6.3 What to do if there are convergence problems . . . 62
6.4 Some important components of the sienaFit object . . . 63
6.5 Algorithm . . . 65
6.6 Output . . . 66
6.6.1 Convergence check . . . 67
6.6.2 Parameter values and standard errors . . . 67
6.6.3 Collinearity check . . . 68
6.7 Other estimation procedures . . . 69
6.8 Generalized Method of Moments estimation . . . 70
6.9 Maximum Likelihood and Bayesian estimation . . . 71
6.10 Other remarks about the estimation algorithm . . . 72
6.10.1 Conditional and unconditional estimation . . . 72
6.10.2 Fixing parameters . . . 73
6.10.3 Automatic fixing of parameters . . . 73
6.10.4 Required changes from conditional to unconditional estimation . . . 74
6.11 Using multiple processes . . . 74

7 Standard errors **76**
7.1 Multicollinearity . . . 76
7.2 Precision of the finite differences method . . . 77

8 Tests **78**
8.1 Wald-type tests . . . 78
8.1.1 Standard errors of linear combinations . . . 80
8.2 Score-type tests . . . 81
8.3 Example: one-sided tests, two-sided tests, and one-step estimates . . . 82
8.3.1 Multi-parameter tests . . . 83
8.4 Alternative application: convergence problems . . . 84
8.5 Testing differences between independent groups . . . 84
8.6 Testing time heterogeneity in parameters . . . 85

9 Simulation **87**
9.1 Accessing the generated networks . . . 88
9.2 Conditional and unconditional simulation . . . 89

10 Getting started **90**
10.1 Model choice . . . 91

11 Multilevel network analysis **92**
11.1 Multi-group Siena analysis . . . 93
11.2 Meta-analysis of Siena results . . . 94
11.2.1 Meta-analysis directed at the mean and variance of the parameters . . . 95
11.2.2 Meta-analysis directed at testing the parameters . . . 97
11.2.3 Contrast between the two kinds of meta-analysis . . . 98
11.3 Random coefficient multilevel Siena analysis . . . 98
11.3.1 Which data sets to use for sienaBayes . . . 99
11.3.2 Model specification . . . 100
11.3.3 How to enter your data in sienaBayes . . . 100
11.3.4 How to choose the parameter settings for sienaBayes . . . 101
11.3.5 Prior distributions . . . 101
11.3.6 Operation of sienaBayes() . . . 103
11.3.7 Assessing convergence . . . 103
11.3.8 Interpreting results of sienaBayes . . . 104

12 Formulas for effects **106**
12.1 Network evolution . . . 106
12.1.1 Network evaluation function . . . 107
12.1.2 Multiple network effects . . . 125
12.1.3 Network creation and endowment functions . . . 136
12.1.4 Network rate function . . . 137
12.2 Behavioral evolution . . . 139
12.2.1 Behavioral evaluation function . . . 139
12.2.2 Behavioral creation function . . . 149
12.2.3 Behavioral endowment function . . . 150
12.2.4 Behavioral rate function . . . 150
12.3 Effects for estimation by Generalized Method of Moments . . . 152

13 Parameter interpretation **153**
13.1 Networks . . . 153
13.2 Behavior . . . 154
13.3 Ego – alter selection tables . . . 155
13.4 Ego – alter influence tables . . . 161

14 Error messages **165**
14.1 During estimation . . . 165
14.2 As result of a score-type test (including time test) . . . 166
14.3 In sienaGOF . . . 167

III Programmers' manual **168**

15 Get the source code **168**

16 Other tools you need **168**

17 Building, installing and checking the package **169**

18 Understanding and adding an effect **170**
18.1 Example: adding the truncated out-degree effect 172
18.2 Notes on effectGroups and two-mode networks 176

A List of Functions in Order of Execution **178**

B Changes compared to earlier versions **189**

C References **214**

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), and Greenan (2015); also see Steglich et al. (2010). A tutorial for these models is in Snijders et al. (2010b).

Snijders and Steglich (2017b,a)

A website for SIENA is maintained at http://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.

This is a manual for RSiena, which also may be called SIENA version 4.0; the manual is provisional in the sense that it is continually being updated, taking account of updates in the package. RSiena is a contributed package for the R statistical system which can be downloaded from http://cran.r-project.org. For the operation of R, the reader is referred to the corresponding literature and help pages.

RSiena was originally programmed by Ruth Ripley and Krists Boitmanis, in collaboration with Tom Snijders. Since May 2012 the maintainer is Tom Snijders. Further contributions were made by Josh Lospinoso, Charlotte Greenan, Felix Sch"onenberger, Christian Steglich, Johan Koskinen, Mark Ortmann, Nynke Niezink, and Robert Hellpap.

In addition to the 'official' R distribution of RSiena, there is an additional distribution at R-Forge, which is a central platform for the development of R packages offering facilities for source code management. It is quite usual that later versions of RSiena are available at http://r-forge.r-project.org/R/?group_id=461 before being incorporated into the R package that can be downloaded from CRAN. In addition, at R-Forge there is a package RSienaTest which may include additional options that are still in the testing stage. Some of the options described in this manual may apply to RSienaTest only, with the plan to transfer this to RSiena in the future.

[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).

The development of RSiena was carried out by researchers of the Universities of Oxford, Groningen, and Konstanz. 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.

Currently, RSiena is maintained by a research group at the University of Groningen, seconded by researchers in Zürich, Konstanz, and Manchester.

Part I
Minimal Intro

1.1 Giving references

When using SIENA, it is appreciated that you refer to this manual and to one or more relevant references of the methods implemented in the program. The reference to this manual is the following.

Ruth M. Ripley, Tom A.B. Snijders, Zsófia Boda, András Vörös, and Paulina Preciado, 2016. Manual for SIENA version 4.0 (version May 28, 2016). Oxford: University of Oxford, Department of Statistics; Nuffield College. http://www.stats.ox.ac.uk/siena/

A tutorial is Snijders et al. (2010b). 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). A basic reference for the Bayesian estimation is Koskinen and Snijders (2007) and for the maximum likelihood estimation Snijders et al. (2010a).

More specific 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 the model for diffusion of innovations in dynamic networks, please refer to Greenan (2015). For assessing and correcting time heterogeneity, and goodness of fit assement and associated model selection considerations, refer to Lospinoso et al. (2011) and Lospinoso (2012). For Generalized Method of Moments estimators, the reference is Amati et al. (2015).

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 Modelcan 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 SIENA 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 undirected 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 SIENA 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 5.9 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 4.3.1 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 or ordinal discrete (the extension for continuous behavioral data is currently being developed). 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, 2015): 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 12.2.4.

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 4.3.3 for more details). In the second case, missing values are treated as randomly missing (see Section 4.3.2 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, SIENA 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.

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

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'

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 odds (more precisely: they are components of the linear predictor for the log-odds) for the creation, maintenance or presence of network ties. Table 2 helps to imagine what the odds 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

a) evaluation

t_1	t_2
$i \quad j$	$i \to j$
$i \to j$	$i \to j$
$i \to j$	$i \quad j$
$i \quad j$	$i \quad j$

b) creation

t_1	t_2
$i \quad j$	$i \to j$
$i \to j$	$i \to j$
$i \to j$	$i \quad j$
$i \quad j$	$i \quad j$

c) endowment

t_1	t_2
$i \quad j$	$i \to j$
$i \to j$	$i \to j$
$i \to j$	$i \quad j$
$i \quad j$	$i \quad j$

According to this distinction, network evolution may be modeled in SIENA 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-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 SIENA 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

SIENA estimates parameters by the function siena07() and (alternatively) sienacpp(), 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 siena07() (or sienacpp()). See the help page for siena07().

2.1.4 Further useful options in RSiena

- Checking for time heterogeneity (Sections 5.9 and 8.6)
- Goodness of fit (Section 5.11)
- Meta-analysis of SIENA results (Section 11.2)
- Simulation without estimation (Section 9)

2.2 Installing R and SIENA

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`.

2. Install the package RSiena or RSienaTest, with dependencies. The other packages used are tcltk, parallel and tools (all included in the basic R distribution); Matrix, MASS, lattice, codetools ('recommended' packages included in most R distributions); network and xtable; and, for RSienaTest, RUnit. For goodness of fit testing it will be useful also to install sna and igraph.
   You can just install RSiena and the other packages in the regular way from CRAN. However, it is advisable to have the latest version of RSiena or RSienaTest from R-Forge or the SIENA website. You can go to
   http://r-forge.r-project.org/R/?group_id=461
   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 can be done in various ways — by the function `install.packages()` in R, via the drop-down menu in the R console, or in command mode which for Mac is the 'terminal'. If a binary file is available (`.zip` for Windows, `.tgz` for Mac), then using the binary is recommended. Installation from binary is much faster than installation from source.

   Installation from the R-Forge repository can be done as follows. In these commands, `RSienaTest` can be replaced by `RSiena`.

   - for Windows:
     `install.packages("RSienaTest", repos="http://R-Forge.R-project.org")`
   - for Mac the binary file code is not available on R-Forge, but the source code may also work:

`install.packages("RSienaTest", repos="http://R-Forge.R-project.org", type = "source")`
If this does not work, try one of the following methods.

Installation from a downloaded file can be done as follows, assuming the root name of the file is `RSienaTest_1.1-294`, and filling in the correct path name. It will be convenient to first navigate to the directory containing the RSiena binary or source file so that this is the current directory. Then the pathname consists only of the filename.

- In R from binary:
  for Windows:
  `install.packages("pathname to RSienaTest_1.1-294.zip", repos = NULL, type="binary")`
  for Mac:
  `install.packages("pathname to RSienaTest_1.1-294.tgz", repos = NULL, type="binary")`
- In R from source:
  `install.packages("RSienaTest_1.1-294.tar.gz", repos = NULL, type="source")`
- In command.com or in batch mode (Windows) from binary:
  `R CMD INSTALL RSienaTest_1.1-294.zip`
- In the terminal (Mac) from binary:
  `R CMD INSTALL RSienaTest_1.1-294.tgz`
- In command.com or in batch mode (Windows) or in the terminal (Mac) from source:
  `R CMD INSTALL RSienaTest_1.1-294.tar.gz`
- In drop-down menu in R:
  for Windows: go to Packages → Install package(s) from local zip file
  for Mac: go to Packages & Data → Package Installer
- In RStudio:
  go to Tools → Install packages → Install From: Package archive file (zip; tar.gz)

2.3 Using SIENA within R

1. Load data (networks, behavior, covariates) into R (see Section 4.1):
   (a) Network data should be in objects of class matrix or sparse matrix (edgelist);
   (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.

2. All missing data should be set to NA (see Section 4.3.2).

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 SIENA objects for each data object using the appropriate functions (see Section 4.1):

   (a) sienaDependent() for networks and behavior variables;

   (b) only for two-mode networks, sienaNodeSet() for defining nodesets;

   (c) coCovar() and varCovar() for constant and changing/varying individual covariates respectively;

   (d) coDyadCovar() and varDyadCovar() for constant and changing/varying dyadic covariates respectively;

   (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 SIENA objects specified above using the function sienaDataCreate() (see Section 4.1).

6. Use getEffects() to create an effects object. This already gives a very simple model specification containing the outdegree and a reciprocity effects (see Section 5.2 - for two-mode networks see Section 5.3).

7. Use sienaAlgorithmCreate() to create an algorithm object (see Section 5).

8. Use print01Report() to produce an output file presenting some descriptive statistics for the objects included in the model.

9. Use functions includeEffects(), setEffect() and includeInteraction() to further specify the model (see Sections 5.2 – 5.6).

10. Use siena07() or sienacpp() to run the estimation procedure.[3]

11. Basic output is written to a log file in the actual working directory. The filename is the project name specified in the sienaAlgorithmCreate() function. Results can also be inspected in R using various functions.

[2] For directions on how to handle composition change, see Section 4.3.3

[3] The use of multiple processes can speed up the estimation. For directions on how to utilize multiple processors, see Section 6.11.

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 siena07() 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 SIENA 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;
- 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 print01Report() 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 distance 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 algorithm 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 6.1.2.

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 sienaAlgorithmCreate. 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 6.6 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 `http://groups.yahoo.com/groups/stocnet/` .

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 http://r-forge.r-project.org/R/?group_id=461. The version is identified by a version number (e.g. 1.1-294) and an R-Forge revision number. You can find both numbers of your current installed version by opening R , and typing `packageDescription("RSiena")`. The version is near the top, the revision number near the end. Both are also displayed at the start of SIENA output files produced by print01Report().

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 R-Forge. The latest version will always be available from R-Forge. (Frequent updates are discouraged on CRAN, so bug-fixes are likely to appear first on R-Forge.)

Installation When using the repository at R-Forge, *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.

R-Forge help list If you are a programmer, then for technical questions about the RSiena code (as distinct from the methodology), you can send an email to rsiena-help@lists.r-forge.r-project.org, or post in the help forum for RSiena in R-Forge. You need to be a registered member of R-Forge (and possibly of RSiena) to post to a forum, but anyone can send emails (at present!). In your message, please tell us which operating system, which version of R, and which version of RSiena you are using.

Part II
Users' manual

3 Steps of modelling

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

1. input of basic data description (see Section 4),
2. model specification (see Section 5),
3. estimation of parameter values using stochastic simulation (see Section 6),
4. testing parameters and assessing goodness of fit (see Sections 7 and 8),
5. simulation of the model with given and fixed parameter values (see Section 9).

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.

The main output of the estimation procedure is written to a text file named *pname*.out, where *pname* is the name specified in the call of sienaAlgorithmCreate().

4 Input data

SIENA 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 4.1. Section 4.2 describes the most commonly occuring data transformations that are done internally by SIENA. Finally, Section 4.3 shows further options for users to define their data.

4.1 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. SIENA 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 sienaDependent(), coCovar(), varCovar(), coDyadCovar(), varDyadCovar(), or sienaCompositionChange(). Second, the variables must be combined into one RSiena data set by the function sienaDataCreate(). 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.

4.1.1 Network data

For data specification by the sienaDependent function, the network must be specified as a matrix or array or list of sparse matrix of triples.

For data specification by the graphical interface siena01Gui (documented separately) or by the function sienaDataCreateFromSession, 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 sienaDependent(), have 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 sienaDependent and in sienaDataCreateFromSession.
   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 sienaDependent shows by examples how the specification can be given by sparse matrices.

2. *Pajek format.*
   These can be used in sienaDataCreateFromSession.
   If the digraph data file has extension name `.net`, then the program assumes that the data file has Pajek format. The file should relate to one observation only, and should contain a list of vertices (using the keyword `*Vertices`, together with (currently) a list of arcs, using the keyword `*Arcs` followed by data lines according to the Pajek rules. These keywords must be in lines that contain no further characters. An example of such input files is given in the s50 data set that is distributed in the `examples` directory of the source code.

3. *Siena format.*
   These can be used in sienaDataCreateFromSession.
   An edge list is a matrix containing three or four columns: from, to, value, wave (optional).
   Like the Pajek format, this has the advantage that absent ties (tie variables with the value 0) do not need to be mentioned in the data matrix. By specifying the waves in the fourth column in the Siena format, one matrix can be used to contain data for all the waves.

Missing values must be indicated in the way usual for R, by `NA`. For data specification by the graphical interface siena01Gui or by the function sienaDataCreateFromSession, instead of `NA` any numerical code can be used given that this is indicated to be a missing value code.

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 4.2.3 and note the possibility of using allowOnly=TRUE.

4.1.2 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]] <- edges[, 3]
```

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.

4.1.3 Behavioral data

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

If any values are not integers, a warning will be printed on the initial report given by print01Report() and the values will be truncated towards zero.

A special case of behavioral data can be used for *diffusion of innovations* (Greenan, 2015): 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 12.2.4.

4.1.4 Individual covariates

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

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 observa-

tion 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 varCovar, using for input an $n \times (M-1)$ matrix where the columns correspond to the $M-1$ periods between observations.

The mean is always subtracted from the covariates. See Section 4.2.2 on centering.

When an actor covariate is constant within waves, i.e., within each wave it has the same value for all actors; or, more generally, when within each wave it has the same value for all actors within components separated by structural zeros (which means that ties between such components are not allowed), then only the ego effect of the actor covariate is made available. This is because the other effects then are meaningless. This may cause problems for combining several data sets in a multi-group project (see Section 11). If at least one case is missing (i.e., has the missing value data code), then the other covariate effects are made available. When analysing multiple data sets in parallel, for which the same set of effects is desired to be included, it is therefore advisable to give data sets in which a given covariate has the same value for all actors one missing value in this covariate; purely to make the total list of effects independent of the observed data.

4.1.5 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 dyadic covariate, a $n \times n \times (M-1)$ array is needed.

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 coDyadCovar, and changing dyadic covariates by varDyadCovar.

The mean is always subtracted from the covariates. See Section 4.2.2 on centering.

4.2 Internal data treatment

4.2.1 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. 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*, defined by $1 - \big(|v_i - v_j|/r_V\big)$, and centered so the mean of this similarity variable becomes 0;
   note that before centering, the similarity variable is 1 if the two actors have the same value, and 0 if one has the highest and the other the lowest possible value; the mean of the similarity variable is calculated by function sienaDataCreate and stored as the simMean attribute of `mydata$cCovars$myvar`, where `mydata` is the name of the object created by sienaDataCreate, and `myvar` is the name of the variable used as the argument for sienaDataCreate, 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 4.2.2.

2. *same* V, defined by 1 if $v_i = v_j$, and 0 otherwise (not centered) (V is the name of the variable). 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, SIENA offers the possibility of user-defined two- and three-variable interactions between covariates; see Section 5.8.

4.2.2 Centering

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

For individual covariates, the mean value is subtracted by function SienaDataCreate. 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 print01Report(). For the multi-group option (section 11.1), the subtracted values are the global means across all groups.

Centering of covariates can be turned off by specifying `centered=FALSE` in the call of coCovar(), varCovar(), coDyadCovar(), or varDyadCovar(), respectively.

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 projects, 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 11.1), dyadic covariates are treated differently from individual covariates: for dyadic covariates in multi-group projects, centering is done by the within-group mean; actor covariates in multi-group projects are centered by the overall mean.

For dependent behavioral variables, the effects are defined in Section 12.2 as functions of centered variables.

The means of covariates are stored as attributes on the object created by SienaDataCreate. 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 <- sienaDataCreate( 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`

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 12. It is also reported in the output and available – 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.

4.2.3 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 print01Report. 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 sienaDependent (see the help file for this function).

If a dependent variable is only increasing or only decreasing for all periods and sienaDependent 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 sienaDependent with allowOnly=FALSE.

4.3 Further data specification options

4.3.1 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 SIENA 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.

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 4.3.3. 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.

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 print01Report — 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 6.10.1) is preferable.

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 given in Section 11. 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 SIENA 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 5.4. This is because the other effects then are meaningless. If at least one case is missing, then the other covariate effects are made available.

When SIENA 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 struc-

turally 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.

4.3.2 Missing data

SIENA 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 enable 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 coCovar() or varCovar(), 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 coCovar() and varCovar(), 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.

4.3.3 Composition change: joiners and leavers

SIENA 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 4.3.1. 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 6.10.1).

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 sienaCompositionChangeFromFile() in case a file is used, and sienaCompositionChange() 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 sienaCompositionChangeFromFile().

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 <- sienaCompositionChange(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 sienaCompositionChangeFromFile or sienaCompositionChange is of class compositionChange and can be used in the function sienaDataCreate.

The method of joiners and leavers for representing composition change does not combine properly with the sienaGOF function (Section 5.11).

5 Model specification

5.1 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 size or activity differences between actors, it is possible that different advice must be given, and it may be necessary to let the rate function depend on the individual covariate that indicates this size; or on the out-degree.)

- *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, with the same parameters. For any given effect, normally 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.

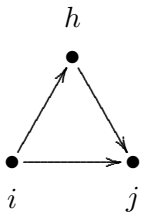
[6] A special case of the *gratification function* in Snijders (2001).

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

5.1.1 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 12.



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 includeEffects() and setEffect() are only `eval`, `creat`, and `endow`. In Chapter 12 almost all effects are evaluation effects, and the effects that are elementary (and not evaluation) effects are mentioned as such.

5.1.2 Specification in SIENA

The model specification is defined in SIENA by the so-called *effects object*, which formally is an object of class sienaEffects or, for multiple groups as discussed in Chapter 11 of class sienaGroupEffects. This object is originally created by the function getEffects and subsequently modified by the functions includeEffects and/or setEffects. The scripts on the SIENA website give examples. An important ingredient here is the so-called shortName of each effect, which is used to identify it; effects of covariates 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 variable name (name) also is required to specify the effect. The shortNames are part of the effects object. For the practical use of SIENA, the shortNames are important. A list of effects with their shortNames can be displayed in a browser by using the function:

```
effectsDocumentation()
```

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`.

5.1.3 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. The probability that the next network is x is given by

$$\frac{\exp\bigl(u_i(x^0, x)\bigr)}{\sum_{x' \in \mathcal{C}} \exp\bigl(u_i(x^0, x')\bigr)} \;. \tag{1}$$

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,

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

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

$$\begin{aligned} u_i(x^0, x) \;=& \big(f_i(x) - f_i(x^0)\big) \\ &+\; \Delta^+(x^0, x)\,\big(c_i(x) - c_i(x^0)\big) \;+\; \Delta^-(x^0, x)\,\big(e_i(x) - e_i(x^0)\big) \;. \end{aligned} \tag{2}$$

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).

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

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 12. Here we give a more qualitative description.

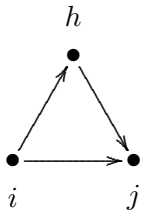
A default model choice could consist of (1) the out-degree and reciprocity effects; (2) one network closure effect, e.g. transitive triplets, transitive ties, or gwesp; the transitive

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.

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

1. The *out-degree effect* which always must be included.
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. Some researchers may find the last effect (distances two) less appealing because it expresses network closure inversely.

   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 $i \rightarrow j$ is proportional to the total number of transitive triplets that it forms – which can be transitive triplets of the type $\{i \rightarrow j \rightarrow h;\ i \rightarrow h\}$ as well as $\{i \rightarrow h \rightarrow j;\ i \rightarrow j\}$;



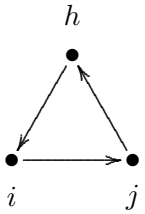
   b. The *balance effect*, which may also be called *structural equivalence with respect to outgoing ties*. This expresses a preference of actors to have ties to those other actors who have a similar set of outgoing ties as themselves. Whereas the transitive triplets effect focuses on how many same choices are made by ego (the focal actor) and alter (the other actor) — the number of h for which $i \rightarrow h$ and $j \rightarrow h$, i.e., $x_{ih} = x_{jh} = 1$ where i is ego and j is alter — , the balance effect considers in addition how many the same non-choices are made — $x_{ih} = x_{jh} = 0$.

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

   d. The *gwesp* effect (see later in this manual).

   d. 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.

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.


Block (2015) has argued convincingly that instead of the three-cycles effect, it is often advisable to use 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 $h \not\to j$.

⊙ The following eight 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.

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 *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.

11. 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.

12. 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.

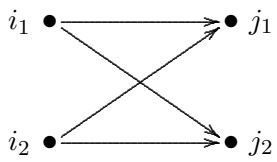
13. The *out-out degree assortativity effect* (with parameters 2 or 1 in similar roles) reflects tendencies for actors with high out-degrees to preferably be tied to other actors with high out-degrees.

5.3 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 12. 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.

5.4 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 is 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 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. In the current version of SIENA, there are no effects of dyadic covariates on behavioral evolution.

5.5 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 $i \overset{W}{\to} j$ promotes the creation or maintenance of a tie $i \overset{X}{\to} j$.

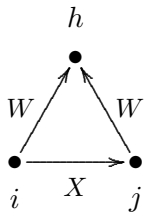
2. If both networks are one-mode, then a next effect is the reciprocity effect with W on X, representing the extent to which the existence of a tie $j \overset{W}{\to} i$ promotes the creation or maintenance of a tie, in the reverse direction, $i \overset{X}{\to} j$.

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 $i \overset{W}{\leftrightarrow} j$ promotes the creation or maintenance of a tie $i \overset{X}{\to} j$.

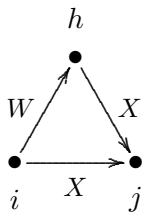
4. The *outdegree W activity effect* (where parameter 2 is the sqrt version, while parameter 1 is the non-sqrt version – see above for explanations of this) reflects the extent to which actors with high outdegrees on W will make more choices in the X network.

⊙ Several mixed transitivity effects can be important.

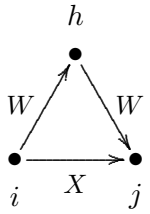
5. If X is a one-mode network, the *from W agreement* effect represents the extent to which agreement between i and j with respect to outgoing W-ties promotes the creation or maintenance of a tie $i \overset{X}{\to} j$.



6. If W is a one-mode network, the *W to agreement* effect represents the extent to which a W tie $i \overset{W}{\to} h$ 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, $i \overset{X}{\to} j$ and $h \overset{X}{\to} j$.



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



5.6 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 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 5.8.

5.7 Model Type: non-directed networks

Non-directed networks are an undocumented option (there currently only is the presentation Snijders (2007).

SIENA 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 (2007). This is specified by the parameter modelType in function sienaAlgorithmCreate. 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.

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.

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.

5.8 Additional interaction effects

It is possible for the user to define additional interaction effects for the network and the behavior. The basis is provided by the initial definition, by SIENA, of 'unspecified interaction effects'. The interaction is defined by the columns `effect1` and `effect2`, and for three-way effects, `effect3`, in the effects object; they contain the `effectNumber` (sequence number) 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 getEffects().

Interactions can be specified by the function includeInteraction, explained in the following subsections.

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; an explanation of the background of this 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 "".

The information necessary for working with interaction effects – the interaction types, short names, and sequence numbers of the effects – are 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 12 of this manual also gives the short names of all effects. The short name of all unspecified interaction effects is `unspInt` for network effects, and `behUnspInt` for behaviour effects.

5.8.1 Interaction effects for network dynamics

The following kinds of user-defined interactions are possible for network dynamics.

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

b. Dyadic effects can interact with each other.

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 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.

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 includeInteraction are as follows.

