Manual for RSiena

Ruth M. Ripley
Tom A.B. Snijders
Paulina Preciado

University of Oxford: Department of Statistics; Nuffield College

January 29, 2012

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, which also may be called SIENA version 4, a contributed package to the statistical system R. The manual is based on the earlier manual for SIENA version 3, and also contains contributions written for that manual by Mark Huisman, Michael Schweinberger, and Christian Steglich.

Contents

1 General information

SIENA[1], shorthand for Simulation Investigation for Empirical Network Analysis, is a computer program that carries out the statistical estimation of models for repeated measures of social networks according to the dynamic actor-oriented model of Snijders and van Duijn (1997), Snijders (2001), Snijders et al. (2007), and Snijders et al. (2010a); also see Steglich et al. (2010). A tutorial for these models is in Snijders et al. (2010b). Some examples are presented, e.g., in van de Bunt (1999); van de Bunt et al. (1999) and van Duijn et al. (2003); and Steglich et al. (2006).

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.

This is a manual for RSiena, which also may be called SIENA version 4.0; the manual is provisional in the sense that it still is continually being updated. 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 manual. If desired, SIENA can be operated *apparently* independently of R, as is explained in Section 2.2.

RSiena was programmed by Ruth Ripley and Krists Boitmanis, in collaboration with Tom Snijders.

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. Sometimes latest 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.

We are grateful to NIH (National Institutes of Health) for their funding of programming RSiena. This is 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 integrated research program *The dynamics of networks and behavior* (project number 401-01-550), the project *Statistical methods for the joint development of individual behavior and peer networks* (project number 575-28-012), the project *An open software system for the statistical analysis of social networks* (project number 405-20-20), and to the foundation ProGAMMA, which all contributed to the work on SIENA.

[1]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).

Part I
Minimal Intro

2 Getting started with SIENA

There are two ways of getting started with SIENA: by using the graphical user interface (*GUI*) via siena01Gui or by using R commands in the regular R way. As a preliminary, we start with section 2.1 on data formats. Then we give a minimal cookbook-style introduction for getting started with SIENA using the graphical user interface via siena01Gui. In Section 2.3 we explain how to run SIENA as the package RSiena from within R. If you are looking for help with a specific problem, read Section 2.9.

2.1 Data formats

1. Network and covariate files should be text files with a row for each node. The numbers should be separated by spaces or tabs.

2. An exogenous events file can be given, indicating change of composition of the network in the sense that some actors are not part of the network during all the observations. This will trigger treatment of such change of composition according to Huisman and Snijders (2003). This file must have one row for each node. Each row should be consist of a set of pairs of numbers which indicate the periods during which the corresponding actor was present. For example,

   ```
   1 3
   1.5 3
   1 1.4 2.3 3
   2.4 3
   ```

   would describe a network with 4 nodes, and 3 observations. Actor 1 is present all the time, actor 2 joins at time 1.5, actor 3 leaves and time 1.4 then rejoins at time 2.3, actor 4 joins at time 2.4. All intervals are treated as closed.

3. If you use software such as Excel or SPSS to create input files to use with RSiena on a Mac, try to ensure that you do not create Unicode[2] files. This is an option in SPSS, and will depend on the file type with Excel.

More about input data can be found in Section 4.

[2]Unicode is one of the standards for encoding text, an alternative to ASCII. Unicode formats are denoted by symbols such as UTF-8 and UCS-2.

2.2 Installation and running the graphical user interface under Windows

1. Install R (most recent version). Note that if this leads to any problems or questions, R has an extensive list of 'frequently asked questions' which may contain adequate help for you.
   Start R, click on Packages and then on Install packages(s).... You will be prompted to select a mirror for download. Then select the packages xtable, network, rlecuyer, snow, and RSiena. (There may be later zipped version of RSiena available on our web site: to install this, use Install package(s) from local zip files, and select RSiena.zip (with the appropriate version number in the file name).
   If you are using Windows Vista and get an error of denied permission when trying to install the packages, you may get around this by right-clicking the R icon and selecting 'Run as administrator'.

2. If you want to get the latest beta version of RSiena, before installing the packages, select Packages/Select repositories... and select R-forge. Then install the packages in the normal way.

3. Start up R from the start menu or by (double-)clicking a shortcut on the taskbar (or desktop).

4. By right-clicking the shortcut and clicking 'Properties' you can change the startup working directory, given in the 'Start in' field. Data files will be searched for in the first instance in this directory.

5. Load the RSiena package via the menu `Packages`

6. Type
   `siena01Gui()`