```
myeff <- includeInteraction( myeff, egoX, recip,
                                    interaction1 = c("smoke1", "") )
myeff <- includeInteraction( myeff, egoX, egoX,
                                    interaction1 = c("smoke1", "alcohol") )
```

Note the `interaction1` parameter; this parameter is used also when defining these effects using includeEffects or setEffect. In this case, however, two effects are defined, and accordingly the `interaction1` parameter has two components, combined by the `c` function. For effects such as `recip` that have no `interaction1` parameter, the corresponding string is just the empty string, `""`. (Note that the name `interaction1` does not itself refer to interactions in the sense of this section.)

Interactions between three effects are defined similarly, but now the `interaction1` parameter must combine three components.

The list of effects in Chapter 12 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).

5.8.2 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 includeInteraction. 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 so-called `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 12.2. 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`.

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 <- includeEffects(myeff, avAlt,
                name="drinkingbeh", interaction1="friendship")
myeff <- includeInteraction(myeff, quad, avAlt,
                name="drinkingbeh", interaction1=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 drinking behavior on drinking behavior. Briefly, the interaction is between current drinking behavior and the average drinking behavior of friends. By consulting Section 12.2.1 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

[8] The value is `OK` for the effects of which the formula as defined in Section 12.2.1 is given by z_i multiplied by something not dependent on z_i.

shape effects are included in the model.

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

In addition, there are predefined interactions available between actor variables and influence, as described in Section 12.2.1.

5.9 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 sienaTimeTest function, as explained in Section 8.6. To specify a model with time heterogeneous parameters, the function includeTimeDummy can be used, as follows. Consider the reformulation of the evaluation function into

$$f_{ij}^{(m)}(\mathbf{x}) = \sum_k \left(\beta_k + \delta_k^{(m)} h_k^{(m)}\right) s_{ik}\big(\mathbf{x}(i \rightsquigarrow j)\big) \tag{3}$$

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 sienaTimeTest function, see Section 8.6.

See Lospinoso et al. (2011) for a technical presentation and examples of how the test works, and Lospinoso (2010) for a walkthrough on model selection.

5.10 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 sienaAlgorithmCreate function, which determines the settings of the algorithm.

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.

5.11 Goodness of fit with auxiliary statistics

There is now available in RSienaTest a function `sienaGOF` which 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 5.2 to 5.6 – and explained also in Snijders et al. (2010b) –, whether the end result gives a good representation also of these other statistics.

The sienaGOF function, proposed and elaborated by Lospinoso (2012), 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 results of `sienaGOF` 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.

The method of joiners and leavers for representing composition change (Section 4.3.3) does not combine properly with the sienaGOF function.

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

For combining sienaGOF 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.

5.11.1 Treatment of missing data and structural values in sienaGOF

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 sienaGOF. 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).

6 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); 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; the other two 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 the following, 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.

6.1 The estimation function siena07

The estimation process implemented in functions siena07() and sienacpp() starts with initial values for the parameters, and returns a so-called sienaFit object, in this example called `results1`, which contains the estimates and their standard errors and a lot of further information. Since the estimate is iterative (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.

Much of what follows is about the use of siena07() but applies equally to sienacpp(). The difference between these two functions is that estimation by sienacpp() stays entirely in the 'back end' C++ part of SIENA, contrasting with siena07() which carries out the simulations in C++ but the Robbins-Monro updates in the R 'front end' part; this yields greater computational efficiency for sienacpp(). Since the simulations take the largest amount of processing time for medium-sized and large networks, the time difference is notable, in a proportional sense, mainly for data sets where simulations run very quickly (i.e., number of actors and distance between first and last simulations are small). Further, results from sienacpp() cannot used for sienaGOF().

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

```
algorithm1    <-  sienaAlgorithmCreate(projname = "trypro", useStdInits = FALSE)
```

```
results1      <-  siena07(algorithm1, data = mydata, effects = myeff)
```

The function sienaAlgorithmCreate defines an algorithm specification object with options for the algorithm, and the function siena07 carries out the estimation. If you do not want to see the graphical interface with intermediate results, or if your computer has problems showing this, then add the option `batch = TRUE`, as in

```
results1  <-  siena07(algorithm1, data = mydata, effects = myeff,
                      batch = TRUE)
```

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  <-  siena07(algorithm1, data = mydata, effects = myeff,
                      verbose = TRUE)
```

The estimation produces an output file in the current working directory, of which the name is defined by the projname option; in this example, the name is `trypro.out`. 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,

```
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.

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 siena07(). Given that the earlier result was already called `results1`, this is done, e.g., by

```
results1  <-  siena07(algorithm1, data = mydata, effects = myeff,
                      prevAns = results1)
```

Further see below for more information about convergence.

6.1.1 Initial Values

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

1. The default: if `useStdInits = FALSE` and no prevAns parameter is given in the call of siena07, 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 parameter 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 updateTheta, 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 setEffect function in which the initialValue then must be set.

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

   ```
   results1 <- siena07(algorithm1, data = mydata, effects = myeff,
                       prevAns = results0)
   ```

the initial parameter estimates are taken from the results of what is given as the prevAns parameter. 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 sienaAlgorithmCreate, 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`.

6.1.2 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.

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') parameter, while taking care that `useStdInits = FALSE` has been specified. An example is

```
results1 <- siena07(algorithm1, 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.

6.1.3 Continued estimation to obtain convergence

Above, the prevAns parameter 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 parameters in the sienaAlgorithmCreate() function, to try and improve convergence. The main parameters of sienaAlgorithmCreate() that can be used for this purpose are the following. For the technical background, see Siena_algorithms.pdf which can be downloaded from the SIENA website.

- `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. Try `doubleAveraging=0`, which starts using this step from phase 2.1.

- `diagonalize`
  This parameter 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 (the default) gives greatest stability; smaller values may give greater efficiency. Very small values (less than 0.2) may lead to an unstable algorithm.
  Advice: try values such as 0.7, 0.5, or even 0.2; if the algorithm becomes unstable reduce `firstg` (e.g., to 0.005 or 0.001) but if this does not help, the value used for `diagonalize` is too small.

- `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. 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.

- `firstg`
  This determines the step sizes in the estimation algorithm. If the algorithm is unstable, use a smaller value (but greater than 0).

If convergence is not very good even with repeated estimation with the prevAns option, sometimes it can be useful to try and use updateTheta() 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.

The following function will iterate the execution of siena07() 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.

```
siena07ToConvergence <- function(alg, dat, eff, ans0=NULL, ...){
   numr <- 0
   ans <- siena07(alg, data=dat, effects=eff, prevAns=ans0, ...) # the first run
   repeat {
       save(ans, file=paste("ans",numr,".RData",sep="")) # to be safe
       numr <- numr+1           # count number of repeated runs
       tm <- ans$tconv.max      # convergence indicator
```

[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.

```
        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
        ans <- siena07(alg, data=dat, effects=eff, prevAns=ans, ...)
        }
    if (tm > 0.25)
    {
       cat("Warning: convergence inadequate.\n")
    }
    ans
}
```

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 parameter 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 6.3.

6.2 Use of the algorithm parameters

The parameters 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 parameter sequentially. The advice is to use (1.) or (2.), in both cases together with (3.).

1. In the estimations where prevAns is used, use more than 4 subphases: nsub=5 or, if that is not sufficient, nsub=6.

2. First do an estimation with the default parameters 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 n2start; and firstg clearly less than the default 0.2, e.g., about half the smallest standard error reported in the first estimation. What is 'a high value' depends on the data set - model combination, the number of parameters in the first place. The baseline value (see the preceding section[10]) would be $(p+7) \times (2.52)^4$, where p is the number of estimated parameters (excluding the rate parameters estimated by conditioning, if conditional estimation is used). For a 'high

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

value', start with a value that is about twice as large, Note that computation time in Phase 2 will be proportional to n2start. Continue estimation with these algorithmic parameters using prevAns, further perhaps increasing n2start if necessary.

3. Use for n3 a larger value than the default. E.g., n3 = 3000 or 5000; if it was necessary to further increase nsub or n2start, it may also be necessary to further increase n3.

All these choices clearly increase computing time; but less so, and with more likely success, than just carrying on very long with siena07ToConvergence() using the default algorithmic parameters.

6.3 What to do if there are convergence problems

If there are persisting convergence problems even after repeated estimations using the `prevAns` parameter 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 model 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 SIENA 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 triplers. This may help with convergence.

  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.

  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: 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 5.9 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 8.2) 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 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.

6.4 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 siena07.

`ans$theta`	estimates (but not for the rate parameter used for conditioning; if time dummies were requested using sienaTimeFix, these are also in `theta`).
`ans$covtheta`	covariance matrix of the estimates
`ans$se`	standard errors of the estimates
`ans$pp`	number of parameters
`ans$targets`	targets (observed statistics) for Method of Moments estimation
`ans$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 for Methods of Moments estimation (see Section 9.1), if returnDeps = TRUE in the call of siena07
`ans$estMeans`	estimated expected values of the target statistics (if the Dolby option was chosen, this is not equal to the average of the simulations!)
`ans$effects`	the effects object with only the requested effects
`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/Test version
`ans$revision`	the R-Forge revision.

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 `siena07ToConvergence` presented above.

6.5 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. The parameter values change from run to run, reflecting the deviations between generated and observed values of the statistics. The changes in the parameter values are smaller in the later subphases.
   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 6.9). 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 default number of runs in phase 3 is 1000. 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 sienaAlgorithmCreate.

During the estimation process, if the graphical user interface is used (the default `batch = FALSE` in the call of siena07), 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.

6.6 Output

Output can be obtained in several ways.

1. On the R console.
   When the sienaFit object produced by siena07 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. In package RSienaTest, the function siena.table is available which serves a similar purpose, but not using xtable. 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 siena07 writes an output file which is an ASCII ('text') file that can be read by any text editor. It is called *pname*.out, where *pname* is the name specified in the call of sienaAlgorithmCreate().

   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.

6.6.1 Convergence check

This was discussed in Section 6.1.2 above.

6.6.2 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 in section 12.1.4 below. 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 12.1.1, the reciprocity effect s_{i2}, transitive triplets effect s_{i3}, three-cycles effect s_{i5}, sex similarity effect s_{i65}. Therefore the estimated evaluation function here is

$$-2.53\, s_{i1}(x) \;+\; 2.11\, s_{i2}(x) \;+\; 0.54\, s_{i3}(x) \;+\; 0.08\, s_{i5}(x) \;+\; 0.45\, s_{i65}(x)$$

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 8.)

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.

6.6.3 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.

6.7 Other estimation procedures

SIENA 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); 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, 2015).

The Generalized Method of Moments is somewhat more time-consuming and is theoretically more efficient than the Method of Moments. On the other hand, function sienacpp() is coded in a more efficient way than siena07().

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 not to be no reason to use the more time-consuming ML or Bayesian methods. In not-so-nice situations (very small network data sets, small network and behavior data sets in combination with complex models), however, ML and Bayesian methods tend to produce somewhat more accurate results than MoM. Statistical theory suggests that ML is a more efficient estimation method than MoM in the sense of producing estimates with smaller standard errors. But in the 'nice situations' 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) is implemented in the functions siena07() and sienacpp(). Estimation by the Generalized Method of Moments (GMoM) is implemented in the function sienacpp().

ML estimation is done by the function siena07(), using a set of options created by sienaAlgorithmCreate() with maxlike=TRUE. Bayesian estimation is done by the function sienaBayes(), but is as yet implemented only for multilevel network modeling (Section 11.3).

6.8 Generalized Method of Moments estimation

Estimation by the Generalized Method of Moments (GMoM) was proposed in Amati et al. (2015) and is implemented in the function sienacpp().

If the algorithm object created by sienaAlgorithmCreate() uses `maxlike=FALSE`, the estimation function sienacpp() will use the MoM or the GMoM; the latter will be used as soon as at least one of the effects were specified in the effects object with `type='gmm'`.

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 includeEffects() 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 <- sienaDependent(array(c(s501,s502,s503), dim=c(50,50,3)))
dataset <- sienaDataCreate(net)
eff <- getEffects(dataset)
eff <- includeEffects(eff, density)
eff <- includeEffects(eff, density, type='gmm')
```

```
eff <- includeEffects(eff, recip)
eff <- includeEffects(eff, recip, realrecip, persistrecip, type='gmm')
eff <- includeEffects(eff, transTrip)
eff <- includeEffects(eff, 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 6.10.1). 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.

6.9 Maximum Likelihood and Bayesian estimation

ML estimation is done by the function siena07(), using a set of options created by sienaAlgorithmCreate() with `maxlike=TRUE`. Bayesian estimation is done by the function sienaBayes(). The following further information in this section is about ML estimation; Bayesian estimation is as yet implemented only for multilevel network modeling (see Section 11.3).

For ML estimation, the method of joiners and leavers (Section4.3.3) is not available. Further, R may run into an error (the program will hang) if there are any actors who are inactive at the first wave, as indicated by all structural zeros.

For ML estimation, an important parameter for tuning the algorithm is the so-called Multiplication factor, given in sienaAlgorithmCreate() as the argument `mult`. This determines the number of Metropolis-Hastings steps taken for simulating each new network.

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 networks 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 between 0.1 and 0.3 or 0.4.

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.

Another parameter 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.

6.10 Other remarks about the estimation algorithm

6.10.1 Conditional and unconditional estimation

SIENA 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.

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'.

Conditioning on the network variable means running simulations until the number of different entries between 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 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 sienaAlgorithmCreate function by setting the cond parameter. 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 parameters in this function.

If there are changes in network composition (see Section 4.3.3), only the unconditional estimation procedure is available.

If there are a lot of structurally determined values (see Section 4.3.1) then unconditional estimation is preferable.

6.10.2 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 setEffect function with the fix = TRUE option.

6.10.3 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 model 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 10.1).

6.10.4 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 initial parameter values were chosen in an unfortunate way such that the conditional simulation does not succeed in ever attaining the condition required by its stopping rule (see Section 6.10.1). The solution is either to use different (perhaps standard) initial values or to go over to unconditional estimation.

6.11 Using multiple processes

1. If multiple processors are available, then using multiple processes can speed up the estimation in siena07.

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 a parameter given in the call of siena07()) 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 returns.

3. For estimation by Maximum Likelihood (ML) and by sienaBayes(), parallelization goes by period; for multi-group projects, period × group. Therefore, for this type of

estimation, the maximum meaningful number of parallel processes is (number of waves – 1) × (number of groups).

4. The parameters required to run all processes on one computer are fairly simple: in your call to siena07, set nbrNodes to the number of processes and useCluster and initC to TRUE. The Model Options screen also allows you to specify the number of processes, and will automatically set the other required parameters for you.

5. To use more than one machine is more complicated, but it can be done by using, in addition, the clusterString parameter. 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 siena07 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.

7 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 6.5) 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 6.5). 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 2000 or 4000 iterations in phase 3.

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 sienaAlgorithmCreate() that creates the object with specifications for the algorithm.

7.1 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[11] 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 parameters (when a parameter value was given or was reached where the model is not sensitive to some parameter or combination of parameters). The remedy here usually is to drop some of the effects.

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 6.6.3). 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

[11] Since version 1.1-285.

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.

7.2 Precision of the finite differences method

The implementation of the finite differences method is not scale-invariant[12], 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.

[12]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.

8 Tests

Two types of tests 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 8.1 indicates how to construct multi-parameter tests from the same principle.

2. Score-type tests of single and multiple parameters are described in Section 8.2.

In addition, there are procedures for assessing goodness of fit as explained in Section 5.11.

8.1 Wald-type tests

Wald-type tests 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 6.4 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

$$\frac{\hat{\theta}_k}{\text{s.e.}(\hat{\theta}_k)} = \texttt{ans\$theta[k] / ans\$se[k]} \quad . \tag{4}$$

This can be easily calculated by hand from the RSiena results.

In some cases, however, the t-statistic (4) 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 (4) 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 6.10.2 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 8.2. See the script `RscriptSienaMultiple.R` on the SIENA webpage for an example.

Multi-parameter Wald tests

Now suppose that we wish to test another null hypothesis, which 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. For example, if $p = 5$, for testing

$$H_0 : \; \theta_2 \; = \; \theta_3$$

we would use the matrix

$$A \; = \; (0, 1, -1, 0, 0) \; ;$$

while for testing

$$H_0 : \; \theta_2 \; = \; \theta_3 \; = \; 0$$

if $p = 5$, we would use the matrix

$$A \; = \; \left(\begin{array}{c} 0, 1, 0, 0, 0 \\ 0, 0, 1, 0, 0 \end{array} \right) \; .$$

Then the function Wald.RSiena() can be used to produce the Wald-type test: the chi-squared value of the test statistic, the number of degrees of freedom, and the p-value.

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

```
                              Estimate   Standard   t statistic
                                           Error
Rate parameters:
  0       Rate parameter      10.4625  ( 1.8917   )
  1. eval outdegree (density) -1.8760  ( 0.1883   )  0.0173
  2. eval reciprocity          1.1405  ( 0.3397   ) -0.0430
  3. eval transitive triplets  0.3703  ( 0.0639   ) -0.0425
  4. eval 3-cycles            -0.3373  ( 0.1238   )  0.0020
  5. eval primary              0.6718  ( 0.2125   ) -0.0619
  6. eval sex alter            0.1390  ( 0.2011   ) -0.0545
  7. eval sex ego              0.3165  ( 0.1938   ) -0.0055
  8. eval sex similarity       0.8197  ( 0.2126   ) -0.0298
```

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

```
Covariance matrix of estimates (correlations below diagonal)
    0.035    -0.030    -0.009     0.011    -0.009     0.004    -0.003    -0.018
   -0.470     0.115     0.009    -0.031    -0.003    -0.009     0.002    -0.006
   -0.767     0.432     0.004    -0.006     0.003     0.001     0.000     0.005
    0.475    -0.731    -0.761     0.015    -0.005    -0.001     0.001    -0.005
   -0.218    -0.044     0.246    -0.180     0.045     0.004    -0.003     0.011
    0.096    -0.132     0.067    -0.058     0.090     0.040    -0.017     0.002
   -0.092     0.029    -0.017     0.037    -0.079    -0.437     0.038     0.009
   -0.442    -0.076     0.369    -0.172     0.234     0.054     0.208     0.045
```

To test the null hypothesis that the three sex effects (ego, alter, similarity) are zero, the matrix A is constructed as follows, and the Wald test then is requested.

```
A       <- matrix(0, 3, 8)
A[1, 6] <- 1
A[2, 7] <- 1
A[3, 8] <- 1
Wald.RSiena(A, ans)
```

The result is

```
chisquare        df    pvalue
   16.556     3.000     0.000872
```

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

The Wald test is frequently applied to test the null hypothesis that several parameters are 0. The extra work to define the matrix A above can be automated by using the following function Multipar.RSiena(). The test of the preceding example is then produced by the command

```
Multipar.RSiena(ans, 6, 7, 8)
```

8.1.1 Standard errors of linear combinations

Sometimes there can be interest in a linear combination of parameters. Suppose this is for a sienaFit object called `ans`. The number of parameters is `ans$pp` which is the same as `length(ans$theta)`. This is the number of estimated parameters, excepting (if any) the rate parameters used for conditioning in conditional estimation. Let `a` be the vector with the coefficients of the linear combination. Then

```
sum(a*ans$theta)
```

is the linear combination;

```
t(a) %*% ans$covtheta %*% a
```

is the variance, and

```
sqrt(t(a) %*% ans$covtheta %*% a)
```

is the standard error.

8.2 Score-type tests

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 score test that is implemented for the Method of Moments estimation method in 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 setEffect 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 <- setEffect(myeff, transTies, fix=TRUE, test=TRUE, initialValue=0.0)
```

The score-type test then proceeds by simply estimating the restricted model (not the unrestricted model, with unrestricted parameters) by the standard SIENA estimation algorithm. The result of the score-type test is presented in the summary of the estimation results (the sienaFit object obtained from siena07) and also in the output file. 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 siena07 overrides the initial values in the effects object, so that using siena07 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 usually will be preferable to proceed as follows. First update the initial values using updateTheta, then set the hypothesized value by setEffect, and then carry out the estimation and score-type test by siena07. 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.

```
myeff <- updateTheta(myeff, myans0)
myeff <- setEffect(myeff, transTies, fix=TRUE, test=TRUE, initialValue= 0)
myans1 <- siena07(estimationSettings, data=mydata, effects=myeff)
summary(myans1)
```

8.3 Example: one-sided tests, two-sided tests, and one-step estimates

Suppose that it is desired to test the goodness-of-fit of the model restricted by the null hypothesis that the reciprocity parameter is zero. The following output may be obtained:

```
@2
Generalised score test <c>
--------------------------

Testing the goodness-of-fit of the model restricted by

 (1)   eval:  reciprocity                                =  0.0000
-----------------------------------------------

   c =   3.9982   d.f. = 1   p-value =   0.0455
   one-sided (normal variate):   1.9996
-----------------------------------------------

One-step estimates:

l: constant network rate (period 1)                 6.3840
l: constant network rate (period 2)                 6.4112
eval:  outdegree (density)                          0.9404
eval:  reciprocity                                  1.2567
```

To understand what test statistic `<c>` is about, consider the case where the network is observed at two time points, and let R be the number of reciprocated ties at the second time point. Then it can be shown that the test statistic is some function of

$$\text{Expected } R \text{ under the restricted model } - \text{ observed } R.$$

Thus, the test statistic has some appealing interpretation in terms of goodness-of-fit: when reciprocated ties do have added value for the firms—which means that the reciprocity parameter is not 0, other than the model assumes—then the deviation of the observed R from the R that is expected under the model will be large (large misfit), and so will be the value of the test statistic. Large values of the test statistic imply low p-values, which, in turn, suggests to abandon the model in favor of models incorporating reciprocity.

The null distribution of the test statistic c tends, as the number of observations increases, to the chi-square distribution, with degrees of freedom equal to the number of restricted parameters. The corresponding p-value is given in the output file.

In the present case, one parameter is restricted (reciprocity), hence there is one degree of freedom `d.f. = 1`. The value of the test statistic `c = 3.9982` at one degree of freedom gives `p = 0.0455`. That is, it seems that reciprocity should be included into the model and estimated as the other parameters.

The one-sided test statistic, which can be regarded as normal variate, equals `1.9996` indicating that the value of the transitivity parameter is positive.

The one-step estimates are approximations of the unrestricted estimates (that is, the estimates that would be obtained if the model were estimated once again, but without restricting the reciprocity parameter). The one-step estimate of reciprocity, `1.2567`, hints that this parameter is positive, which agrees with the one-sided test.

8.3.1 Multi-parameter tests

In the case where $K > 1$ model parameters are restricted, SIENA evaluates the test statistic with K degrees of freedom. A low p-value of the joint test would indicate that the goodness-of-fit of the model is intolerable. However, the joint test with K degrees of freedom gives no clue as to what parameters should be included into the model: the poor goodness-of-fit could be due to only one of the K restricted parameters, it could be due to two of the K restricted parameters, or due to all of them. Hence SIENA carries out, in addition to the joint test with K degrees of freedom, additional tests with one degree of freedom that test the single parameters one-by-one. The goodness-of-fit table is as follows:

```
@2
Generalised score test <c>
--------------------------

Testing the goodness-of-fit of the model restricted by

 (1)   eval:  covariate_ij (centered)                    =  0.0000
 (2)   eval:  covariate_i alter                          =  0.0000
 (3)   eval:  covariate_i similarity                     =  0.0000
-------------------------------------------------

Joint test:
-----------
   c =  92.5111   d.f. = 3   p-value < 0.0001

(1) tested separately:
----------------------
 - two-sided:
   c =  62.5964   d.f. = 1   p-value < 0.0001
 - one-sided (normal variate):   7.9118

(2) tested separately:
----------------------
 - two-sided:
   c =  16.3001   d.f. = 1   p-value < 0.0001
 - one-sided (normal variate):   4.0373

(3) tested separately:
----------------------
 - two-sided:
   c =  23.4879   d.f. = 1   p-value < 0.0001
 - one-sided (normal variate):   4.8464
-------------------------------------------------

One-step estimates:

l: constant network rate (period 1)                7.4022
l: constant network rate (period 2)                6.4681
eval:  outdegree (density)                        -0.4439
eval:  reciprocity                                 1.1826
eval:  transitive triplets                         0.1183
```

```
eval:  covariate_ij (centered)                    0.4529
eval:  covariate_i alter                          0.1632
eval:  covariate_i similarity                     0.4147
```

In the example output, three parameters are restricted. The joint test has test statistic c, which has under the null hypothesis a chi-squared distribution with d.f. = 3. The p-value corresponding to the joint test indicates that the restricted model is not tenable. Looking at the separate tests, it seems that the misfit is due to all three parameters. Thus, it is sensible to improve the goodness-of-fit of the baseline model by including all of these parameters, and estimate them.

8.4 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.

8.5 Testing differences between independent groups

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

$$\frac{\hat{\beta}_a - \hat{\beta}_b}{\sqrt{s.e_a^2 + s.e_b^2}} \,, \tag{5}$$

which under the null hypothesis of equal parameters has an approximating standard normal distribution.

8.6 Testing time heterogeneity in parameters

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 $\mathbf{h}$:

$$h_k^{(m)} = \begin{cases} 1 & \{m : w_m \in \mathcal{W}, m \neq 1\} \\ 0 & \text{elsewhere} \end{cases}, \tag{6}$$

where k corresponds to an effect included in the model.[13] 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 (9):

$$f_{ij}^{(m)}(\mathbf{x}) = \sum_k \left(\beta_k + \delta_k^{(m)} h_k^{(m)}\right) s_{ik}\big(\mathbf{x}(i \rightsquigarrow j)\big) \tag{7}$$

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:

$$\begin{aligned} H_0 &: \delta_k^{(m)} = 0 \text{ for all } k, m \\ H_1 &: \delta_k^{(m)} \neq 0 \text{ for some } k, m. \end{aligned} \tag{8}$$

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 <- sienaAlgorithmCreate(nsub=2, n3=100)
mynet1      <- sienaDependent(array(c(s501, s502, s503), dim=c(50, 50, 3)))
mydata      <- sienaDataCreate(mynet1)
myeff       <- getEffects(mydata)
myeff       <- includeEffects(myeff, transTrip, balance)
ans2        <- siena07(myalgorithm, data=mydata, effects=myeff, batch=TRUE)
```

and conduct the timetest through

```
## Conduct the score type test to assess whether heterogeneity is present.
tt2 <- sienaTimeTest(ans2)
plot(tt2, effects=1:2)
```

If as a consequence of this analysis you wish to add time dummy terms, this may be done via

[13] The dummy $\delta_k^{(1)}$ is always zero so that period w_1 is (arbitrarily) considered the reference period.

```
myeff <- includeTimeDummy(myeff, recip, balance, timeDummy="2")
ans3  <- siena07(myalgorithm, data=mydata, effects=myeff, batch=TRUE)
```

and testing again,

```
tt3 <- sienaTimeTest(ans3)
```

and so on.

See Lospinoso (2010) for a walkthrough of the model selection process for time dummy terms.

9 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 siena07 function also used for parameter estimation, but by calling it in such a way that only Phase 3 is carried out (see section 6.5). This is done by requesting `nsub = 0` in the model specification in function sienaAlgorithmCreate. By also requesting `simOnly = TRUE` the calculation of standard errors, which usually is not meaningful when simulating without estimating, is suppressed.

```
sim_model  <-  sienaAlgorithmCreate( projname = "sim_model", cond = FALSE,
                      useStdInits = FALSE, nsub = 0 , simOnly = TRUE)
sim_ans    <-  siena07( sim_model, data = mydata, effects = myeff )
```

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. 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 assigning the desired value to the vector

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

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 4.2.3) can be undesirable. This restriction can be lifted by using `allowOnly = FALSE` in the call of sienaDependent (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 siena07. 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 6.4, 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.

9.1 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 <- siena07( myalgorithm, 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., 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 11.1 (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 number of the wave minus 1 (here 1 because there are supposed to be 2 waves). This type of information can be found out by requesting

```
str(sim_ans$sims[[568]])
```

Section 4.1.2 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
                     }
# 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 sienaGOF.

9.2 Conditional and unconditional simulation

The distinction between conditional and unconditional simulation is the same for the simulation as for the estimation option of SIENA, described in Section 6.10.1. The choice between conditional and unconditional simulation is made in the sienaAlgorithmCreate 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 sienaDataCreate. 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 6.10.1), 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 Getting started

The best way to get started is to download the R scripts from the SIENA website and start reading and playing with them.

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 sienaDataCreate, getEffects, and sienaAlgorithmCreate, as indicated in Chapters 4 and 6. At first, leave the whole model specification as it is by default (see Section 5): a constant rate function, the out-degree effect, and the reciprocity effect.

Then let the program estimate the parameters, using function siena07. 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 6.

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 6.1.2. 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 transitive ties effect and the 3-cycles effect (see, e.g., Section 5); you can find their shortName, needed to specify them, in Chapter 12, 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.

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 5.

10.1 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 5; the effects are defined mathematically in Chapter 12.

Approaches to model specification are presented in Chapter 5 and in Snijders et al. (2010b).

When the distribution of the out-degrees is fitted poorly (which can be inspected using the sienaGOF function of Section 5.11), 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.

11 Multilevel network analysis

For combining SIENA 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 'sub-projects' 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 4.3.1. The special effort to be made here is the construction of the data files for the large (combined) network.

2. Combining different sub-projects into one *multi-group* project, and analyzing this by siena07. The 'sub-projects' are the same as the 'different networks' mentioned here. This is explained in Section 11.1.
   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, 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 sub-project, and other effects.

   Usually, option (2) is preferable to option (1).

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 siena08. This is explained in Section 11.2.

4. Combining different sub-projects into one *multi-group* project as in option (2), but analyzing this by sienaBayes. This is explained in Section 11.3.
   Here the assumption for the parameters is that all basic rate parameters may differ arbitrarily between the sub-projects; for the other parameters, some are identical and others vary randomly across sub-projects according to a multivariate normal distribution. The distinction between "some" and "others" here is made by the parameter `random` in function setEffect().

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 SIENA 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. The fourth option is a middle ground between the first two and the third. Therefore 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.

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 sub-project 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 sienaTimeTest to test for which of the effects especially there is a large variation in parameter values across the sub-projects; 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 4.2.3. 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 sienaDependent.

11.1 Multi-group Siena analysis

The multi-group option 'glues' several projects (further referred to as *sub-projects*) after each other into one larger multi-group project. These sub-projects 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 sub-projects. Also their names must be the same. The number of actors and the number of waves can be different, however. These sub-projects then are combined into one project where the number of actors is the largest of the number of actors of the sub-projects, and the number of observations is the sum of the observations of the sub-projects. This is done by the function sienaGroupCreate which creates a so-called `sienaGroupEffects` object, which is a list of `sienaEffect` objects with some additional information.

As an example, suppose that three projects 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 project has 35 actors and 11 observations. The step from observation 2 to 3 switches from sub-project `sub1` to sub-project `sub2`, while the step from observation 6 to 7 switches from sub-project `sub2` to `sub3`. These switching steps do not correspond to simulations of the actor-based model, because that would not be meaningful.

The different sub-projects 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 print01Report, and checking that the tendencies in the dependent variable or variables, upward/stable/downward, are not too different between the sub-projects. The sienaTimetest 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 sub-projects 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 projects, individual covariates are centered by subtracting the overall mean (across all groups), but dyadic covariates are centered by subtracting the within-group means.

11.2 Meta-analysis of Siena results

The function siena08 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 siena08 takes as input the sienaFit objects produced by separate runs of siena08. 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 sienaAlgorithmCreate; 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 4.2.3) 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 siena08, by using `allowOnly = FALSE` in the call of sienaDependent, as mentioned in Section 4.2.3.

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 pa-

rameters 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), siena08() 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 siena08(), 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 SIENA 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.

11.2.1 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

$$\mu_\theta = \mathcal{E}\,\theta_j \ ,$$
$$\sigma_\theta^2 = \operatorname{var}\theta_j \ .$$

Each of these parameters must have been estimated in a run of siena07, 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 siena08 estimates the mean μ_θ and the variance σ_θ^2 of the distribution of θ, and tests several hypotheses concerning these 'meta-parameters':

1. Test $H_0^{(0)}$: $\mu_\theta = \sigma_\theta^2 = 0$ (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 siena08 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 6.10.2.) 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`.

11.2.2 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[14] 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 right 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;

[14] This correlation is defined for the population of networks, and if the population does not exist then also the correlation is not defined.

(*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;

(*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.

11.2.3 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.

11.3 Random coefficient multilevel Siena analysis

The function sienaBayes is for Bayesian estimation of one group or of multiple groups all having the same number of waves and the same model specification. 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. The difference is made by setting the parameter `random` in the function setEffect. 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. 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, a flat prior is assumed.

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 steps for groupwise parameters are obtained from, 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, followed by one subphase of the Robbins-Monro algorithm for Method of Moments estimates for the groups separately, with step size initgainGroupwise. The proposal covariance matrices then are scaled, 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 (nrunMHBatches of a number of MH steps sampling chains, 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)) are performed. In the warming as well as the final phase, the number of MH steps is determined by parameter mult ("multiplication factor") in the call of sienaAlgorithmCreate that created the algorithm object.

The function sienaBayes 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`.

11.3.1 Which data sets to use for sienaBayes

sienaBayes uses as data set a sienaGroup object with 2 or more groups. The number of waves should be the same for all groups.

sienaBayes may run into an error (the program will hang) if there are any actors who are inactive at the first wave, as indicated by all structural zeros.

sienaBayes 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 siena07() with the (default) Method of Moments, followed by a meta-analysis by means of siena08(), may be preferable.

It is advisable to first do a multi-group analysis of the same model, followed by a sienaTimeTest, to get an initial understanding of where problems might occur. You may then later use the result for the `prevAns` parameter of sienaBayes.

11.3.2 Model specification

The extra part of model specification, compared to siena07(), is that it is required to specify which parameters are randomly varying from group to group, and which are fixed across groups. The basic rate parameters always are randomly varying. Other rate parameters can not yet be included.

The specification of fixed vs. randomly varying is done in the function setEffect.

There currently is little advice about this. 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 sienaTimeTest, 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 fixed 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.)

amount of information: which specification will use the information in the data most efficiently?
(Here finally we do have an answer. Assuming that an effect is fixed 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 11.3.8. 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 fixed.)

11.3.3 How to enter your data in sienaBayes

See the example at the bottom of the sienaBayes 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 sienaDependent; 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.

11.3.4 How to choose the parameter settings for sienaBayes

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 print the 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=100, 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 sienaAlgorithmCreate. 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).

11.3.5 Prior distributions

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 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 `setEffect`. The list and order of the randomly varying effects, except for the rate parameters, is shown by requesting

```
myeff[myeff$randomEffects & myeff$include, ]
```

if the effects object is called `myeff`.

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 sienaBayes, 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`.

Rate parameters

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 try and include some groups with high estimated rate parameters, the 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 plausible values for the first rate parameter would be centered about 4 and for the second about 5, in both cases with a standard deviation of 1.4, corresponding to a variance of about 2. One could use the following piece of code.

```
m7 <- c(4, 5, 0, 0, 0, 0, 0)
S7 <- matrix(0, 7, 7)
diag(S7) <- c(2, 2, 1, 1, 1, 1, 1)
ans <- sienaBayes(...., priorMu=m7, priorSigma=S7, priorDf=20,
                  priorRatesFromData = 0, ....)
```

`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. By choosing it, as suggested here, equal to the number of groups, the prior will have a non-negligible but also not a very strong influence.

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. Since many SIENA parameters are defined in such a way that they have values in the range between –1 and +1, a prior variance in the range from 0.02 to 0.5 would often be reasonable; but it will be good to consider earlier studies to have a good grounding for this choice.

Continuing the example above, if the 5 varying non-rate parameters would start with an outdegree and a reciprocity parameter, the prior means and prior variances could be modified, e.g., to

```
m7 <- c(4, 5, -2, +1.5, 0, 0, 0)
S7 <- matrix(0, 7, 7)
diag(S7) <- c(2, 2, 1, 0.3, 0.2, 0.1, 0.1)
```

11.3.6 Operation of sienaBayes()

In Section 11.3.4 it was already suggested to start with a run with very low values of the sample size settings for the MCMC procedure.

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 glueBayes(). This is illustrated in the example on the help page.

During operation of sienaBayes(), 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 sienaBayesFit objects, and therefore can be used in the print and summary functions; they also can be used for the `prevBayes` option to continue estimation.

11.3.7 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 sienaBayes 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 print and summary functions for sienaBayesFit 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 sienaAlgorithmCreate().

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 4 or 5 such sequences. Function extract.sienaBayes() 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 stan or with the help of package coda.

Currently there is no good way in sienaBayes() 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 starting point and several continuations using `prevBayes`. In the latter option one uses a long chain instead of parallel chains, which is too bad, but this is the best we have currently on offer; and using monitor() on such parts of a long chain still will give an acceptable impression of convergence.

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.

11.3.8 Interpreting results of sienaBayes

The print and 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.

The functions simpleBayesTest() and multipleBayesTest() 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 sienaBayesFit object, as mentioned in the help page, may be useful.

It should be noted that the `.out` files produced by sienaBayes() are produced somewhere in the initial phase of the project and not meant to be informative for final results. They may be disregarded.

When comparing results for specifications that differ with respect to specifying the effects as fixed 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 fixed. 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.

12 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 similarity effects of actor covariates. There is one additional effect for two-mode networks, viz., the four-cycle effect.

Some of the effects contain a number which is denoted in this section by c, 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.) These are set or modified by the `setEffect` function, e.g.,

```
myeffects <- setEffect(myeffects, gwespFF, parameter=69)
```

12.1 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 5.1.

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).)

12.1.1 Network evaluation function

The network evaluation function for actor i is defined as

$$f_i^{\text{net}}(x) = \sum_k \beta_k^{\text{net}} s_{ik}^{\text{net}}(x) \tag{9}$$

where β_k^{net} are parameters and $s_{ik}^{\text{net}}(x)$ are effects as defined below. If the model also contains some elementary effects (see Section 5.1.1), the objective function is the sum of this and

$$f_i^{\text{el}}(x) = \sum_k \beta_k^{\text{el}} s_{ik}^{\text{el}}(x) \ , \tag{10}$$

see Section 5.1.3. Elementary effects are of the type $s_{ijk}^{\text{el}}(x) = x_{ij}\, s_{ijk}^{\text{el0}}(x)$, where $s_{ijk}^{\text{el0}}(x)$ does not depend on x_{ij}.

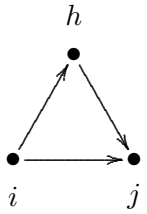
The potential effects in the network evaluation function are the following. Note that in all effects where a constants c occurs, this constant can be chosen and changed by the user; this is the internal effect parameter mentioned above, which can be modified by the function `setEffect(..., parameter=..., ...)`. 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.

Structural effects

Structural effects are the effects depending on the network only. The following list also contains some elementary effects (see Section 5.1.1). 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.

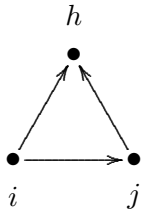
1. *out-degree effect* or *density effect* (`density`), defined by the out-degree
   $s_{i1}^{\text{net}}(x) = x_{i+} = \sum_j x_{ij}$,
   where $x_{ij} = 1$ indicates presence of a tie from i to j while $x_{ij} = 0$ indicates absence of this tie;

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 j is tied to h),


for directed networks, $s_{i3}^{\text{net}}(x) = \sum_{j,h} x_{ij}\, x_{ih}\, x_{hj}$;
and for non-directed networks, $s_{i3}^{\text{net}}(x) = \sum_{j<h} x_{ij}\, x_{ih}\, x_{hj}$;
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 j is tied to i and i to h, while also j is tied to h), which is different from the transitive triplets effect only for directed networks,
$s_{i6}^{\text{net}}(x) = \sum_{j,h} x_{ji}\, x_{ih}\, x_{jh}$;
this cannot be used together with the transitive triplets effect in Method of Moments estimation, because of perfect collinearity of the fit statistics;

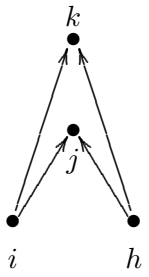
7. *transitive reciprocated triplets effect* (`transRecTrip`), which can be regarded as an interaction between the transitive triplets effect and reciprocity, where the reciprocated tie is the tie $i \leftrightarrow j$ that closes the two-path $i \to h \to j$,
$s_{i7}^{\text{net}}(x) = \sum_{j,h} x_{ij}\, x_{ji}\, x_{ih}\, x_{hj}$;

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 $\{i \to h \to j, i \to j\}$,
$s_{i8}^{\text{net}}(x) = \sum_{j,h} x_{ij}\, x_{ih}\, x_{hj}\, x_{jh}$;
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`,
$s_{i10}^{\text{net}}(x) = \sum_{j,k,h;\text{all different}} x_{ij}\, x_{hj}\, x_{ik}\, x_{hk}$;
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)` for the two-mode and `(cycle4ND)` for the non-directed case,
$s_{i11}^{\text{net}}(x) = \sum_{j,k,h;\text{all different}} x_{ij}\, x_{ik}\, x_{hj}\, x_{hk}$;
for parameter $p = 2$ the square root is taken;
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)`,
$s_{i13}^{\text{net}}(x) = \sum_{j,h} x_{hi}\, x_{ij}\, (1 - x_{hj})$;

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 SIENA 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,

$$s_{i15}^{\text{net}}(x) = \sum x_{ij} d_{ij} \tag{11a}$$

with

$$d_{ij} = \sum_{\substack{h=1 \\ h \neq i,j}}^{n} \left(b_0 - \mid x_{hi} - x_{hj} \mid \right) . \tag{11b}$$

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.

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,
$s_{i19}^{\text{net}}(x) = \#\{j \mid x_{ij} = 0, \sum_h (x_{ih}\, x_{hj}) \geq 2\}$;
endowment effect only likelihood-based;

20. *number of dense triads* (`denseTriads`), defined as triads containing at least c ties,
$s_{i20}^{\text{net}}(x) = \sum_{j,h} x_{ij}\, I\{x_{ij} + x_{ji} + x_{ih} + x_{hi} + x_{jh} + x_{hj}) \geq c\}$,
where the 'indicator function' $I\{A\}$ is 1 if the condition A is fulfilled and 0 otherwise, and where c 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 `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

$$\text{GWESPFF}(i,\alpha) \;=\; \sum_{k=1}^{n-2} e^{\alpha} \left\{1 \,-\, \left(1 - e^{-\alpha}\right)^k\right\} \text{EPFF}_{ik} \;, \tag{12a}$$

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 $i \to h \to j$.

An equivalent way of writing this is

$$\text{GWESPFF}(i,\alpha) \;=\; \sum_{j=1}^{n} x_{ij}\, e^{\alpha} \left\{1 \,-\, \left(1 - e^{-\alpha}\right)^{\sum_{h=1}^{n} x_{ih} x_{hj}}\right\} \;, \tag{12b}$$

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

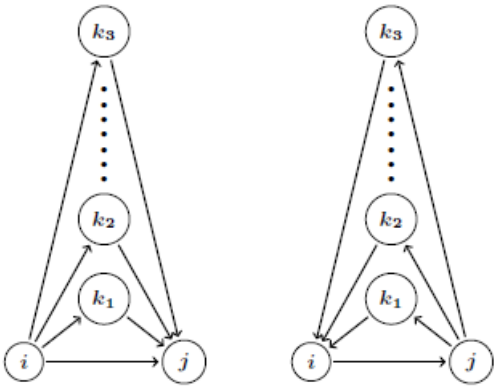


Figure. Geometrically weighted edgewise shared partners;
left, transitive ('gwespFF'); right, cyclical ('gwespBB'); for F='forward', B='backward'.

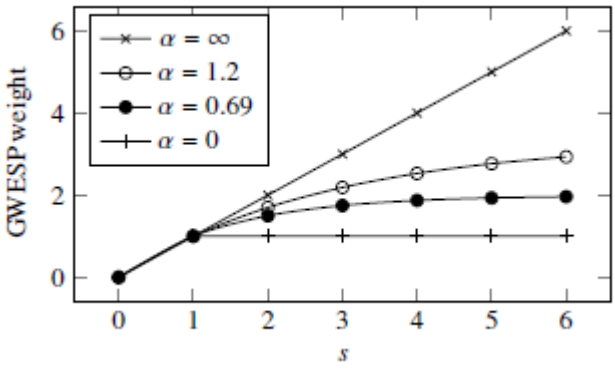


Figure. GWESP weight for a tie closing s two-paths for $\alpha = \infty, 1.2, 0.69, 0$.

$k \geq 1$, and for $\alpha \to \infty$ they tend to k. Since we can write

$$\sum_{j,h} x_{ih} x_{hj} x_{ij} \; = \; \sum_{k=1}^{n-2} k \, \mathrm{EP}_{ik} \; ,$$

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 <- setEffect(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 (12b) are considered, and not of the tie $i \to h$.

Thus, the GWESP effects are defined in RSiena as the elementary effects

$$s_{i21}^{\mathrm{el}}(x) \;=\; x_{ij}\, e^{\alpha} \left\{ 1 \;-\; \left(1 - e^{-\alpha}\right)^{\sum_{h=1}^{n} x_{ih} x_{hj}} \right\} \;. \tag{13}$$

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 $i \leftarrow h \to j$;
`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 $i \to h \leftarrow j$;
`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 $i \rightleftarrows h \rightleftarrows j$.

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
$c_1 + c_2 \times \mathrm{GWESPFF}(i, \alpha')$
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. *number of (unilateral) peripheral relations to dense triads*,
$s_{i22}^{\mathrm{net}}(x) = \sum_{j,h,k} x_{ij}(1-x_{ji})(1-x_{hi})(1-x_{ki}) I\{(x_{jh}+x_{hj}+x_{jk}+x_{kj}+x_{hk}+x_{kh}) \geq c\}$,
where c is the same constant as in the *dense triads* effect;
for symmetric networks, the 'unilateral' condition is dropped, and the definition is
$s_{i22}^{\mathrm{net}}(x) = \sum_{j,h,k} x_{ij}(1 - x_{hi})(1 - x_{ki}) I\{(x_{jh} + x_{hj} + x_{jk} + x_{kj} + x_{hk} + x_{kh}) \geq c\}$;

23. *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,
$s_{i23}^{\mathrm{net}}(x) = \sum_j x_{ij}\, x_{+j} = \sum_j x_{ij} \sum_h x_{hj} = \sum_j x_{ij} (\sum_{h \neq i} x_{hj} + 1)$;
in SIENA 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;

24. *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,
$s^{\text{net}}_{i24}(x) = \sum_j x_{ij}\,\sqrt{x_{+j}} = \sum_j x_{ij}\sqrt{\sum_h x_{hj}}$;
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;

25. *out-degree related popularity effect* (`outPop`) (earlier called *activity* or *activity of alter effect*), defined by the sum of the out-degrees of the others to whom i is tied,
$s^{\text{net}}_{i25}(x) = \sum_j x_{ij}\,x_{j+} = \sum_j x_{ij}\sum_h x_{jh}$;
until version 3.313, this effect was multiplied by a factor $1/n$;

26. *out-degree related popularity (sqrt) effect* (`outPopSqrt`) (earlier called *activity of alter (sqrt measure) effect*), defined by the sum of the square roots of the out-degrees of the others to whom i is tied,
$s^{\text{net}}_{i26}(x) = \sum_j x_{ij}\,\sqrt{x_{j+}} = \sum_j x_{ij}\sqrt{\sum_h x_{jh}}$;
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;

27. *reciprocal degree-related popularity effect* (`reciPop`) defined by the sum of the reciprocal degrees of the others to whom i is tied,
$s^{\text{net}}_{i27}(x) = \sum_j x_{ij}\,x^{(r)}_j$,
where the reciprocal degree is defined by
$x^{(r)}_j = \sum_h x_{jh}x_{hj}$.

28. *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,
$s^{\text{net}}_{i28}(x) = \sum_j x_{ij}\,\sqrt{x^{(r)}_j}$,
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;

29. *in-degree related activity effect*, (`inAct`) defined as the cross-product of the actor's in- and out-degrees,
$s^{\text{net}}_{i29}(x) = x_{i+}\,x_{+i}$;
endowment effect only likelihood-based;

30. *in-degree related activity (sqrt) effect*, (`inActSqrt`) defined by
$s^{\text{net}}_{i30}(x) = x_{i+}\,\sqrt{x_{+i}}$;

31. *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, $s_{i31}^{\text{net}}(x,z) = I\{x_{+i}=0\} \sum_j x_{ij}$;

32. *out-degree related activity effect* (`outAct`), defined as the squared out-degree of the actor, $s_{i32}^{\text{net}}(x) = x_{i+}^2$;
endowment effect only likelihood-based;

33. *out-degree related activity (sqrt) effect* (`outActSqrt`) (earlier called *out-degree^(1.5)*), defined by
$s_{i33}^{\text{net}}(x) = x_{i+}^{1.5} = x_{i+}\sqrt{x_{i+}}$
endowment effect only likelihood-based;

34. *reciprocal degree-related activity effect* (`reciAct`) defined by the degree of i multiplied by i's reciprocal degree,
$s_{i34}^{\text{net}}(x) = x_{i+}\, x_i^{(r)}$,
where the reciprocal degree is defined as above;

35. *out-degree up to c* (*truncated out-degree*), where c is some constant (internal effect parameter, see above), there are two implementations here: `outTrunc` and `outTrunc2`, to enable the simultaneous use of this effect with 2 different internal effect parameters; the effect is defined by
$s_{i35}^{\text{net}}(x) = \min(x_{i+}\,,\, c)$;
note that for $c = 1$ 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 $c = 1$) is interpreted as a positive tendency $+1.2$ toward outdegrees equal to 0; the standard error is unchanged. In the case of $c = 1$, an alternative and more directly comprehensible name is *outdegree at least 1* effect.

36. *square root out-degree*, defined by
$s_{i36}^{\text{net}}(x) = \sqrt{x_{i+}}$;
this is left out in later versions of SIENA;

37. *squared (out-degree – c)*, where c is some constant, defined by
$s_{i37}^{\text{net}}(x) = (x_{i+} - c)^2$,
where c is chosen to diminish the collinearity between this and the density effect;
this is left out in later versions of SIENA;

38. *sum of (1/(out-degree + c)* (`outInv`), where c is some constant, defined by
$s_{i38}^{\text{net}}(x) = 1/(x_{i+} + c)$;
endowment effect only likelihood-based;

39. *sum of (1/(out-degree + c)(out-degree + c + 1))* (`outSqInv`), where c is some constant, defined by

$s^{\text{net}}_{i39}(x) = 1/(x_{i+} + c)(x_{i+} + c + 1)$;
endowment effect only likelihood-based.

⊙ the following assortativity effects are, for $c = 1$, various orientations of three-paths (plus a bit of two-paths):

40. *out-out degree^(1/c) 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,
$s^{\text{net}}_{i40}(x) = \sum_j x_{ij}\, x^{1/c}_{i+} x^{1/c}_{j+}$;
c can be 1 or 2 (the latter value is the default);

41. *out-in degree^(1/c) assortativity* (`outInAss`), which represents the differential tendency for actors with high out-degrees to be tied to other actors who have high in-degrees,
$s^{\text{net}}_{i41}(x) = \sum_j x_{ij}\, x^{1/c}_{i+} x^{1/c}_{+j}$;
c can be 1 or 2 (the latter value is the default);

42. *in-out degree^(1/c) assortativity* (`inOutAss`), which represents the differential tendency for actors with high in-degrees to be tied to other actors who have high out-degrees,
$s^{\text{net}}_{i42}(x) = \sum_j x_{ij}\, x^{1/c}_{+i} x^{1/c}_{j+}$;
c can be 1 or 2 (the latter value is the default);

43. *in-in degree^(1/c) 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,
$s^{\text{net}}_{i43}(x) = \sum_j x_{ij}\, x^{1/c}_{+i} x^{1/c}_{+j}$;
c can be 1 or 2 (the latter value is the default);

44. *network-isolate effect*, (`isolateNet`), the effect of ego having in-degree as well as out-degree zero, i.e., being a total isolate,
$s^{\text{net}}_{i44}(x, z) = I\{x_{+i} = x_{i+} = 0\}$;

45. *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,
$s^{\text{net}}_{i45}(x) = \sum_j I\{x_{+j} \geq 1, x_{j+} = 0\}$;

46. *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:
$s^{\text{net}}_{i46}(x) = \sum_j I\{x_{+j} \geq 1\}$;

47. *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:
$s^{\text{net}}_{i47}(x) = \sum_j I\{x_{+j} \geq 2\}$;

48. *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:
$s^{\text{net}}_{i48}(x) = \sum_j I\{x_{+j} \geq 3\}$;

49. *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),
$s^{\text{net}}_{i49}(x) = \sum_j x_{ij}\, I\{x_{+j} = 1, x_{j+} = 0\}$.
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).

50. *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:

$$x^{\text{simm}}_{ij} = \begin{cases} 1 & x_{ij} = x_{ji} = 1 \text{ and} \\ & \quad \text{there is at least one } h \text{ for which } x_{hi} = x_{ih} = x_{hj} = x_{jh} = 1 \\ 0 & \text{else.} \end{cases}$$

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 Simmelian outdegree of i is
$s^{\text{net}}_{i50}(x) = \sum_j x^{\text{simm}}_{ij}$.

Dyadic covariate effects

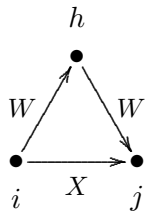
The effects for a dyadic covariate w_{ij} are

51. *covariate (centered) main effect* (`X`),
$s^{\text{net}}_{i51}(x) = \sum_j x_{ij}\,(w_{ij} - \bar{w})$
where $\bar{w}$ is the mean value of w_{ij} ;

52. *covariate (centered) × reciprocity* `(XRecip)`,
$s^{\text{net}}_{i52}(x) = \sum_j x_{ij}x_{ji}\,(w_{ij} - \bar{w})$.

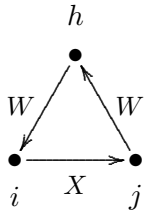
⊙ Various different ways can be modeled in which a triadic combination can be made between the dyadic covariate and the network. In the explanation, the dyadic covariate is regarded as a weighted network (which will be reduced to a non-weighted network if w_{ij} only assumes the values 0 and 1). By way of exception, the dyadic covariate is not centered in these three 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.

53. *WW => X closure of covariate* `(WWX)`,
$s^{\text{net}}_{i53}(x) = \sum_{j\neq h} x_{ij}\, w_{ih}\, w_{hj}$;
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.

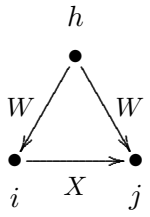


Since the dyadic covariates are represented by square arrays and not by edgelists, this and the following effects will be relatively time-consuming if the number of nodes is large.

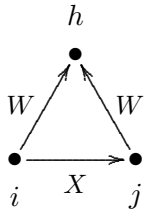
54. *mixed cyclic WW => X closure*, (X: cyclic closure of W) `(cyWWX)`
$s^{\text{net}}_{i54}(x) = \sum_{j\neq h} x_{ij}\, w_{jh}\, w_{hi}$;
this refers to the cyclic closure of $W - W$ two-paths (weighted if the dyadic covariate W does not only have 0 and 1 values); the contribution of the tie $i \overset{X}{\to} j$ is proportional to the number of product of weights of $W - W$ two-paths $j \overset{W_{jh}}{\longrightarrow} h \overset{W_{hi}}{\longrightarrow} i$;



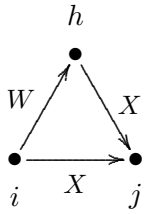
55. *incoming shared WWX*, (X: incoming shared W) `(InWWX)`
$s^{\text{net}}_{i55}(x) = \sum_{j\neq h} x_{ij}\, w_{hi}\, w_{hj}$;
this refers to shared incoming W ties contributing to the tie $i \overset{X}{\to} j$;



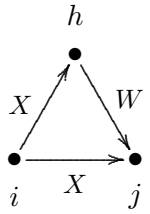
56. *incoming shared* WWX, (X: incoming shared W) `(OutWWX)`
$s^{\text{net}}_{i56}(x) = \sum_{j \neq h} x_{ij}\, w_{ih}\, w_{jh}$;
this refers to shared outgoing W ties contributing to the tie $i \overset{X}{\to} j$;



57. WX => X *closure of covariate* `(WXX)`,
$s^{\text{net}}_{i57}(x) = \sum_{j \neq h} x_{ij}\, w_{ih}\, x_{hj}$;
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;



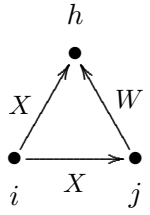
58. XW => X *closure of covariate* `(XWX)`,
$s^{\text{net}}_{i58}(x) = \sum_{j \neq h} x_{ij}\, x_{ih}\, w_{hj}$;
this refers to the closure of mixed $X-W$ two-paths; each $X-W$ two-path $i \to h \overset{W}{\to} j$ is weighted by w_{hj} and the sum of these weights measures the strength of the tendency toward closure of these mixed $X - W$ twopaths by a tie.



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.

59. 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 is the dependent variable here. The effect is defined as
$s^{\text{el}}_{i59}(x) = x_{ij} \sum_{h;h \neq j} x_{ih}\, w_{hj}$;

60. 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 is the dependent variable. The effect is defined as
$s^{\text{el}}_{i60}(x) = x_{ij} \sum_{h;h \neq j} x_{ih}\, w_{jh}$.



Monadic covariate effects

For actor-dependent covariates v_j (recall that these are centered internally by SIENA) 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:

61. *covariate-alter* or *covariate-related popularity* (`altX`), defined by the sum of the covariate over all actors to whom i has a tie,
$s_{i61}^{\text{net}}(x) = \sum_j x_{ij}\, v_j$;

62. *covariate squared - alter* or *squared covariate-related popularity* (`altSqX`), defined by the sum of the squared centered covariate over all actors to whom i has a tie, (not included if the variable has range less than 2)
$s_{i62}^{\text{net}}(x) = \sum_j x_{ij}\, v_j^2$;

63. *covariate-ego* or *covariate-related activity* (`egoX`), defined by i's out-degree weighted by his covariate value,
$s_{i63}^{\text{net}}(x) = v_i\, x_{i+}$;

64. *covariate squared - ego* or *squared covariate-related activity* (`egoSqX`), defined by i's out-degree weighted by his covariate value,
$s_{i64}^{\text{net}}(x) = v_i^2\, x_{i+}$;

65. *covariate-related similarity* (`simX`), defined by the sum of centered similarity scores sim_{ij}^v between i and the other actors j to whom he is tied,
$s_{i65}^{\text{net}}(x) = \sum_j x_{ij}(\text{sim}_{ij}^v - \widehat{\text{sim}^v})$,
where $\widehat{\text{sim}^v}$ is the mean of all similarity scores, which are defined as $\text{sim}_{ij}^v = \frac{\Delta - |v_i - v_j|}{\Delta}$ 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")`);

66. *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,
$s_{i66}^{\text{net}}(x) = \sum_j x_{ij}\, (v_j - v_i)$;

67. *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,
$s_{i67}^{\text{net}}(x) = \sum_j x_{ij}\, (v_j - v_i)^2$;

68. *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,
$s_{i68}^{\text{net}}(x) = \sum_j x_{ij}\, v_i\, (v_j - v_i)$;

69. *covariate-related similarity* $\times$ *reciprocity* (`simRecipX`), defined by the sum of centered similarity scores for all reciprocal dyads in which i is situated,
$s_{i69}^{\text{net}}(x) = \sum_j x_{ij} x_{ji} (\text{sim}_{ij}^v - \widehat{\text{sim}^v})$;

70. *covariate-related similarity* × *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,
$s_{i70}^{\text{net}}(x) = \sum_j x_{ij}\, x_{ih}\, x_{hj}(\text{sim}_{ij}^{v} - \widehat{\text{sim}^{v}})$;

71. *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,
$s_{i71}^{\text{net}}(x) = \sum_j x_{ij}\, I\{v_i = v_j\}$,
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;

72. *same covariate* × *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,
$s_{i72}^{\text{net}}(x) = \sum_j x_{ij} x_{ji}\, I\{v_i = v_j\}$;

73. *same covariate* × *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,
$s_{i73}^{\text{net}}(x) = \sum_j x_{ij}\, x_{ih}\, x_{hj}\, I\{v_i = v_j\}$;

74. *different covariate* × *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,
$s_{i74}^{\text{net}}(x) = \sum_j x_{ij}\, x_{ih}\, x_{hj}\, I\{v_i \neq v_j\}$;

75. *indegree popularity from same covariate* (`sameXInPop`) defined by the number of incoming ties received by those to whom i is tied and sent by others who have the same covariate value as i,
$s_{i75}^{\text{net}}(x) = \sum_j x_{ij} \sum_h x_{hj}\, I\{v_i = v_h\}$;

76. *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,
$s_{i76}^{\text{net}}(x) = \sum_j x_{ij} \sum_h x_{hj}\, I\{v_i \neq v_h\}$;

77. *outdegree activity to same covariate* (`sameXOutAct`) defined by the squared number of ties to those others who have the same covariate value as i,
$s_{i77}^{\text{net}}(x) = \big(\sum_j x_{ij}\, I\{v_i = v_j\}\big)^2$;

78. *outdegree activity to different covariate* (`diffXOutAct`) defined by the squared number of ties to those others who have a different covariate value than i,
$s_{i78}^{\text{net}}(x) = \big(\sum_j x_{ij}\, I\{v_i \neq v_j\}\big)^2$;

79. *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),
$s_{i79}^{\text{net}}(x) = \sum_j x_{ij} \sum_h x_{ih}\, I\{v_j = v_h\}$;

80. *outdegree activity weighted by alter's covariate* (`altXOutAct`) defined by the squared sum of ties weighted by alter's covariate values,
$s^{\text{net}}_{i80}(x) = \big(\sum_j x_{ij}\, v_j\big)^2$;
since the covariate here is used as a weight, this probably makes sense especially for non-centered covariates;

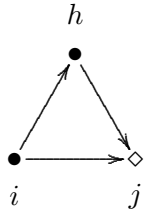
81. *Simmelian alter X effect*, (`simmelianAltX`), the effect of Simmelian ties weighted by alter's value of X. The Simmelian transformation of the network is the network composed of all ties embedded in at least one complete triad:

$$x^{\text{simm}}_{ij} = \begin{cases} 1 & x_{ij} = x_{ji} = 1 \text{ and} \\ & \quad \text{there is at least one } h \text{ for which } x_{hi} = x_{ih} = x_{hj} = x_{jh} = 1 \\ 0 & \text{else.} \end{cases}$$

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 Simmelian alter V effect is
$s^{\text{net}}_{i81}(x) = \sum_j x^{\text{simm}}_{ij}\, v_j$.

82. *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,
$s^{\text{net}}_{i82}(x) = \sum_j x_{ij}\, x_{ih}\, x_{hj}\, I\{v_i = v_j = v_h\}$;

83. *transitive triplets jumping to different V* (`jumpXTransTrip`),
$s^{\text{net}}_{i83}(x) = \sum_{j \neq h} x_{ij}\, x_{ih}\, x_{hj}\, I\{v_i = v_h \neq v_j\}$;
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;



84. *covariate-ego* × *alter* (`egoXaltX`), defined by the product of i's covariate and the sum of those of his alters,
$s^{\text{net}}_{i84}(x) = v_i \sum_j x_{ij}\, v_j$;

85. *covariate-ego* × *alter* × *reciprocity* (`egoXaltXRecip`), defined by the product of i's covariate and the sum of those of his reciprocated alters,
$s^{\text{net}}_{i85}(x) = v_i \sum_j x_{ij}\, x_{ji}\, v_j$;

86. *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,
$s^{\text{net}}_{i86}(x) = \sum_j x_{ij}\, \text{dsign}(v_i - v_j)$,
where $\text{dsign}(d) = 0$ for $d < 0$, 0.5 for $d = 0$, and 1 for $d > 0$.

87. *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),
$s^{\text{net}}_{i87}(x) = \sum_j (1 - x_{ij})\big(\max_h x_{ih} x_{hj}\big) v_j$.

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

$$\breve{v}_j = \begin{cases} \dfrac{\sum_h x_{jh} v_h}{x_{j+}} & \text{if } x_{j+} > 0 \\ 0 & \text{if } x_{j+} = 0. \end{cases} \tag{14}$$

(Since v is centered, the value of 0 in case $x_{j+} = 0$ is also the mean value of the original variable.)
(It may be noted that 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.

88. *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,

$$s^{\text{net}}_{i88}(x) = \sum_j x_{ij} \breve{v}_j \qquad \text{(parameter 1)}$$

For parameter 2, it is defined similarly, but for an alters' covariate-average excluding ego:

$$s^{\text{net}}_{i88}(x) = \sum_j x_{ij} \breve{v}^{(-i)}_j \qquad \text{(parameter 2)}$$

where

$$\breve{v}^{(-i)}_j = \begin{cases} \dfrac{\sum_{h \neq i} x_{jh} v_h}{x_{j+} - x_{ji}} & \text{if } x_{j+} - x_{ji} > 0 \\ 0 & \text{if } x_{j+} - x_{ji} = 0. \end{cases} \tag{15}$$

To compute the contribution for this effect, note that

$$\sum_j x_{ij} \breve{v}^{(-i)}_j = \sum_j x_{ij} \, \frac{x_{j+} \breve{v}_j - x_{ji} v_i}{x_{j+} - x_{ji}}$$

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).

89. *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

$$s_{i89}^{\text{net}}(x) = \sum_j x_{ij}\, x_{j+} \breve{v}_j \qquad \text{(parameter 1)}$$

and for parameter 2, as

$$s_{i89}^{\text{net}}(x) = \sum_j x_{ij} (x_{j+} - x_{ji}) \breve{v}_j^{(-i)} \qquad \text{(parameter 2)}$$

where $\breve{v}_j$ and $\breve{v}_j^{(-i)}$ are as above.

90. *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,

$$s_{i90}^{\text{net}}(x) = \sum_j x_{ij} \left(\text{sim}(\grave{v})_{ij} - \widehat{\text{sim}^v} \right) ,$$

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")`.

91. *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,

$$s_{i91}^{\text{net}}(x) = \sum_j x_{ij} \left(\text{sim}(\breve{v})_{ij} - \widehat{\text{sim}^v} \right) ,$$

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.

92. *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:

$$s_{i92}^{\text{net}}(x) = \sum_j x_{ij} \check{v}_j^{(-i)}$$

where

$$\check{v}_j^{(-i)} = \begin{cases} \dfrac{\sum_{h \neq i} x_{hj}\, v_h}{x_{+j} - x_{ij}} & \text{if } x_{+j} - x_{ij} > 0 \\ 0 & \text{if } x_{+j} - x_{ij} = 0. \end{cases} \tag{16}$$

93. *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:

$$s_{i93}^{\text{net}}(x) = \sum_j x_{ij}(x_{+j} - x_{ij}) \check{v}_j^{(-i)} \ .$$

94. *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,

$$s_{i94}^{\text{net}}(x) = \sum_j x_{ij} \left(\text{sim}(\acute{v})_{ij} - \widehat{\text{sim}^v} \right) ,$$

where the similarity scores $\text{sim}(\acute{v})_{ij}$ are defined as

$$\text{sim}(\acute{v})_{ij} = \frac{\Delta - |v_i - \check{v}_j^{(-i)}|}{\Delta} \ ,$$

while $\check{v}_j^{(-i)}$ 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 alters!), so that this effect is about a comparison between i and the out-neighbourhoods of the j's.

12.1.2 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).

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.

In the SIENA output for projects 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 SIENA 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, SIENA 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 $i \overset{X}{\to} j$;

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 six 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 $p = 2$ its square root) times i's X-out-degree:
$s^{\text{net}}_{i5}(x) = \sum_j x_{ij}\,(w_{+i} - \bar{w}) = x_{i+}\,(w_{+i} - \bar{w})$ or
for $p = 2$ $s^{\text{net}}_{i5}(x) = \sum_j x_{ij}\,(\sqrt{w_{+i}} - \sqrt{\bar{w}}) = x_{i+}\,(\sqrt{w_{+i}} - \sqrt{\bar{w}})$;

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 W-out-degrees of i (for $p = 2$ its square root) 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 $p = 2$ $s^{\text{net}}_{i7}(x) = \sum_j x_{ij}\,(\sqrt{w_{i+}} - \sqrt{\bar{w}}) = x_{i+}\,(\sqrt{w_{i+}} - \sqrt{\bar{w}})$;

8. *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:
$s^{\text{net}}_{i8}(x) = \sum_j x_{ij}\,(w_{+i} - \bar{w})\,(w_{+j} - \bar{w})$ or
for $p = 2$ $s^{\text{net}}_{i8}(x) = \sum_j x_{ij}\,(\sqrt{w_{+i}} - \sqrt{\bar{w}})\,(\sqrt{w_{+j}} - \sqrt{\bar{w}})$;
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.

9. *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:
$s^{\text{net}}_{i9}(x) = \big(\sum_j x_{ij}\,w_{ij}\big)^2$ or
for $p = 2$ $s^{\text{net}}_{i9}(x) = \sum_j x_{ij}\,w_{ij}\sqrt{\sum_h x_{ih}\,w_{ih}}$;

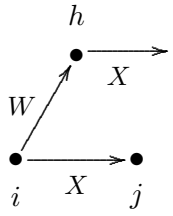
10. *Duplex XW in-degree popularity effect* (X: duplex W indegree$^{1/p}$ popularity)
`(doubleInPop)`
which is like the in-degree popularity effect, but now for the degrees in the joined X and W network; for parameter $p = 2$ the square root of the out-degrees is taken:
$s^{\text{net}}_{i10}(x) = \sum_j x_{ij}\,w_{ij}\sum_h x_{hj}\,w_{hj}$ or

for $p = 2$ $s^{\text{net}}_{i10}(x) = \sum_j x_{ij}\, w_{ij} \sqrt{\sum_h x_{hj}\, w_{hj}}$.

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.

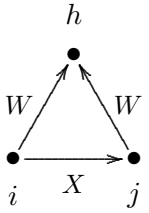
11. *Effect of W-betweenness on X-popularity* (X: betweenness$^{1/p}$ W popularity) `(betweenPop)` defined by the sum of the W-betweenness counts of the others to whom i is tied:
$s^{\text{net}}_{i11}(x) = \sum_j x_{ij} \left(\sum_{h,k;h\neq k} w_{hj}\, w_{jk}\, (1 - w_{hk}) \right)^{1/p}$;

12. mixed twopath activity effect (X: twopath$^{1/p}$ W activity) `(outOutActIntn)` defined by the outdegree of i multiplied by sum of X-outdegrees of those to whom i has an X-tie:
for $p = 1$, this is
$s^{\text{net}}_{i12}(x) = x_{i+} \sum_h w_{ih} \big(x_{h+} - \bar{x} \big)$
and for $p = 2$, there is a square root transformation:
$s^{\text{net}}_{i12}(x) = x_{i+},\ \sum_h w_{ih} \big(\sqrt{x_{h+}} - \sqrt{\bar{x}} \big)$
where $\bar{x}$ is the average degree over all waves.

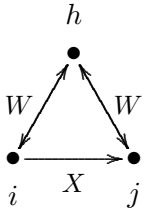


Further there are a number of mixed triadic effects.

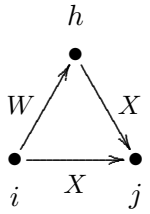
13. *agreement along W leading to X*, (X: from W agreement) `(from)`
$s^{\text{net}}_{i13}(x) = \sum_{j\neq h} x_{ij}\, w_{ih}\, w_{jh}$;
this refers to agreement of actors with respect to their W-choices (structural equivalence with respect to outgoing W-choices); the contribution of the tie $i \overset{X}{\rightarrow} j$ is proportional to the number of joint W choices of others, $i \overset{W}{\rightarrow} h \overset{W}{\leftarrow} j$.



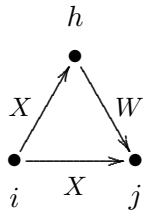
14. *agreement along mutual W-ties leading to X*, (X: from W mutual agreement) `(fromMutual)`
$s^{\text{net}}_{i14}(x) = \sum_{j\neq h} x_{ij}\, w_{ih}\, w_{hi}\, w_{jh}\, w_{hj}$;
this refers to agreement of actors with respect to their mutual W-choices (structural equivalence with respect to mutual W-choices); the contribution of the tie $i \overset{X}{\rightarrow} j$ is proportional to the number of joint mutual W choices of others, $i \overset{W}{\leftrightarrow} h \overset{W}{\leftrightarrow} j$.



15. *W leading to agreement along X*, (*X*: *W* to agreement) `(to)`
$s^{\text{net}}_{i15}(x) = \sum_{j \neq h} x_{ij}\, w_{ih}\, x_{hj}$;
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.



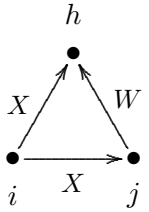
16. *XWX closure of W*, `(cl.XWX)`
$s^{\text{net}}_{i16}(x) = \sum_{j \neq h} x_{ij}\, x_{ih}\, w_{hj}$;
this refers to the closure of mixed $X - W$ two-paths; the contribution of the tie $i \overset{X}{\to} j$ is proportional to the number of mixed $X - W$ two-paths $i \overset{X}{\to} h \overset{W}{\to} j$ plus the number of mixed $X - W$ two-in-stars $i \overset{X}{\to} h$, $j \overset{W}{\to} h$.
The interpretation is the closure of $X \to W$ paths: if there is a tie $h \overset{W}{\to} j$, then ties $i \overset{X}{\to} j$ and $i \overset{X}{\to} h$ will tend to entrain each other.
(The reported target statistic is multiplied by 2.)



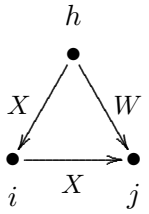
There are two partial variants of this effect; they can be distinguished not by the Method of Moments, but only by Maximum Likelihood and Bayesian estimation.

17. *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 $i \overset{X}{\to} h \overset{W}{\to} j$.
So the dependent variable here is only the tie variable $i \to j$ in the figure above. The effect is defined as
$s^{\text{el}}_{i17}(x) = x_{ij} \sum_{h;h \neq j} x_{ih}\, w_{hj}$;

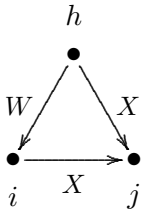
18. *XWX closure-2 of W*, `(cl.XWX2)`
This is an elementary effect, not an evaluation effect, comprising of the 'XWX closure of W' effect only the contribution of the number of mixed $X - W$ two-in-stars $i \overset{X}{\to} h$, $j \overset{W}{\to} h$.
In other words, only the $i \to j$ tie in the figure here is the dependent variable. The effect is defined as
$s^{\text{el}}_{i18}(x) = x_{ij} \sum_{h;h \neq j} x_{ih}\, w_{jh}$.



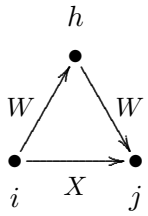
19. *shared mixed incoming X and W* , (X: mixed incoming with W) `(mixedInXW)`
$s^{\text{net}}_{i19}(x) = \sum_{j \neq h} x_{ij}\, x_{hi}\, w_{hj}$;
this refers to the closure of mixed incoming X and W ties; the contribution of the tie $i \xrightarrow{X} j$ is proportional to the number of mixed $X - W$ out-two-stars $h \xrightarrow{X} i, h \xrightarrow{W} j$.
This is an elementary effect for actor i: only the x_{ij} tie indicator in the formula, corresponding to the tie $i \xrightarrow{X} j$, is the dependent variable here.



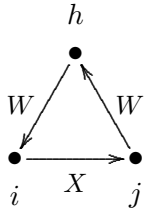
20. *shared mixed incoming W and X*, (X: mixed incoming from W) `(mixedInWX)`
$s^{\text{net}}_{i20}(x) = \sum_{j \neq h} x_{ij}\, w_{hi}\, x_{hj}$;
this refers to the closure of mixed incoming W and X ties; the contribution of the tie $i \xrightarrow{X} j$ is proportional to the number of mixed $W - X$ out-two-stars $h \xrightarrow{W} i, h \xrightarrow{X} j$.
This is an elementary effect for actor i: only the x_{ij} tie indicator in the formula, corresponding to the tie $i \xrightarrow{X} j$, is the dependent variable here.



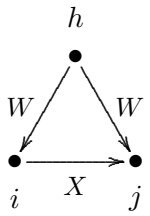
21. *mixed WW => X closure*, (X: closure of W) `(closure)`
$s^{\text{net}}_{i21}(x) = \sum_{j \neq h} x_{ij}\, w_{ih}\, w_{hj}$;
this refers to the closure of $W - W$ two-paths; the contribution of the tie $i \xrightarrow{X} j$ is proportional to the number of $W - W$ two-paths $i \xrightarrow{W} h \xrightarrow{W} j$.
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';



22. *mixed cyclic WW => X closure*, (X: cyclic closure of W) `(cyClosure)`
$s^{\text{net}}_{i22}(x) = \sum_{j \neq h} x_{ij}\, w_{jh}\, w_{hi}$;
this refers to the cyclic closure of $W - W$ two-paths; the contribution of the tie $i \xrightarrow{X} j$ is proportional to the number of $W - W$ two-paths $j \xrightarrow{W} h \xrightarrow{W} i$.
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'.



23. *closure of shared incoming WW => X*,
(X: shared incoming W) `(sharedIn)`
$s^{\text{net}}_{i23}(x) = \sum_{j \neq h} x_{ij}\, w_{hi}\, w_{hj}$;
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.

24. *Jaccard similarity with respect to outgoing W ties effect* `(JoutMix)`, the Jaccard similarity with respect to outgoing W-ties,
$s^{\text{net}}_{i24}(x) = \sum_j x_{ij}\, J_{W,\,\text{out}}(i,j)$, where

$$J_{W,\,\text{out}}(i,j) \;=\; \frac{\sum_h w_{ih}\, x_{jh}}{w_{i+} + w_{j+} - \sum_h w_{ih}\, w_{jh}}$$

(where 0/0 is taken as 0).

25. *Jaccard similarity with respect to incoming W ties effect* `(JinMix)`,
defined by the Jaccard similarity with respect to incoming W-ties,
$s^{\text{net}}_{i25}(x) = \sum_j x_{ij}\, J_{W,\,\text{in}}(i,j)$, 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).

26. 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

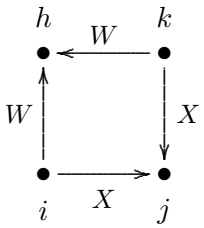
worthwhile to try out different values of α to see which one gives the best fit. For large α the `gwespFFMix` effect will approximate the `closure` effect, and similarly for the other correspondences mentioned above.

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`).

27. *shared W leading to agreement along X*, (X: shared W to agreement) `(sharedTo)`
$s_{i27}^{\text{net}}(x) = \sum_{j,h,k;k\neq i} x_{ij}\, x_{kj}\, \big(w_{ih}\, w_{kh} - c\big)$;
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 $i \overset{W}{\rightarrow} h \overset{W}{\leftarrow} k \overset{X}{\rightarrow} j$.
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 $w_{ih}\, w_{kh}$:

$$c = \begin{cases} \big(1/Mn(n-1)\big) \sum_{m,h} \big(w_{+h}^2(t_m) - w_{+h}(t_m)\big) & (p \geq 3) \\ 0 & (p < 3)\ . \end{cases}$$

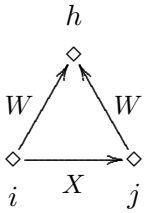


Note that since this is the evaluation function for actor i with respect to network X, only the x_{ij} tie indicator in the formula, corresponding to the tie $i \overset{X}{\rightarrow} j$, is the dependent variable here. The interpretation is that actors have the tendency to make the same outgoing X-choices as those (k) with whom they share many outgoing W-ties.
For $p = 2$ and $p = 4$, the square root is taken:
$s_{i27}^{\text{net}}(x) = \sum_{j,k;k\neq i} x_{ij}\, x_{kj} \left(\sqrt{\sum_h w_{ih}\, w_{kh}} - \sqrt{c}\right)$.

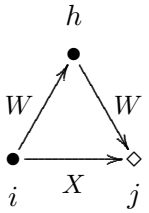
This can be regarded as a higher-order effect related to indegree-popularity. The differences between the centered and non-centered versions amount to a multiple of the indegree-popularity (without $\sqrt{}$) `inPop` effect. Therefore, when the `sharedTo` effect is used, it is advisable also to include the `inPop` effect, so as to avoid misinterpretations.

Then there are some mixed triadic effects restricted to triples with the same or different values on a monadic covariate V.

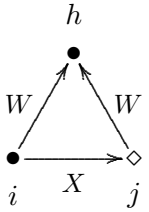
28. *agreement along W leading to X, for same V*,
(X: from W agr. $\times$ same V) (`covNetNet`),
$s^{\text{net}}_{i28}(x) = \sum_{j \neq h} x_{ij}\, w_{ih}\, w_{jh}\, I\{v_i = v_j\}$;
(specified with `interaction1` $= W$, `interaction2` $= V$)
this refers to agreement of actors with respect to their 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 $v_i = v_j$.



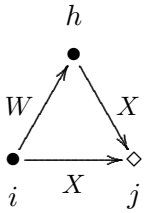
29. *mixed $WW \Rightarrow X$ closure, same-V path jumping to different V*
(X: W closure jumping V), (`jumpWWClosure`)
(specified with `interaction1` $= W$, `interaction2` $= V$)
$s^{\text{net}}_{i29}(x) = \sum_{j \neq h} x_{ij}\, w_{ih}\, w_{hj}\, I\{v_i = v_h \neq v_j\}$;
this refers to the closure 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.



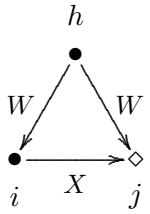
30. *agreement along W, same-V, leading to X for different V* (X: from W agreement jumping V), (`jumpFrom`)
(specified with `interaction1` $= W$, `interaction2` $= V$)
$s^{\text{net}}_{i30}(x) = \sum_{j \neq h} x_{ij}\, w_{ih}\, w_{jh}\, I\{v_i = v_h \neq v_j\}$;
this refers to agreement about W ties, 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.



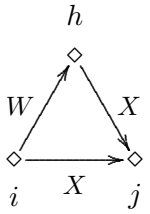
31. *mixed $WX \Rightarrow X$ closure, same-V path jumping to different V*
(X: mixed W closure jumping V), (`jumpWXClosure`)
(specified with `interaction1` $= W$, `interaction2` $= V$)
$s^{\text{net}}_{i31}(x) = \sum_{j \neq h} x_{ij}\, w_{ih}\, x_{hj}\, I\{v_i = v_h \neq v_j\}$;
this refers to the closure 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.



32. *closure of shared incoming WW => X same-V, leading to X for different V*,
(X: shared incoming W jumping V) (`jumpSharedIn`)
$s^{\text{net}}_{i32}(x) = \sum_{j \neq h} x_{ij}\, w_{hi}\, w_{hj}\, I\{v_i = v_h \neq v_j\}$;
(specified with `interaction1` $= W$, `interaction2` $= V$)
this refers to shared incoming W ties contributing to the tie $i \overset{X}{\to} j$, 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.



33. *mixed WX ⇒ X closure, homogeneous on V* (X: mixed W closure homog. V), (`homWXClosure`)
(specified with `interaction1` $= W$, `interaction2` $= V$)
$s^{\text{net}}_{i33}(x) = \sum_{j \neq h} x_{ij}\, w_{ih}\, x_{hj}\, I\{v_i = v_j = v_h\}$;
this refers to the closure of $W - X$ paths, restricted to triples that are homogeneous with respect to V.



There are two effects similar to the effects described above depending on the auxiliary variable $\breve{v}_i$, "alters' v-average". Now the 'alter' is defined, however, by the other network W. Thus, *W-alters' v-average* $\breve{v}^W_i$ is defined by

$$\breve{v}^W_i = \begin{cases} \dfrac{\sum_j w_{ij} v_j}{w_{i+}} & \text{if } w_{i+} > 0 \\ 0 & \text{if } w_{i+} = 0. \end{cases} \tag{17}$$

34. *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,

$$s^{\text{net}}_{i34}(x) = \sum_j x_{ij} \breve{v}^W_j \qquad \text{(parameter 1).}$$

For parameter 2, it is defined similarly, but for an alters' covariate-average excluding ego:

$$s^{\text{net}}_{i34}(x) = \sum_j x_{ij} \breve{v}^{W(-i)}_j \qquad \text{(parameter 2)}$$

where

$$\breve{v}^{W(-i)}_j = \begin{cases} \dfrac{\sum_{h \neq j} w_{jh} v_h}{w_{j+} - w_{ji}} & \text{if } w_{j+} - w_{ji} > 0 \\ 0 & \text{if } w_{j+} - w_{ji} = 0. \end{cases} \tag{18}$$

35. *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

$$s_{i35}^{\text{net}}(x) = \sum_j x_{ij}\, x_{j+}\, \breve{v}_j^W \qquad \text{(parameter 1)}.$$

For parameter 2, it is defined similarly, but for an alters' covariate-total excluding ego:

$$s_{i35}^{\text{net}}(x) = \sum_j x_{ij}\, (x_{j+} - x_{ji})\, \breve{v}_j^{W(-i)} \qquad \text{(parameter 2)}$$

where $\breve{v}_j$ and $\breve{v}_j^{W(-i)}$ are as above.

36. *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,

$$s_{i36}^{\text{net}}(x) = \sum_j x_{ij} \left(\text{sim}(\grave{v}^W)_{ij} - \widehat{\text{sim}^v} \right) ,$$

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, $\breve{v}_j^{(-i)}$ 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")`.

37. *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,

$$s_{i37}^{\text{net}}(x) = \sum_j x_{ij} \left(\text{sim}(\breve{v}^W)_{ij} - \widehat{\text{sim}^v} \right) ,$$

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.

38. *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,

$$s_{i38}^{\text{net}}(x) = \sum_j x_{ij} \left(\text{sim}(\acute{v}^W)_{ij} - \widehat{\text{sim}^v} \right) ,$$

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.

The following is an effect for three networks: the dependent network is X, the two explanatory networks are W and Z.

39. *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,

$$s_{i39}^{\text{net}}(x) = \sum_j x_{ij} \left(\sum_h w_{ih}\, w_{jh}\, z_{+h} \right) .$$

For internal effect parameter $p = 2$, the effect is

$$s_{i39}^{\text{net}}(x) = \sum_j x_{ij} \left(\sum_h w_{ih}\, w_{jh}\, \sqrt{z_{+h}} \right) .$$

W is given as `interaction1`, Z as `interaction2`.

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 5.1.1).

12.1.3 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

$$c_i^{\text{net}}(x) = \sum_k \zeta_k^{\text{net}}\, s_{ik}^{\text{net}}(x) \tag{19}$$

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

$$e^{\mathrm{net}}(x) \;=\; \sum_k \gamma_k^{\mathrm{net}}\, s_{ik}^{\mathrm{net}}(x) \tag{20}$$

for dissolution of ties. In this formula, the γ_k^{net} are the parameters for the endowment function. The potential effects $s_{ik}^{\mathrm{net}}(x)$ 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

$$f_i^{\mathrm{net}}(x) \;+\; c_i^{\mathrm{net}}(x)\;, \tag{21}$$

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

$$f_i^{\mathrm{net}}(x) \;+\; e_i^{\mathrm{net}}(x)\;, \tag{22}$$

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.

12.1.4 Network rate function

The network rate function λ^{net} (lambda) is defined for Model Type 1 (which is the default Model Type) as a product

$$\lambda_i^{\mathrm{net}}(\rho, \alpha, x, m) = \lambda_{i1}^{\mathrm{net}} \lambda_{i2}^{\mathrm{net}} \lambda_{i3}^{\mathrm{net}}$$

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 $m = 1, ..., M-1$. If there are only $M = 2$ observations, the basic rate parameter is called ρ^{net}.

2. The effect of actor covariates (`RateX`) with values v_{hi} can be represented by the factor

   $$\lambda_{i2}^{\text{net}} = \exp(\sum_h \alpha_h \, v_{hi}) \, .$$

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 out-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

     $$\exp\left(\alpha_h/(x_{i+}+1)\right) \, .$$

   - logarithmic outdegree effect (`outRateLog`)
     This effect multiplies the factor $\lambda_{i3}^{\text{net}}$ by

     $$\exp\Big(\ln\left(\alpha_h(x_{i+}+1)\right)\Big) \; = \; \left(x_{i+}+1\right)^{\alpha_h} \, .$$

     This effect works properly only for non-conditional estimation (set `cond = FALSE` *in* `sienaAlgorithmCreate()`*).*

     The exponential link function and logarithmic transformation collaborate to produce direct proportionality to (outdegree + 1), in case the parameter is $\alpha_h = 1$.

For the two latter effects, the addition of 1 to the outdegrees avoids problems (division by 0, logarithm of 0) that otherwise would occur when $x_{i+} = 0$.

When rate effects are included, sometimes it can be helpful to specify a smaller value for the initial gain parameter in sienaAlgorithmCreate(); e.g., `initg=0.01` or even `initg=0.001`. When the estimation algorithm diverges and any rate effects are in the model, this option should be considered.

12.2 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.

12.2.1 Behavioral evaluation function

The behavior evaluation function for actor i is defined as

$$f_i^{\mathrm{beh}}(x, z) = \sum_k \beta_k^{\mathrm{beh}} s_{ik}^{\mathrm{beh}}(x, z) \tag{23}$$

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, which is given in the output file 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:
   $s_{i2}^{\mathrm{beh}}(x, z) = z_i^2$.

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 includeEffects, it is necessary[15] to specify the dependent behavior variable in the keyword name as well as the network in the keyword interaction1. For example,

[15] If this behavior variable is the only dependent variable, then this is not necessary. But this seldom happens.

```
myCoEvolutionEff <- includeEffects( myCoEvolutionEff, name = "drinkingbeh",
                                    avSim, indeg, outdeg,
                                    interaction1 = "friendship" )
```

The list of these effects is the following.

3. *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,
   $s_{i3}^{\mathrm{beh}}(x,z) = x_{i+}^{-1} \sum_j x_{ij}(\mathrm{sim}_{ij}^z - \widehat{\mathrm{sim}^z})$;
   (and 0 if $x_{i+} = 0$) ;

4. *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,
   $s_{i4}^{\mathrm{beh}}(x,z) = \sum_j x_{ij}(\mathrm{sim}_{ij}^z - \widehat{\mathrm{sim}^z})$;

5. *indegree effect* (`indeg`),
   $s_{i5}^{\mathrm{beh}}(x,z) = z_i \sum_j x_{ji}$;

6. *outdegree effect* (`outdeg`),
   $s_{i6}^{\mathrm{beh}}(x,z) = z_i \sum_j x_{ij}$;

7. *isolate effect* (`isolate`), the differential attractiveness of the behavior for isolates,
   $s_{i7}^{\mathrm{beh}}(x,z) = z_i I\{x_{+i} = 0\}$,
   where again $I\{A\}$ denotes the indicator function of the condition A;

8. *average similarity* $\times$ *reciprocity effect* (`avSimRecip`), defined by the sum of centered similarity scores sim_{ij}^z between i and the other actors j to whom he is reciprocally tied,
   $s_{i8}^{\mathrm{beh}}(x,z) = x_{i(r)}^{-1} \sum_j x_{ij}x_{ji}(\mathrm{sim}_{ij}^z - \widehat{\mathrm{sim}^z})$;
   (and 0 if $x_{i(r)} = 0$) ;

9. *total similarity* $\times$ *reciprocity effect* (`totSimRecip`), defined by the sum of centered similarity scores sim_{ij}^z between i and the other actors j to whom he is reciprocally tied,
   $s_{i9}^{\mathrm{beh}}(x,z) = \sum_j x_{ij}x_{ji}(\mathrm{sim}_{ij}^z - \widehat{\mathrm{sim}^z})$;

10. *average similarity* $\times$ *popularity alter effect* (`avSimPopAlt`), defined by the sum of centered similarity scores sim_{ij}^z between i and the other actors j to whom he is tied, multiplied by their indegrees,
    $s_{i10}^{\mathrm{beh}}(x,z) = x_{i+}^{-1} \sum_j x_{ij}x_{+j}(\mathrm{sim}_{ij}^z - \widehat{\mathrm{sim}^z})$;
    (and 0 if $x_{i+} = 0$) ;

11. *popularity alter effect* (`popAlt`), defined by the average in-degrees of the other actors j to whom i is tied,
    $s_{i11}^{\mathrm{beh}}(x,z) = z_i\, x_{i+}^{-1} \sum_j x_{ij}x_{+j}$;
    (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;

12. *total similarity* × *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,
$s^{\text{beh}}_{i12}(x,z) = \sum_j x_{ij}x_{+j}(\text{sim}^z_{ij} - \widehat{\text{sim}^z})$;

13. *average similarity* × *reciprocity* × *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,
$s^{\text{beh}}_{i13}(x,z) = x^{-1}_{i(r)} \sum_j x_{ij}x_{ji}x_{+j}(\text{sim}^z_{ij} - \widehat{\text{sim}^z})$;
(and 0 if $x_{i(r)} = 0$) ;

14. *total similarity* × *reciprocity* × *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,
$s^{\text{beh}}_{i14}(x,z) = \sum_j x_{ij}x_{ji}x_{+j}(\text{sim}^z_{ij} - \widehat{\text{sim}^z})$;

15. *average alter effect* (`avAlt`), defined by i's behavior multiplied by the average behavior of his alters (a kind of ego-alter behavior covariance),
$s^{\text{beh}}_{i15}(x,z) = z_i\big(\sum_j x_{ij}\, z_j\big)/\big(\sum_j x_{ij}\big)$
(and the mean behavior, i.e. 0, if the ratio is 0/0) ;

16. *total alter effect* (`totAlt`), defined by i's behavior multiplied by the sum of behavior of his alters,
$s^{\text{beh}}_{i16}(x,z) = z_i\big(\sum_j x_{ij}\, z_j\big)$;

17. *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),
$s^{\text{beh}}_{i17}(x,z) = z_i\big(\sum_j x_{ji}\, z_j\big)/\big(\sum_j x_{ji}\big)$
(and the mean behavior, i.e. 0, if the ratio is 0/0) ;

18. *total alter effect* (`totInAlt`), defined by i's behavior multiplied by the sum of behavior of his in-alters,
$s^{\text{beh}}_{i18}(x,z) = z_i\big(\sum_j x_{ji}\, z_j\big)$;

19. *average reciprocated alter effect* (`avRecAlt`), defined by i's behavior multiplied by the average behavior of his reciprocated alters,
$s^{\text{beh}}_{i19}(x,z) = z_i\big(\sum_j x_{ij}\, x_{ji}\, z_j\big)/\big(\sum_j x_{ij}\, x_{ji}\big)$
(and 0 if the ratio is 0/0) ;

20. *total reciprocated alter effect* (`totRecAlt`), defined by i's behavior multiplied by the sum of behavior of his reciprocated alters,
$s^{\text{beh}}_{i20}(x,z) = z_i\big(\sum_j x_{ij}\, x_{ji}\, z_j\big)$;

21. *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:

$$x_{ij}^{\text{simm}} = \begin{cases} 1 & x_{ij} = x_{ji} = 1 \text{ and} \\ & \quad \text{there is at least one } h \text{ for which } x_{hi} = x_{ih} = x_{hj} = x_{jh} = 1 \\ 0 & \text{else.} \end{cases}$$

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 average Simmelian alter effect is
$s_{i21}^{\text{beh}}(x, z) = z_i\big(\sum_j x_{ij}^{\text{simm}}\, z_j\big)/\big(\sum_j x_{ij}^{\text{simm}}\big)$
(and the mean behavior, i.e. 0, if the ratio is 0/0) ;

22. *total Simmelian alter effect*, (`totSimmelianAlt`), the effect of the sum of behavior for the Simmelian alters, defined by
$s_{i22}^{\text{beh}}(x, z) = z_i\big(\sum_j x_{ij}^{\text{simm}}\, z_j\big)$;

23. *average alter effect at distance 2* (`avAltDist2`), defined by i's behavior multiplied by the average of the alter-averages $\breve{z}_j^{(-i)}$ of his alters, excluding the contribution from a tie $j \to i$ (if any), defined by

$$\breve{z}_j^{(-i)} = \begin{cases} \dfrac{\sum_{h \neq i} x_{jh}\, z_h}{x_{j+} - x_{ji}} & \text{if } x_{j+} - x_{ji} > 0 \\ 0 & \text{if } x_{j+} - x_{ji} = 0. \end{cases} \tag{24}$$

(also see (15)),
$s_{i23}^{\text{beh}}(x, z) = z_i\big(\sum_j x_{ij}\, \breve{z}_j^{(-i)}\big)/\big(\sum_j x_{ij}\big)$
(and the mean behavior, i.e. 0, if the ratio is 0/0) ;

24. *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),
$s_{i24}^{\text{beh}}(x, z) = z_i \sum_j x_{ij} \sum_{h \neq i} x_{jh}\, z_h = z_i \sum_j x_{ij}\, (x_{j+} - x_{ji})\, \breve{z}_j^{(-i)}$;

25. *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),
$s_{i25}^{\text{beh}}(x, z) = z_i\big(\sum_j x_{ij}\, (x_{j+} - x_{ji})\, \breve{z}_j^{(-i)}\big)/\big(\sum_j x_{ij}\big) = z_i\big(\sum_j x_{ij} \sum_{h \neq i} x_{jh}\, z_h\big)/\big(\sum_j x_{ij}\big)$
(and the mean behavior, i.e. 0, if the ratio is 0/0) ;

26. *total average alter effect at distance 2* (`totAAltDist2`), defined by i's behavior multiplied by the total of the alter-averages $z_j^{(-i)}$ of his alters, excluding the contribution from a tie $j \to i$ (if any),
$s_{i26}^{\text{beh}}(x, z) = z_i\big(\sum_j x_{ij}\, \breve{z}_j^{(-i)}\big)$;

27. *average incoming alter effect at distance 2* (`avInAltDist2`), defined by i's behavior multiplied by the average of the incoming alter averages $\check{z}_j^{(-i)}$ of his alters, excluding the contribution from a tie $i \to j$ (if any), defined by

$$\check{z}_j^{(-i)} = \begin{cases} \dfrac{\sum_{h \neq i} x_{hj}\, z_h}{x_{+j} - x_{ij}} & \text{if } x_{+j} - x_{ij} > 0 \\ 0 & \text{if } x_{+j} - x_{ij} = 0. \end{cases} \tag{25}$$

(cf. (24)),
$s_{i27}^{\text{beh}}(x, z) = z_i\big(\sum_j x_{ij}\, \check{z}_j^{(-i)}\big)/\big(\sum_j x_{ij}\big)$
(and the mean behavior, i.e. 0, if the ratio is 0/0) ;

28. *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),
$s_{i28}^{\text{beh}}(x, z) = z_i \sum_j x_{ij} \sum_{h \neq i} x_{hj}\, z_h = z_i \sum_j x_{ij}\, (x_{+j} - x_{ij})\, \check{z}_j^{(-i)}$;

29. *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),
$s_{i29}^{\text{beh}}(x, z) = z_i\big(\sum_j x_{ij}\, (x_{+j} - x_{ij})\, \check{z}_j^{(-i)}\big)/\big(\sum_j x_{ij}\big)$
(and the mean behavior, i.e. 0, if the ratio is 0/0) ;

30. *total average incoming alter effect at distance 2* (`totAInAltDist2`), defined by i's behavior multiplied by the total of the incoming alter-averages $z_j^{(-i)}$ of his alters, excluding the contribution from a tie $i \to j$ (if any),
$s_{i30}^{\text{beh}}(x, z) = z_i\big(\sum_j x_{ij}\, \check{z}_j^{(-i)}\big)$;

31. *maximum alter effect* (`maxAlt`), defined by i's behavior multiplied by the maximum behavior of his alters,
$s_{i31}^{\text{beh}}(x, z) = z_i\big(\max_j x_{ij}\, z_j\big)$
(and the mean behavior, i.e. 0, if $\sum_j x_{ij} = 0$) ;

32. *minimum alter effect* (`minAlt`), defined by i's behavior multiplied by the minimum behavior of his alters,
$s_{i32}^{\text{beh}}(x, z) = z_i\big(\min_j x_{ij}\, z_j\big)$
(and the mean behavior, i.e. 0, if $\sum_j x_{ij} = 0$) ;

33. *dense triads effect* (`behDenseTriads`), defined by i's behavior multiplied by the number of dense triads in which actor i is located,
$s_{i33}^{\text{beh}}(x, z) = z_i \sum_{j,h} I\{x_{ij} + x_{ji} + x_{ih} + x_{hi} + x_{jh} + x_{hj}) \geq c\}$,
where c is either 5 or 6;

34. *peripheral effect*, defined by i's behavior multiplied by the number of dense triads to which actor i stands in a unilateral-peripheral relation,

$s^{\text{beh}}_{i34}(x,z) =$
$z_i \sum_{j,h,k} x_{ij}(1-x_{ji})(1-x_{hi})(1-x_{ki}) I\{x_{ij}+x_{ji}+x_{ih}+x_{hi}+x_{jh}+x_{hj}\,) \geq c\}$,
where c is the same constant as in the *dense triads* effect;
for symmetric networks, the unilateral condition is dropped, and the effect is
$s^{\text{beh}}_{i34}(x,z) = z_i \sum_{j,h,k} x_{ij}(1-x_{hi})(1-x_{ki}) I\{x_{ij}+x_{ji}+x_{ih}+x_{hi}+x_{jh}+x_{hj}\,) \geq c\}$;

35. *reciprocated degree effect* (`recipDeg`),
$s^{\text{beh}}_{i35}(x,z) = z_i \sum_j x_{ij}\, x_{ji}$;

36. *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,
$s^{\text{beh}}_{i36}(x,z) = x_{+i}\, x_{i+}^{-1} \sum_j x_{ij}(\text{sim}^z_{ij} - \widehat{\text{sim}^z})$;
(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.

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 `interaction1`, X_2 as `interaction2`.

For the combined degree-effects, mentioned first, `F` means 'Forward', `B` means 'Backward', and `R` means 'Reciprocal'.

37. *double outdegree effect* (`FFDeg`),
$s^{\text{beh}}_{i37}(x,z) = z_i \sum_j x_{1ij}\, x_{2ij}$;

38. *double indegree effect* (`BBDeg`),
$s^{\text{beh}}_{i38}(x,z) = z_i \sum_j x_{1ji}\, x_{2ji}$;

39. *combined out-indegree effect* (`FBDeg`),
$s^{\text{beh}}_{i39}(x,z) = z_i \sum_j x_{1ij}\, x_{2ji}$;

40. *combined out-reciprocated degree effect* (`FRDeg`),
$s^{\text{beh}}_{i40}(x,z) = z_i \sum_j x_{1ij}\, x_{2ij}\, x_{2ji}$;

41. *combined in-reciprocated degree effect* (`BRDeg`),
$s^{\text{beh}}_{i41}(x,z) = z_i \sum_j x_{1ji}\, x_{2ij}\, x_{2ji}$.

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.

42. *main covariate effect* `(effFrom)`,
$s^{\text{beh}}_{i42}(x, z) = z_i v_i$;
here too, the other dependent behavioral variables are centered so that they have overall mean 0;

43. *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 (26),
$s^{\text{beh}}_{i43}(x, z) = z_i \, \breve{v}_i$.
This is similar to the 'average alter' effect; for $v_i = z_i$ it would reduce to the latter effect.

44. *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 (26),
$s^{\text{beh}}_{i44}(x, z) = z_i \, x_{i+} \, \breve{v}_i$.
This is similar to the 'total alter' effect; for $v_i = z_i$ it would reduce to the latter effect.

45. *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

$$\breve{w}_i = \begin{cases} \dfrac{\sum_h x_{ih} w_{ih}}{x_{i+}} & \text{if } x_{i+} > 0 \\ 0 & \text{if } x_{i+} = 0. \end{cases} \tag{26}$$

leading to the effect
$s^{\text{beh}}_{i45}(x, z) = z_i \, \breve{w}_i$.

46. *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,
$s^{\text{beh}}_{i46}(x, z) = z_i \, \sum_h x_{ih} w_{ih}$.

47. *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 (26),
$s^{\text{beh}}_{i47}(x, z) = z_i \, x_{i+} \, \breve{v}_i$.
This is similar to the 'total alter' effect; for $v_i = z_i$ it would reduce to the latter effect.

48. *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

$$\breve{v}_i = \begin{cases} \dfrac{\sum_j x_{ji} v_j}{x_{+i}} & \text{if } x_{+i} > 0 \\ 0 & \text{if } x_{+i} = 0, \end{cases} \tag{27}$$

the effect being
$s^{\text{beh}}_{i48}(x, z) = z_i \, \breve{v}_i$.
This is similar to the 'average in-alter' effect; for $v_i = z_i$ it would reduce to the latter effect.

49. *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 (27),
$s^{\text{beh}}_{i49}(x, z) = z_i \, x_{i+} \, \breve{v}_i$.
This is similar to the 'total in-alter' effect; for $v_i = z_i$ it would reduce to the latter effect.

50. *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

$$\breve{v}_j^{(-i)} = \begin{cases} \dfrac{\sum_{h \neq i} x_{jh} \, v_h}{x_{j+} - x_{ji}} & \text{if } x_{j+} - x_{ji} > 0 \\ 0 & \text{if } x_{j+} - x_{ji} = 0. \end{cases} \tag{28}$$

(also see (15) and (24)),
$s^{\text{beh}}_{i50}(x, z) = z_i \big(\sum_j x_{ij} \, \breve{v}_j^{(-i)} \big) / \big(\sum_j x_{ij} \big)$
(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;

51. *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),
$s^{\text{beh}}_{i51}(x, z) = z_i \sum_j x_{ij} \, \sum_{h \neq i} x_{jh} \, v_h = z_i \sum_j x_{ij} \, (x_{j+} - x_{ji}) \, \breve{v}_j^{(-i)}$;
this is similar to `totAltDist2`, but now for covariate V instead of the behavior Z;

52. *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),
$s^{\text{beh}}_{i52}(x, z) = z_i \big(\sum_j x_{ij} \, (x_{j+} - x_{ji}) \, \breve{v}_j^{(-i)} \big) / \big(\sum_j x_{ij} \big)$
(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;

53. *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),
$s^{\text{beh}}_{i53}(x,z) = z_i\big(\sum_j x_{ij}\, \check{v}^{(-i)}_j\big)$;
this is similar to `totAAltDist2`, but now for covariate V instead of the behavior Z;

54. *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 $\check{v}^{(-i)}_j$ of his alters, excluding the contribution from a tie $j \to i$ (if any), defined by

$$\check{v}^{(-i)}_j = \begin{cases} \dfrac{\sum_{h\neq i} x_{hj}\, v_h}{x_{+j} - x_{ij}} & \text{if } x_{+j} - x_{ij} > 0 \\ 0 & \text{if } x_{+j} - x_{ij} = 0. \end{cases} \tag{29}$$

(also see (25)),
$s^{\text{beh}}_{i54}(x,z) = z_i\big(\sum_j x_{ij}\, \check{v}^{(-i)}_j\big)/\big(\sum_j x_{ij}\big)$
(and the mean behavior, i.e. 0, if the ratio is 0/0) ;
this is similar to `avInAltDist2`, but now for covariate V instead of the behavior Z;

55. *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),
$s^{\text{beh}}_{i55}(x,z) = z_i \sum_j x_{ij} \sum_{h\neq i} x_{hj}\, v_h = z_i \sum_j x_{ij}\, (x_{+j} - x_{ij})\, \check{v}^{(-i)}_j$;
this is similar to `totInAltDist2`, but now for covariate V instead of the behavior Z;

56. *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),
$s^{\text{beh}}_{i56}(x,z) = z_i\big(\sum_j x_{ij}\, (x_{+j} - x_{ij})\, \check{v}^{(-i)}_j\big)/\big(\sum_j x_{ij}\big)$
(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;

57. *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),
$s^{\text{beh}}_{i57}(x,z) = z_i\big(\sum_j x_{ij}\, \check{v}^{(-i)}_j\big)$;
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) and influence effects. These have to be specified using `interaction1` for the covariate and `interaction2` for the network, e.g.,

```
myCoevolutionEff <- includeEffects(myCoevolutionEff, avAltEgoX,
        name="smoking",  interaction1 = "sex", interaction2="friendship")
```

Between parentheses: the functionality of the 'ego'-variant of these effects is duplicated by interaction effects created, for example, as follows:

```
myCoevolutionEff <- includeInteraction(myCoevolutionEff, effFrom, avAlt,
        name="smoking", interaction1 = c("sex", "friendship"))
```

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.

58. *interaction of ego (tie sender) variable with average similarity*, `(avSimEgoX)`
$s^{\text{beh}}_{i58}(x,z) = (v_i/x_{i+}) \sum_j x_{ij}(\text{sim}^z_{ij} - \widehat{\text{sim}^z})$;
(and the mean behavior, i.e. 0, if $x_{i+} = 0$) ;

59. *interaction of ego (tie sender) variable with total similarity*, `(totSimEgoX)`
$s^{\text{beh}}_{i59}(x,z) = v_i \sum_j x_{ij}(\text{sim}^z_{ij} - \widehat{\text{sim}^z})$;

60. *interaction of ego (tie sender) variable with average alter*, `(avAltEgoX)`
$s^{\text{beh}}_{i60}(x,z) = v_i\, z_i\big(\sum_j x_{ij}\, z_j\big)/\big(\sum_j x_{ij}\big)$
(and the mean behavior, i.e. 0, if the ratio is 0/0) ;

61. *interaction of ego (tie sender) variable with total alter*, `(totAltEgoX)`
$s^{\text{beh}}_{i61}(x,z) = v_i\, z_i\big(\sum_j x_{ij}\, z_j\big)$;

62. *interaction of alter (tie receiver) variable with average similarity*, `(avSimAltX)`
$s^{\text{beh}}_{i62}(x,z) = (1/x_{i+}) \sum_j x_{ij}\, v_j(\text{sim}^z_{ij} - \widehat{\text{sim}^z})$;
(and the mean behavior, i.e. 0, if $x_{i+} = 0$) ;

63. *interaction of alter (tie receiver) variable with total similarity*, `(totSimAltX)`
$s^{\text{beh}}_{i63}(x,z) = \sum_j x_{ij}\, v_j(\text{sim}^z_{ij} - \widehat{\text{sim}^z})$;

64. *interaction of alter (tie receiver) variable with average alter*, `(avAltAltX)`
$s^{\text{beh}}_{i64}(x,z) = z_i\big(\sum_j x_{ij}\, v_j\, z_j\big)/\big(\sum_j x_{ij}\big)$
(and the mean behavior, i.e. 0, if the ratio is 0/0) ;

65. *interaction of alter (tie receiver) variable with total alter*, `(totAltAltX)`
$s^{\text{beh}}_{i65}(x,z) = z_i\big(\sum_j x_{ij}\, v_j\, z_j\big)$;

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.

66. *average alter weighted by dyadic covariate* W `(avAltW)`, a weighted version of `avAlt`; depending on the parameter value (1 or 2):
parameter = 1 : $s^{\text{beh}}_{i66}(x,z) = z_i\big(\sum_j x_{ij}\, w_{ij}\, z_j\big)/\big(\sum_j x_{ij}\big)$
parameter = 2 : $s^{\text{beh}}_{i66}(x,z) = z_i\big(\sum_j x_{ij}\, w_{ij}\, z_j\big)/\big(\sum_j w_{ij}\, x_{ij}\big)$;
(and the mean behavior, i.e. 0, if the ratio is 0/0) ;

67. *total alter weighted by dyadic covariate* W (`totAltW`), a weighted version of `totAlt`;
$s^{\text{beh}}_{i67}(x,z) = z_i \sum_j x_{ij}\, w_{ij}\, z_j$;

68. *average similarity weighted by dyadic covariate* W (`avSimW`), a weighted version of `avSim`;
depending on the parameter value (1 or 2):
parameter = 1 : $s^{\text{beh}}_{i68}(x,z) = \sum_j x_{ij}\, w_{ij}\, (\text{sim}^z_{ij} - \widehat{\text{sim}^z})/\big(\sum_j x_{ij}\big)$
parameter = 2 : $s^{\text{beh}}_{i68}(x,z) = \sum_j x_{ij}\, w_{ij}\, (\text{sim}^z_{ij} - \widehat{\text{sim}^z})/\big(\sum_j w_{ij}\, x_{ij}\big)$;
(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;

69. *total similarity weighted by dyadic covariate* W (`totSimW`) , a weighted version of `totSim`;
$s^{\text{beh}}_{i69}(x,z) = z_i \sum_j x_{ij}\, w_{ij}\, (\text{sim}^z_{ij} - \widehat{\text{sim}^z})$.

User-defined interaction effects

The *user-defined interaction effects* of Section 5.8.2 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

$$s^{\text{beh}}_{i[a*b]}(x,z) \;=\; \frac{1}{z_i}\, s^{\text{beh}}_{ia}(x,z)\, s^{\text{beh}}_{ib}(x,z) \;. \tag{30}$$

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

$$s^{\text{beh}}_{i[a*b*c]}(x,z) \;=\; \frac{1}{z_i^2}\, s^{\text{beh}}_{ia}(x,z)\, s^{\text{beh}}_{ib}(x,z)\, s^{\text{beh}}_{ic}(x,z) \;. \tag{31}$$

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

$$s^{\text{beh}}_{i[v_1*v_2]}(x,z) \;=\; \frac{1}{z_i}\, (z_i\, v_{1i})\, (z_i\, v_{2i}) \;=\; z_i\, v_{1i}\, v_{2i} \;, \tag{32}$$

where both component variables v_{1i} and v_{2i} are internally centered, but the product variable will generally not have mean 0.

12.2.2 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

$$\sum_{m=1}^{M-1}\sum_{i=1}^{n} I\{z_i(t_{m+1}) > z_i(t_m)\}\left(s_{ik}^{\mathrm{beh}}(x(t_{m+1})) - s_{ik}^{\mathrm{beh}}(x(t_m))\right), \tag{33}$$

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.

12.2.3 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

$$\sum_{m=1}^{M-1}\sum_{i=1}^{n} I\{z_i(t_{m+1}) < z_i(t_m)\}\left(s_{ik}^{\mathrm{beh}}(x(t_m)) - s_{ik}^{\mathrm{beh}}(x(t_{m+1}))\right), \tag{34}$$

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.

12.2.4 Behavioral rate function

The behavioral rate function λ^{beh} consists of a constant term per period,

$$\lambda_i^{\mathrm{beh}} = \rho_m^{\mathrm{beh}}$$

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.

⊙ 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 (2015). 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 $a_i(x)$.

2. *average exposure effect* (`avExposure`), defined as the proportion of i's alters who have adopted the innovation,

$$a_{i2}(y) = \frac{\sum_{j=1}^{n} z_j x_{ij}}{\sum_{j=1}^{n} x_{ij}};$$

3. *total exposure effect* (`totExposure`), defined as the number of i's alters who have adopted the innovation,

$$a_{i3}(y) = \sum_{j=1}^{n} z_j x_{ij};$$

4. *infection by indegree effect* (`infectIn`), defined as the sum of indegrees of i's alters who are also adopters of the innovation,

$$a_{i4}(y) = \sum_{j=1}^{n} z_j x_{ij} x_{+j};$$

5. *infection by outdegree effect* (`infectOut`), defined as the sum of outdegrees of i's alters who are also adopters of the innovation,

$$a_{i5}(y) = \sum_{j=1}^{n} z_j x_{ij} x_{j+};$$

6. *infection by covariate effect* (`infectCovar`), defined as the sum of covariate values of i's alters who are also adopters of the innovation,

$$a_{i6}(y) = \sum_{j=1}^{n} z_j x_{ij} v_j;$$

7. *susceptibility to average exposure by indegree effect* (`susceptAvIn`), defined as the interaction between i's indegree and i's average exposure,

$$a_{i7}(y) = x_{+i} \frac{\sum_{j=1}^{n} z_j x_{ij}}{\sum_{j=1}^{n} x_{ij}};$$

8. *susceptibility to average exposure by covariate effect* (`susceptAvCovar`), defined as the interaction between i's covariate value and i's average exposure,

$$a_{i8}(y) = v_i \frac{\sum_{j=1}^{n} z_j x_{ij}}{\sum_{j=1}^{n} x_{ij}}.$$

12.3 Effects for estimation by Generalized Method of Moments

A variety of effects can be specified in RSienaTest 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 (Section 6.8). This is still in full development, and will be documented later.

13 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 (9) as

$$f_i^{\text{net}}(x) = \sum_k \beta_k^{\text{net}} s_{ik}^{\text{net}}(x) \ .$$

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[16] 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 $f_i^{\text{net}}(x_b) - f_i^{\text{net}}(x_a)$ 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

$$\exp\left(f_i^{\text{net}}(x_b) - f_i^{\text{net}}(x_a)\right) \ .$$

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).

13.1 Networks

The evaluation function is a weighted sum of 'effects' $s_{ik}^{\text{net}}(x)$. Their formulae can be found in Section 12.1.1. 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 $s_{ik}^{\text{net}}(x)$ of adding the tie $i \to h$ minus the contribution of adding the tie $i \to j$ is the log odds ratio comparing the probabilities of i sending a new tie to h versus sending the tie to j, if all other effects $s_{ik}^{\text{net}}(x)$ yields 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, 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

[16] More exactly, the value is $\sqrt{\pi^2/6}$, the standard deviation of the Gumbel distribution; see Snijders (2001).

be seen in Section 12.1.1 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 odds 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.

13.2 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 (23). In many cases[17] the effect has the form of a product

$$s_{ik}^{\text{beh}}(x, z) \;=\; z_i\, s_{ik}^{0}(x, z) \;, \tag{35}$$

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-odds-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 odds 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 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

[17] For effects satisfying this condition, the `interactionType` is defined as `OK`.

are not of the form (35). To explain the log-odds or odds 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 $\beta_k^{\text{beh}}(a-b-c)/r$, 0, and $\beta_k^{\text{beh}}(c-a-b)/r$, respectively. The contributions for the average similarity effect are $\beta_k^{\text{beh}}(a-b-c)/(rx_{i+})$, 0, and $\beta_k^{\text{beh}}(c-a-b)/(rx_{i+})$. 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 $\hat{\beta}_k^{\text{beh}} = 3.9689$, 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 odds 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 odds 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.

13.3 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. For example, if in a network dynamics model the ego, alter, and similarity effects of a variable V are specified, then the formulae for their contribution to the evaluation function can be obtained from the components listed in Section 12.1.1 as

$$\beta_{\text{ego}}\, v_i\, x_{i+} \;+\; \beta_{\text{alter}} \sum_j x_{ij}\, v_j \;+\; \beta_{\text{sim}} \sum_j x_{ij}(\text{sim}_{ij}^v - \widehat{\text{sim}^v}) \;, \tag{36}$$

where the similarity score is

$$\text{sim}_{ij}^v = 1 - \frac{|v_i - v_j|}{\Delta_V} \;,$$

with $\Delta_V = \max_{ij} |v_i - v_j|$ being the observed range of the covariate v and where $\widehat{\text{sim}^v}$ is the mean of all similarity scores. The superscript $^{\text{net}}$ is left out of the notation for the parameters in order not to clutter the notation.

Similarly to how it was done above, the contribution to (36) of the tie from i to j, represented by the single tie variable x_{ij} — i.e., the difference between the values of (36) for $x_{ij} = 1$ and $x_{ij} = 0$ — can be calculated from this formula. It should be noted that all variables are internally centered by SIENA, and that the mean values used for the centering are given near the beginning of the input file. More precision can be obtained by requesting the "mean" attributes of the covariates, as explained in Section 4.2.2. This is made explicit in the following by the subtraction of the mean $\bar{v}$. The contribution of variable V to the network evaluation function of actor i is given by

$$\begin{aligned}
&\beta_{\text{ego}}\,(v_i - \bar{v}) \;+\; \beta_{\text{alter}}\,(v_j - \bar{v}) \;+\; \beta_{\text{sim}}\,(\text{sim}_{ij}^{v} - \widehat{\text{sim}^{v}}) \\
&\qquad = \beta_{\text{ego}}\,(v_i - \bar{v}) \;+\; \beta_{\text{alter}}\,(v_j - \bar{v}) \;+\; \beta_{\text{sim}}\left(1 - \frac{|v_i - v_j|}{\Delta_V} - \widehat{\text{sim}^{v}}\right) . \qquad (37)
\end{aligned}$$

From this equation a table can be made that gives the outcome of (37) for some values of v_i and v_j.

This can be concretely carried out using the data set s50 which is an excerpt of 50 girls in the data set used in Pearson and Michell (2000); Pearson and West (2003); Steglich et al. (2006) and Steglich et al. (2010). We refer to any of these papers for a further description of the data. The friendship network data over 3 waves are in the files s50-network1.dat, s50-network2.dat, and s50-network3.dat. We also use the attribute data for alcohol use, s50-alcohol.dat, as a dependent variable. It can be seen from the SIENA output file using these data that the alcohol use variable assumes values from 1 to 5, with overall mean equal to $\bar{v} = 3.113$, and mean of the similarity variable $\widehat{\text{sim}^{v}} = 0.6983$. Drug use is used as a changing actor variable, with range 1–4, average $\bar{v} = 1.5$ and average dyadic similarity $\widehat{\text{sim}^{v}} = 0.7533$.

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, transitive ties, number of distances two, the ego, alter, and similarity effects of alcohol use, as well as the ego, alter, and similarity effects of drug use; and for the behavior (i.e., alcohol) dynamics the shape effect, the effect of alcohol on itself (quadratic shape effect), and the average similarity effect.

The results obtained are given in the following part of the output file.

```
Network Dynamics
 1. rate:  constant network rate (period 1)          8.2357  (   1.6225)
 2. rate:  constant network rate (period 2)          5.6885  (   0.8434)
 3. eval:  outdegree (density)                      -2.1287  (   0.1565)
 4. eval:  reciprocity                               2.3205  (   0.2132)
 5. eval:  transitive ties                           0.2656  (   0.2025)
 6. eval:  number of actors at distance 2           -0.9947  (   0.2173)
 7. eval:  drink alter                               0.0899  (   0.1184)
 8. eval:  drink ego                                -0.0100  (   0.1087)
 9. eval:  drink similarity                          0.8994  (   0.5864)
10. eval:  drug use alter                           -0.1295  (   0.1282)
11. eval:  drug use ego                              0.1362  (   0.1253)
12. eval:  drug use similarity                       0.6650  (   0.3381)
```

```
Behavior Dynamics
13. rate:  rate drink period 1                          1.3376  (   0.3708)
14. rate:  rate drink period 2                          1.8323  (   0.4546)
15. eval:  behavior drink linear shape                  0.3618  (   0.1946)
16. eval:  behavior drink quadratic shape              -0.0600  (   0.1181)
17. eval:  behavior drink average similarity            3.9689  (   2.2053)
```

We interpret here the parameter estimates for the effects of drinking behavior and drug use without being concerned with the significance, or lack thereof. For the drinking behavior, formula (37) yields

$$-0.0100\,(v_i - \bar{v}) \,+\, 0.0899\,(v_j - \bar{v}) \,+\, 0.8994\Big(1 \,-\, \frac{|v_i - v_j|}{\Delta_V} - 0.6983\Big)\,.$$

In R the following code can be used to construct the selection table. Note that the function outer here is very convenient.

```
# First define a function that incorporates the relevant part
# of the evaluation function, dependent on the parameters b1, b2, b3,
# the overall average v_av, the similarity average sim_av,
# and the range ran_v
obj_n <- function(vi, vj){
           b1*(vi-v_av) + b2*(vj-v_av) + b3*(1 - abs(vi-vj)/ran_v - sim_av)
                          }
# Now fill in the values of the parameter estimates and the averages.
v_av   <-  3.113
sim_av <-  0.6983
ran_v  <-  4
b1     <- -0.0100
b2     <-  0.0899
b3     <-  0.8994

# Define the value of v for which the table is to be given.
vv     <- c(1, 2, 3, 4, 5)
# And calculate the table
sel_tab <- outer(vv, vv, obj_n)
# It can be displayed
sel_tab
# and if package xtable is loaded, also be written
# to a latex or html file. For example,
tab_sel <- xtable(sel_tab)
print(tab_sel,file="tab_sel.htm", type="html",
             html.table.attributes = "rules = none")
# The html.table.attributes option gives the <table> tag
# used in the html file.
```

The results can be tabulated as follows.

v_i \ v_j	1	2	3	4	5
1	0.10	–0.03	–0.17	–0.30	–0.44
2	–0.13	0.18	0.05	–0.09	–0.22
3	–0.37	–0.05	0.26	0.13	–0.01
4	–0.60	–0.29	0.03	0.34	0.21
5	–0.84	–0.52	–0.21	0.11	0.42

This table shows the preference for similar alters: in all rows, the highest value is at the diagonal ($v_j = v_i$). The ego and alter parameters are close to 0, therefore the similarity effect is dominant. However, note that the formula uses raw values for v_i and v_j but divides the values for the absolute difference $|v_i - v_j|$ by Δ_V which here is $5 - 1 = 4$. Therefore the weight of 0.09 for the alter effect is not completely negligible compared to the weight of 0.90 for the similarity effect. The positive alter effect leads to a preference for ties to alters with a high v_j value which goes against the similarity effect for $v_i = 1$ but strengthens the similarity effect for $v_i = 5$. The table shows that the net resulting preference for similar others is strongest for actors (egos) high on drinking behavior, and weakest for actors in the middle and low range of drinking behavior.

For drug use, the formula yields

$$0.1362\,(v_i - \bar{v}) \;-\; 0.1295\,(v_j - \bar{v}) \;+\; 0.665\Big(1 \;-\; \frac{|v_i - v_j|}{\Delta_V} \;-\; 0.7533\Big)\;,$$

which leads to the following table.

v_i \ v_j	1	2	3	4
1	0.16	–0.19	–0.54	–0.89
2	0.08	0.17	–0.18	–0.53
3	–0.01	0.08	0.17	–0.18
4	–0.10	–0.00	0.09	0.18

In each row the highest value is at the diagonal, which shows that indeed everybody prefers to be friends with similar others also with respect to drug use. The negative alter effect supports this for low v_i values and counteracts it for high v_i values. This is seen in the table in the strong preference of low drug users ($v_i = 1$) for others who are low on drug use, and the very weak preference for high drug users ($v_i = 4$) for others also high on drug use.

An alternative specification uses the drink ego × drink alter interaction together with the drink squared alter effect in the network dynamics model, and similarly for drug use; for the behavior dynamics, an alternative specification uses the average alter effect. This leads to the following table of results.

```
Network Dynamics
 1. rate:  constant network rate (period 1)              8.0978  (   1.5118)
```

```
 2. rate:  constant network rate (period 2)            5.7781  (   0.9474)
 3. eval:  outdegree (density)                        -2.1333  (   0.2196)
 4. eval:  reciprocity                                 2.3033  (   0.2184)
 5. eval:  transitive ties                             0.2430  (   0.2059)
 6. eval:  number of actors at distance 2             -1.0011  (   0.2275)
 7. eval:  drink alter                                 0.1041  (   0.1348)
 8. eval:  drink squared alter                         0.0141  (   0.1329)
 9. eval:  drink ego                                   0.0078  (   0.1157)
10. eval:  drink ego x drink alter                     0.1655  (   0.1095)
11. eval:  drug use alter                             -0.2603  (   0.2436)
12. eval:  drug use squared alter                     -0.0249  (   0.1945)
13. eval:  drug use ego                               -0.0214  (   0.1454)
14. eval:  drug use ego x drug use alter               0.1976  (   0.1146)

Behavior Dynamics
15. rate:  rate drink period 1                         1.3218  (   0.3632)
16. rate:  rate drink period 2                         1.7884  (   0.5053)
17. eval:  behavior drink linear shape                 0.3820  (   0.2421)
18. eval:  behavior drink quadratic shape             -0.5428  (   0.2839)
19. eval:  behavior drink average alter                1.1414  (   0.6737)
```

For this specification, the formulae in Section 12.1.1 imply that the components in the network evaluation function corresponding to the effects of variable V are

$$\beta_{\text{ego}}\,(v_i{-}\bar{v})\,x_{i+} + \beta_{\text{alter}}\sum_j x_{ij}\,(v_j{-}\bar{v}) + \beta_{\text{sq. alter}}\sum_j x_{ij}\,(v_j{-}\bar{v})^2 + \beta_{\text{e}\times\text{a}}\sum_j x_{ij}\,(v_i{-}\bar{v})\,(v_j{-}\bar{v})\;. \tag{38}$$

The contribution of the single tie variable x_{ij} to this formula is equal to

$$\beta_{\text{ego}}\,(v_i-\bar{v}) \;+\; \beta_{\text{alter}}\,(v_j-\bar{v}) \;+\; \beta_{\text{sq. alter}}\,(v_j-\bar{v})^2 \;+\; \beta_{\text{e}\times\text{a}}\,(v_i-\bar{v})\,(v_j-\bar{v})\;. \tag{39}$$

Filling in the estimates for the effects of drinking behavior yields

$$0.0078\,(v_i-\bar{v}) \;+\; 0.1041\,(v_j-\bar{v}) \;+\; 0.0141\,(v_j-\bar{v})^2 \;+\; 0.1655(v_i-\bar{v})\,(v_j-\bar{v})\;.$$

This can be represented in R as follows.

```
# First define a function that incorporates the relevant part
# of the evaluation function, dependent on the parameters b1, b2, b3,
# and the overall average v_av.
# Watch out for statements that take more than one line,
# as used here in the definition of the functions obj_n.
# The rule is that always, the lines before the last
# must be syntactically incomplete.
# In this case, this is satisfied because the first line ends with a +
obj_n <- function(vi, vj){
           b1*(vi-v_av) + b2*(vj-v_av) + b3*(vj-v_av)*(vj-v_av) +
                 b4*(vi-v_av)*(vj-v_av)
                           }
# Now fill in the values of the parameter estimates and the averages.
v_av   <-  3.113
```

```
b1     <-  0.0078
b2     <-  0.1041
b3     <-  0.0141
b4     <-  0.1655
# Define the value of v for which the table is to be given.
vv     <- c(1, 2, 3, 4, 5)
# And calculate and display the table
sel_tab <- outer(vv, vv, obj_n)
sel_tab
```

This gives the following[18] table.

$v_i \setminus v_j$	1	2	3	4	5
1	0.57	0.32	0.07	–0.17	–0.42
2	0.18	0.10	0.02	–0.06	–0.14
3	–0.18	–0.10	–0.01	0.08	0.16
4	–0.51	–0.26	–0.01	0.24	0.49
5	–0.81	–0.40	–0.02	0.43	0.85

For drug use we obtain the formula

$$-0.0214\,(v_i - \bar{v}) \;-\; 0.2603\,(v_j - \bar{v}) \;-\; 0.0249\,(v_j - \bar{v})^2 \;+\; 0.1976(v_i - \bar{v})\,(v_j - \bar{v}) \;.$$

and the following table.

$v_i \setminus v_j$	1	2	3	4
1	0.18	–0.18	–0.53	–0.89
2	0.06	–0.10	–0.26	–0.42
3	–0.11	–0.07	–0.03	0.00
4	–0.33	0.09	0.14	0.38

The fact that we are using three variables involving alter (alter, alter squared, interaction) instead of two (alter and similarity) leads to greater freedom in the curve that is fitted: the top (or, in the rare case of a reversed pattern, bottom) of the attractiveness of alters is not necessarily obtained at the diagonal, i.e., at ego's value. Straightforward calculus shows us that (39) is a quadratic function and obtains its extreme value (a maximum if $\beta_{\text{sq. alter}}$ is negative, a minimum if it is positive – the latter is, in general, less likely) for

$$v_j \;=\; \bar{v} \;-\; \frac{\beta_{\text{alter}} \;+\; \beta_{\text{e×a}}\,(v_i - \bar{v})}{2\,\beta_{\text{sq. alter}}} \;. \tag{40}$$

If the effect $\beta_{\text{sq. alter}}$ of the squared alter's value is negative and the interaction effect $\beta_{\text{e×a}}$ is positive, then this location of the maximum increases with ego's own value, v_i. Of course the number given by (40) will usually not be an integer number, so the actual value of v_j for which attractiveness is maximized is the integer in the range of V closest to (40).

[18] In earlier versions of the manual, there were some differences in this and the following tables, because too much rounding was used at an early stage.

For drinking there is a weak positive effect of squared drinking alter; the effect of squared drug use alter is weak negative. For drinking we see that the most attractive value for egos with $v_i = 1$ or 2 is no drinking, $v_j = 1$, whereas for egos with $v_i \geq 3$ the most attractive alters are those who drink most, $v_j = 5$. We also see that egos with the highest drinking behavior are those who differentiate most strongly depending on the drinking behavior of their potential friends.

For drug use the situation is different. Actors with $v_i = 1$ or 2 prefer friends with drug use $v_j = 1$; for actors with $v_i = 3$ the difference is hardly discernible, but if we consider the differences even though they are tiny, then they are most attracted to others with $v_j = 4$; actors with the highest drug use ($v_i = 4$) differentiate most strongly, and are attracted most to others with also the highest drug use.

The differences between the results with the similarity effects and the interaction effects are minor. The extra degrees of freedom of the latter model gives a slightly closer fit to the data. However, the differences between the two fits are not significant, as can be shown e.g. by score-type tests.

13.4 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 attractive are 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 12.2.1, and it must not be forgotten that all variables are internally centered in RSiena, and the subtracted means are reported in the initial output produced by print01Report, but more precision can be obtained by requesting the "mean" attributes of the covariates, see Section 4.2.2.

In the first model, the estimated coefficients in the behavior evaluation function are as follows.

```
15. eval:  behavior drink linear shape                0.3618  (   0.1946)
16. eval:  behavior drink quadratic shape            -0.0600  (   0.1181)
17. eval:  behavior drink average similarity          3.9689  (   2.2053)
```

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 12.2.1 show that the evaluation function for this model specification is

$$f_i^{\mathrm{beh}} \;=\; \beta_{\mathrm{trend}}\,(z_i - \bar{z}) \;+\; \beta_{\mathrm{drink}}\,(z_i - \bar{z})^2 \;+\; \beta_{\mathrm{av.\ sim}}\,\frac{1}{x_{i+}}\sum_j x_{ij}\big(\mathrm{sim}_{ij}^z - \widehat{\mathrm{sim}^z}\big)\ . \quad (41)$$

In the second model, the table gave the following results.

```
17. eval:  behavior drink linear shape                0.3820  (   0.2421)
18. eval:  behavior drink quadratic shape            -0.5428  (   0.2839)
19. eval:  behavior drink average alter               1.1414  (   0.6737)
```

Here the evaluation function is

$$f_i^{\text{beh}} \;=\; \beta_{\text{trend}}\,(z_i - \bar{z}) \;+\; \beta_{\text{drink}}\,(z_i - \bar{z})^2 \;+\; \beta_{\text{av. alter}}\,(z_i - \bar{z})(\bar{z}_{(i)} - \bar{z})\;, \tag{42}$$

where $\bar{z}_{(i)}$ is the average Z value of i's friends[19],

$$\bar{z}_{(i)} = \frac{1}{x_{i+}} \sum_j x_{ij}\, z_j\;.$$

Equation (42) is simpler than equation (41), because (42) 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 (41) depends on the entire distribution of the Z values of i's friends.

Suppose that, in model (41), 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 (42), on the other hand, since the average is the same, both actors would be drawn equally strongly toward the average value 3.

Since the objective function for model (41) depends not generally on the average behavior of the actor's friends, 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

$$f_i^{\text{beh}} \;=\; 0.3618\,(z_i - \bar{z}) \;-\; 0.0600\,(z_i - \bar{z})^2 \;+\; 3.9689\,\left(\text{sim}_{ij}^{z} - \widehat{\text{sim}^{z}}\right)\;.$$

This can be calculated by R as follows.

```
# Define part of evaluation function
obj_b <- function(zi, zj){
         b1*(zi-z_av) + b2*(zi-z_av)^2 + b3*(1 - abs(zi-zj)/ran_v - sim_av)
                        }
# Fill in the values of the parameter estimates and the averages.
z_av   <-  3.113
sim_av <-  0.6983
b1     <-  0.3618
b2     <- -0.06
b3     <-  3.9689
zz     <-  c(1, 2, 3, 4, 5)
# The table is transposed: zi in the columns!
t(outer(zz, zz, obj_b))
```

The result is the following[20].

[19] If i has no friends, i.e., $x_{i+} = 0$, then $\bar{z}_{(i)}$ is defined to be equal to $\bar{z}$.

[20] There were errors in this table in an earlier version of the manual.

$\bar{z}_{(i)}$ \ z_i	1	2	3	4	5
1	0.17	–0.27	–0.83	–1.51	–2.30
2	–0.83	0.72	0.16	–0.51	–1.31
3	–1.82	–0.27	1.16	0.48	–0.32
4	–2.81	–1.26	0.16	1.47	0.67
5	–3.80	–2.26	–0.83	0.48	1.67

The interpretation 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 $\bar{z}_{(i)}$, 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 less strong than in the case where all the friends drink a lot.

For the other model the objective function depends on the behavior of the focal actor's friends only as a function of their average behavior. Filling in the estimated parameters in (42) yields

$$f_i^{\text{beh}} \;=\; 0.3820\,(z_i - \bar{z}) \;-\; 0.5428\,(z_i - \bar{z})^2 \;+\; 1.1414\,(z_i - \bar{z})(\bar{z}_{(i)} - \bar{z}) \;.$$

For a given average Z values of i's friends, this is a quadratic function of z_i. The following table indicates the behavior evaluation function for z_i (columns) as a function of the average drinking behavior of i's friends (rows).

$\bar{z}_{(i)}$ \ z_i	1	2	3	4	5
1	1.87	1.59	0.22	–2.23	–5.76
2	–0.55	0.32	0.09	–1.22	–3.61
3	–2.96	–0.95	–0.04	–0.20	–1.46
4	–5.37	–2.22	–0.16	0.81	0.70
5	–7.78	–3.49	–0.29	1.82	2.85

We see that, even though the squared function does not necessarily draw the actors toward the average of their friends' behavior, for these parameters the highest values of the behavior evaluation function are obtained indeed when the focal actor (i) behaves just like the average of his friends. The values far away from the maximum contrast in this case more strongly than in the case of the model with the average similarity effect, but neither of these models fits clearly better than the other.

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.

Another way to look at the behavior evaluation function is to consider the location of its maximum. This function here can be written also as

$$f_i^{\text{beh}} \;=\; \big(0.3820 + 1.1414(\bar{z}_{(i)} - \bar{z})\big)\,(z_i - \bar{z}) \;-\; 0.5428\,(z_i - \bar{z})^2 \;.$$

Differentiating with respect to z_i shows that this function is maximal for

$$z_i = \bar{z} + \frac{0.3820 + 1.1414(\bar{z}_{(i)} - \bar{z})}{2 \times 0.5428} = 0.19 + 1.05\,\bar{z}_{(i)} ,$$

just a little bit larger than $\bar{z}_{(i)}$. Indeed in the table 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 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.

Note that it is not difficult to find the source of error messages in the code. The easiest way is searching for it by some search machine such as google. You will probably find it points to R-Forge 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, is to search directly in the code for this function. If the function is called, e.g., siena07, you can write the code to a file called `listing.txt` by the commands

```
sink("listing.txt")
siena07
sink()
```

Note that you should give the function name without parentheses.

After updating

Updates of RSiena and RSienaTest 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.

14.1 During estimation

Unlikely to terminate this epoch: more than 1000000 steps.

This can happen in function siena07; in conditional estimation (`COND=FALSE` in sienaAlgorithmCreate), 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 6.10.1.

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 6.10.1) 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 sienaAlgorithmCreate()).
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 sienaAlgorithmCreate 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 siena07. If the problem still occurs for `firstg`=0.02, use a smaller value (but less than 0.0001 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. In combination with trying out a lower value for `firstg`, you may try setting a higher value for `doubleAveraging` in sienaAlgorithmCreate. Double averaging is a different definition of the Robbins-Monro update step; see Section 6.1.3. 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 siena07, 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 print01Report).
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 6.10.2; this issue might be relevant here.)

14.2 As 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 siena07; or the score test requested in function sienaTimeTest. In the first case it indicates that there are linear dependencies in the list of effects (estimated and fixed) that are used for siena07. If this error message occurs for sienaTimeTest, it indicates linear dependencies in the

list of effects estimated in the siena07 run analyzed by this sienaTimeTest, together with the interactions with time dummies tested by sienaTimeTest. See Sections 5.9 and 8.2.

Solutions.
For siena07: drop some of the requested score-type test, retaining only a set of tested effects between which there are no linear dependencies.
For sienaTimeTest: exclude some of the requested time heterogeneity tests by the excludeEffects parameter.

14.3 In sienaGOF

Error in if (attr(obsData[[groupName]]$depvars[[varName]], "sparse")) { : argument is of length zero

This can happen directly when calling sienaGOF. It indicates that you used a wrong name for groupName or varName. See the help file for sienaGOF.

Solutions.
Use a correct groupName and varName.

Error in if (isbipartite) { : argument is of length zero

This can happen directly when calling sienaGOF. It indicates that you used a wrong name for groupName or varName. See the help file for sienaGOF.

Solutions.
Use a correct groupName and varName.

Part III
Programmers' manual

15 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, if RSiena is loaded, the command

```
sienaAlgorithmCreate
```

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 can be done by downloading, from CRAN or R-Forge, the 'tarball' with extension .tar.gz. This file can be extracted by compression/decompression programs (perhaps you need to do a double extraction). If you do not succeed in extracting the tar ball, see below for the use of `RTools` for this purpose. This will lead to a directory structure where at some place there is a directory called `RSiena` and/or a directory called `RSienaTest`, which includes the source code of the package.

In the file structure for `RSienaTest` there is a directory `doc` which contains a lot of programmers' documentation. These are in the form of LaTeX files, which can be compiled to produce .pdf files. The file `Siena_Algorithms.tex` contains a lot of details about algorithms used. For code developers, important files are `HowToCommit.tex` and `RSienaDeveloper.tex`. The latter file will guide you to the further documentation.

The file `Siena_algorithms.pdf` can also be downloaded from the SIENA website.

16 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.

17 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: you can do all these things with RSienaTest also.)

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_1.0.n.tar.gz
```

(where `n` 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_1.0.n.tar.gz
```

(where again `n` 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
```

18 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

$$\Delta_{kij}(x) = s_{ki}(x^{+ij}) - s_{ki}(x^{-ij}) \tag{43}$$

   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 12).
   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 18.2.

   - 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.

- 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 furethr changes necessary in the file effects.r; also see Section 18.2.

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++.

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.

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.

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.

18.1 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

$$s_i^{\text{net}}(x) = \min(x_{i+}\,,\, c) \tag{44}$$

where c is an internal effect parameter.

1. The change statistic (43) is

$$\Delta_{ij}(x) = \begin{cases} 1 & \text{if } \{x_{i+} < c, x_{ij} = 0\} \text{ or } \{x_{i+} \leq c, x_{ij} = 1\} \\ 0 & \text{else.} \end{cases} \tag{45}$$

   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 12 that depend only on the outdegree. Consider the effects Outdegree activity - sqrt (short name `outActSqrt`) and sum of (1/(out-degree + c) (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

$$s_i^{\text{net}}(x) = \sum_j x_{ij}\, r_{ij}(x) \ . \tag{46}$$

The latter defines the effect through a calculateContribution function and an egoStatistic function, which is the effect as defined in (44). 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 (46), 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 c in (44).

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 keyword 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 (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 c. 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 (45), 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 (45), 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. 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       <- sienaDependent(array(c(s501, s502), dim=c(50, 50, 2)))
mydata      <- sienaDataCreate(mynet)
myalgorithm <- sienaAlgorithmCreate(projname="s50_12")
myeff       <- getEffects(mydata)
myeff       <- setEffect(myeff, outTrunc, parameter = 3)
ans         <- siena07(myalgorithm, 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.

6. 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.

18.2 Notes on effectGroups and two-mode networks

For two-mode networks the difference between the two node sets implies some peculiarities. Recall that effectGroups are used in the file `allEffects.csv` and in function effects.r.

In the function getEffects in file effects.r, some additional measures are taken for effects in effectGroup covarBipartiteEff. This implies that for adding such effects, it will be necessary to see whether this function also must be modified; this will have to be done in function covarBipartiteEff(). (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.

For effects defined for two dependent networks and one actor covariate, the following effectGroups are defined:

- covarNetNetObjective is for effects where the second network is one-mode;
- covarABNetNetObjective is without restriction on the modes;
- covarANetNetObjective is for effects where, if the second network is two-mode, the covariate is defined for the first node set;
- covarBNetNetObjective is for effects where, if the second network is two-mode, the covariate is defined for the second node set.

Further some of the complex effectGroups are the following:

- tripleNetworkObjective is for combinations of three networks where for the two in the role of explanatory networks, either both should be one-mode, or both should be bipartite with the same second node sets.
- 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.

A List of Functions in Order of Execution

This appendix provides a description of the functions that constitute the RSiena package. This is intended as a quick reference or catalogue for the user to employ Stochastic Actor Oriented Models (SAOM) to analyze network dynamics in R.

The functions are presented in execution order (more or less as they would be used in practice). A list of useful R functions to read and prepare the data set is also included at the beginning. In all cases examples on how to use these functions are provided. In the 'syntax' column, when arguments of functions are followed by = and a single option, this is the default option.

The descriptions provided are suitable for beginner and intermediate R and Siena users. For the advanced specifications of the functions the user should refer to the help by typing "?funName" in the R console, where "funName" is the name of the function.

We consider that the model estimation is composed by 6 stages:

1. Getting started
2. Get the data the right format or check that it is in the correct format
3. Data specification
4. Model specification
5. Model estimation
6. Working with the results

Tables 3 and 4 present the list of useful R functions and the list of RSiena functions in execution order, respectively.

Stage	Name	Syntax	Examples	Description
1	help*	help(funame)	help(siena01Gui)	Opens the help on the function named "funame"; this can also be done by typing "?" followed by "funame" in the console. This is the general way to get information about further options of this function.
1	getwd	getwd()		Returns the name of the current working directory. Does not require arguments
1	list.files	list.files(dir)	list.files ("C:/User/ MyDocuments/ MySiena")	Returns a character vector with the names of the files in the directory "dir". If no argument is provided, "dir" is the current working directory.
1	setwd*	setwd(dir)	setwd("C:/MyDocuments/ MySiena")	Sets the working directory to "dir". In this context the working directory should be where the data is saved
1	install.packages*	install.packages()		It is used to install packages. If no arguments are provided it opens a GUI to select a mirror site and the packages that we want to download and install. This is not necessary if the package has already been installed.
1	library*	library(package)	library(RSiena)	Loads the library named "package".
1	read.table	read.table(file, header=FALSE, sep="", quote="",...)	net1 <- read.table('network1.dat', header=F)	Reads a file in table format and creates a data frame from it. The argument "file" is the file containing the data. In the case of adjacency matrices, the file should have the same number of columns and rows. "header" is a logical argument indicating whether the first row of the data contains the column names. "sep" is the field separator character (such as space, comma, etc.). See the help on the function to specify other arguments
2	as.matrix	as.matrix(x,...)	net1 <- as.matrix(net1)	Transforms an object "x" into a matrix. Siena works with matrices and not with data frames
2	class	class(x)	class(net1)	Returns the type of object that "x" is
2	dim	dim(x)	dim(net1)	Returns the dimension of object "x"
4	fix*	fix(x)	fix(effects)	Allows editing the object "x" by opening a window and it replaces the old object by the edited "x"

Table 3: Useful functions from R in execution order

* Also available via a menu option

Table 4: List of RSiena Functions in order of Execution

Stage	Name	Syntax	Examples	Description
1	installGui	installGui()		Starts the installer for the standalone version of RSiena. Only for Windows. Does not require arguments
3 – 5	siena01Gui	siena01Gui()		Does not require arguments. Opens a GUI to be used to run the model estimation or to create a session from which to work within R. Details on how to run the estimation under the GUI can be found in section **??** and **??**.
3	sienaNodeSet	sienaNodeSet (n, nodeSetName= "Actors", names=NULL)		Creates a Siena node set which can be used as the nodes in a siena network. "n" is the number of actors or nodes; "nodeSetName" is a character string to name the node set (defaults to "Actors") and "'names" is a string vector with length n with the names of each node (optional)
3	sienaDependent	sienaDependent (netarray, type= c("oneMode", "bipartite", "behavior"), nodeSet="Actors", sparse=is.list (netarray))	sienaDependent(array(c(net1,net2,net3), dim=c(dim(net1),3)))	Creates a Siena network object by forming an array of network observations represented as matrices, arrays or sparse matrices. "netarray" is a matrix (type="behavior" only) or array of values or list of sparse matrices of type "dgTMatrix";"type" is either "one mode" (default), "bipartite" or "behaviour"; "nodeSet" is the name of the node set. It is a vector with two strings for a bipartite network; "sparse" is logical and it is set to TRUE if the data is in sparse matrix format, FALSE otherwise
3	coCovar	coCovar(val, nodeSet ='Actors')	cons <– as.matrix(read.table ('cons.DAT')) cons1 <– coCovar (cons[,1])	Creates a constant covariate object, where val is the vector of covariate values and nodeSet is the name of the actors' set. The dimension of val should be (1, # Actors)
3	varCovar	varCovar(val, nodeSet ='Actors')	chan <– as.matrix (read.table ('chan.DAT')) chan <– varCovar (chan[,1])	Creates a changing covariate object where "val" is a matrix with the covariate values with one row for each actor and one column for each period; "nodeSet" is the name of the set of actors

Continued...

Table 4 – continued from previous page

Stage	Name	Syntax	Examples	Description
3	coDyadCovar	coDyadCovar(val, nodeSets= c("Actors", "Actors"))		Creates a constant dyadic covariate object where "val" is a matrix of the same dimension as the network observations and nodeSets are the sets of actors with which the constant covariate is associated
3	varDyadCovar	varDyadCovar(val, nodeSets= c("Actors", "Actors"))		Creates a changing dyadic covariate object where "val" is an array of matrices. Each matrix has the same dimension of the actor set and "val" has one less matrices than observations of the network; "nodeSets" are the sets of actors to which the varying covariate object is associated
3	sienaCompositionChange	sienaCompositionChange(changelist, nodeSet="Actors", option=1)		Creates a list of events describing the moments in which each actor is present in the network: "changelist" is a list with an entry for each actor in the node set. Each entry is a vector indicating intervals in which an actor is present in the network. "nodeSet" is the name of the set of actors corresponding to these composition changes and "option" (defaults to 1) is an integer controlling the processing of the network entries for the actors not currently present. See help(sienaCompositionChange) for details on this
3	sienaCompositionChangeFromFile	sienaCompositionChangeFromFile (filename, nodeSet="Actors", fileobj=NULL, option=1)		Creates a list of events describing the changes over time in the actor set from a file. "filename" is the name of the file containing change information (one line per actor) each line is a series of space delimited numbers indicating intervals. "fileobj" is the result of readLines on "filename". "nodeSet" is the name of the set of actors. "option" (defaults to 1) has the same description that in sienaCompositionChange

Continued...

Table 4 – continued from previous page

Stage	Name	Syntax	Examples	Description
3	sienaDataCreate	sienaDataCreate(..., nodeSets=NULL, getDocumentation=FALSE)	MyData <– sienaDataCreate (net, cons1, cons2, cons3, chan, dyad)	Creates a siena object from networks, covariates, composition and behaviour objects: ."..." represents the objects of class "sienaDependent", "coCovar", "varCovar", "coDyadCovar", "varDyadCovar", "compositionChange". "nodeSets" is a list of Siena node sets. Default is a single set named "Actors" with length equal to the number of rows in the first object of class "SienaNet", it has to match the nodeSet supplied when the arguments are created; "getDocumentation" is a flag to allow documentation for internal functions, not for use by users
3	sienaDataCreateFromSession	sienaDataCreateFromSession(filename=NULL, session=NULL, modelName="Siena", ...)	myobj <– sienaDataCreateFromSession ('Session.csv')	Reads a SIENA session from a file and creates a Siena Data object or group. "file" is the session file; "session" is the input session if the function is called from siena01Gui(); "modelName" is the project's name; "..." refers to other arguments used by siena01Gui()
3	sienaGroupCreate	sienaGroupCreate (objlist, singleOK=FALSE, getDocumentation=FALSE)	sienaGroupCreate (list(MyData1, MyData2))	Creates an object of class "sienaGroup" from a list of Siena data objects: "objlist" is a list of objects of class "siena"; "singleOK" is a boolean variable to indicate if it is OK to have just one object; "getDocumentation" is a flag to allow documentation of internal functions, not for use by users
4	effectsDocumentation	effectsDocumentation()		Prints a html or LaTeX table with the effects details
4	getEffects	getEffects(x, nintn=10, behNintn=4, getDocumentation=FALSE)	MyEff <– getEffects (MyData, nint=2, behNint=1)	Creates a siena effects objects (a data frame) that contains a list of the effects that can be included in the model. Type fix(MyEff) to edit the effects through a GUI (e.g. Including them or excluding them, changing their names, initial values, fixing them, etc.) The arguments are a siena or a siena group object "x", the number of lines for user defined network interactions "nint" and the number of lines for user defined behaviour interactions "behNintn". "getDocumentation" is a flag to allow documentation for internal functions, not to be used by users

Continued...

Table 4 – continued from previous page

Stage	Name	Syntax	Examples	Description
4	includeEffects	includeEffects(myeff, ..., include=TRUE, name=myeff$name[1], type="eval", interaction1="", interaction2="")	MyEff<– includeEffects(MyEff, transTrip, balance) MyEff<– includeEffects(MyEff, sameX, sameXRecip, interaction1="gender")	The function is a way to select the effects to be included. "myeff" is an effects object, as created by getEffects. It is necessary to indicate the short names to identify the effects to be included (argument ...). Use myeff$shortName to get a list of the short names of possible effects to include and myeff$effectName to get the full name of the effects. This information can also be found in the documentation created by effectsDocumentation(). The "include=TRUE" indicates that we want to include the "..." effects in the model, it can be set to FALSE to exclude effects from the model. "name" is the name of the network for which effects are being included. "type" is to include "eval" (evaluation function effects) or "endow" (endowment function effects). "interaction1" and "interaction2" are names of siena objects (where needed) to completely identify the effects e.g. covariate name or behavior variable name. Use myeff$effectName[myeff$include] to get the names of the included effects. It returns a new effects object, so it is important to assign it to a name

Continued...

Table 4 – continued from previous page

Stage	Name	Syntax	Examples	Description
4	includeInteraction	includeInteraction(myeff, ..., include=TRUE, name=myeff $name[1], type =“eval”, interaction1=rep(“”, 3), interaction2=rep(“”, 3))	MyEff<– includeInteraction(MyEff, transTrip, egoX, interaction1= c(“”,“beh”))	This function provides an interface to allow easy update of an unspecified interaction row in a Siena effects object. “myeff” is a Siena effects object as created by getEffects. To specify the effects to interact, list their short names instead of “...”; “include” is a boolean variable, default TRUE to include the interaction, it can be switched to FALSE to turn off an interaction. “name” is the name of network for which interactions are being defined. Defaults to the first in the effects object. “type” is the type of effects to be interacted: currently only “eval” or “endow”. “interaction1” is a vector of siena objects where needed to completely identify the effect e.g. covariate name or behavior variable name. Trailing blanks may be omitted. “interaction2” is a vector of siena objects where needed to completely identify the effect e.g. covariate name or behavior variable name. Trailing blanks may be omitted.
4	setEffect	setEffect(myeff, shortName, parameter=0, fix=FALSE, test=FALSE, initialValue=0, include=TRUE, name=myeff $name[1], type=“eval”, interaction1 = “”, interaction2 = “”)	MyEff <– setEffect(MyEff, transTrip, initialValue=3, include=T)	Interface to change the attributes of a particular effect. The required arguments are an effect object (“myeff”), the short name of the effect to modify (“shortName”) and a required integer value that defaults to zero (“parameter”). Depending on what it is desired to be modified we can supply: “fix=TRUE”, if we wish to fix that parameter; “test = TRUE” if we wish to test that parameter; “initialValue=2” (or any desired number) to modify the effect’s initial value (Defaults to zero); “include=TRUE or FALSE” depending on whether we want to include/exclude the effect (defaults to TRUE). The arguments “name”, “type” and “interaction1” and “interaction2” are defined as in includeInteraction and includeEffects.

Continued...

Table 4 – continued from previous page

Stage	Name	Syntax	Examples	Description
5	print01Report	print01Report(data, modelname="Siena", session=NULL, getDocumentation=FALSE)	print01Report(MyData)	Prints a report of a Siena data object and its default effects. We need to supply a Siena data object ("data") a siena effects object ("myeff") and a model name ("modelname") that defaults to "Siena". It creates and saves a file named "modelname.out" (Siena.out) that contains preliminary information on the data.

Continued...

Table 4 – continued from previous page

Stage	Name	Syntax	Examples	Description
5	sienaAlgorithmCreate	sienaAlgorithmCreate(fn=simstats0c, projname="Siena", MaxDegree=0, useStdInits=FALSE, n3=1000, nsub=4, maxlike=FALSE, diag=!maxlike, condvarno=0, condname="", firstg=0.2, cond=NA, findiff=FALSE, seed=NULL)	MyAlgorithm <- sienaAlgorithmCreate (projname = "MyProject")	Creates a siena algorithm object that can be used to call siena07. "fn" is function to do one simulation in the Robbins-Monro algorithm. "projname" is character string name of project. No embedded spaces. "MaxDegree" is a named vector of maximum degree values for corresponding networks. "useStdInits" is a boolean variable, if TRUE, the initial values in the effects object will be ignored and default values used instead. "n3" is the number of iterations in phase 3 (defaults to 1000). "nsub" is the number of subphases in phase 2 (defaults to 4). "maxlike", boolean to indicate whether to use maximum likelihood method or straightforward simulation (defaults to false). "diag" is boolean to indicate if the complete estimated derivative matrix should be used; "condvarno", if conditional estimation is used the parameter is the sequential number of the network or behaviour variable on which to condition. "condname" is the name of the dependent variable on which to condition (only use condname or condvar, not both). "firstg" initial value of gain parameter in the Robbins-Monro procedure. "cond" is boolean, If TRUE, use conditional simulation. If missing, decision is deferred until siena07 is called, when it is set to TRUE if there is only one dependent variable, FALSE otherwise. "findiff" is boolean, if TRUE, estimate derivatives using finite differences and if FALSE, use scores. "seed" is an integer referring to the starting value of random seed. Not used if parallel testing.
5	model.create			See sienaAlgorithmCreate

Continued...

Table 4 – continued from previous page

Stage	Name	Syntax	Examples	Description
5	siena07	siena07(x, batch=FALSE, verbose=FALSE, silent=FALSE, useCluster=FALSE, nbrNodes=2, initC=FALSE, clusterString= rep("localhost", nbrNodes), tt=NULL, parallelTesting=FALSE, ...)	ans <– siena07(MyModel, data=MyData, effects=MyEff, batch=FALSE, verbose=TRUE, useCluster=TRUE, nbrNodes=2)	Estimates parameters using Robbins-Monro algorithm. Note that the particular model to be used is passed on as the algorithm object, and data for the model must be passed by using named arguments. "x" is a model object; "batch" is a boolean variable to indicate if it is desired to open the GUI of Siena simulation; "verbose" is a boolean variable to produce output on the console; "silent" is also a boolean variable, if true, no output is printed to the console; "useCluster" is a boolean variable to indicate if it is desired to use a cluster of processes; "nbrNodes" is the number of processes to use if useCluster is TRUE; ' "clusterString" is the definition of clusters, default set up to use the local machine only; siena07 returns an object of class sienaFit (let's say ans). The main attributes are ans $theta, which are the estimated coefficients; ans $covtheta is the estimated covariance matrix of theta; ans $dfra is the matrix of estimated derivatives; ans $targets and ans $ targets2 are the observed statistics and the observed statistic by wave, respectively; ans $ssc are the score function contributions for each wave for each simulation in phase 3: ans $sims is the simulated networks as edgelists. Use names(ans) to obtain more characteristics; only recommended if you are proficient in RSiena.
6	print.sienaFit	print(x, tstat=TRUE, ...)	print(ans)	The function prints a table containing the estimated parameter values, standard errors and (optionally) t-statistics for convergence. If "x " is a summary(sienaFit) it prints on the console all the summary elements. "tstat" is a boolean argument, set to TRUE if it is desired for the t-statistics for convergence to be added to the report

Continued...

Table 4 – continued from previous page

Stage	Name	Syntax	Examples	Description
6	summary.sienaFit	summary(x,...)	summary(ans)	Prints a table containing the estimated parameter values, standard errors and t-statistics for convergence together with the covariance matrix of the estimates, the derivative matrix of expected statistics D by parameters, and the covariance matrix of the expected statistics D. The only required argument is a "sienaFit" object "x", as produced by siena07.
6	xtable.sienaFit	xtable(x, caption=NULL, label=NULL, align=NULL, digits=NULL, display=NULL, ...)	sienaxtab <– xtable(ans, caption="My Table", digits=2).	Creates an object of class xtable.sienaFit which inherits from class xtable and passes an extra arguments to the print.xtable. The argument is a sienaFit object "x".

B Changes compared to earlier versions

This only presents changes which affect the user, until version 1.0.6 of end October 2009. For more complete information about changes, consult the ChangeLog file in the source code on CRAN or in the R-Forge repository, which contains a more complete listing.

- 2016-05-28 Revision 294.
  Changes in RSiena and RSienaTest:
  - This manual taken out of the source code distribution; 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 $\alpha = -\log\big(1-\exp(\alpha')\big)$; 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 nondirected 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 sparseMatrix-Extraction() 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.

  For RSiena and 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 print method 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, RSiena and 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 undirected 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) undirected networkd 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 of siena07 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 for RSiena and 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 97 RSiena 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 sienaTimeTest 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 for siena07.
- 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 for siena07.
- 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 print and 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.

C 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. (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.

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. (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.

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.

Goldthorpe, J. H. (2001). Causation, statistics, and sociology. *European Sociological Review*, 17:1–20.

Greenan, C. C. (2015). Diffusion of innovations in dynamic networks. *Journal of the Royal Statistical Society, Series A*, 178:147–166.

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. (1973). The structural implications of measurement error in sociometry. *Journal of Mathematical Sociology*, 3:85–111.

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.

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. (2015). Multilevel longitudinal analysis of social networks. In preparation.

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 Torlò, 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. (2010). Testing and modeling time heterogeneity in longitudinal studies of social networks: A tutorial in RSiena. *In progress*.

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.

Pearson, M. A. and Michell, L. (2000). Smoke rings: Social network analysis of friendship groups, smoking and drug-taking. *Drugs: Education, Prevention and Policy*, 7(1):21–37.

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.

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.

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. (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. (2007). Analysing dynamics of non-directed social networks. In preparation. Transparencies available at internet.

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*. London: Sage, 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., Lomi, A., and Torlò, 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 Steglich, C. E. G. (2017a). *Actor-based Models for Analyzing Network Dynamics*. Cambridge University Press, Cambridge.

Snijders, T. A. B. and Steglich, C. E. G., editors (2017b). *Social Network Dynamics by Examples*. Cambridge University Press, Cambridge.

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.

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.

Steglich, C. E. G., Snijders, T. A. B., and West, P. (2006). Applying SIENA: An illustrative analysis of the coevolution of adolescents' friendship networks, taste in music, and alcohol consumption. *Methodology*, 2:48–56.

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.

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.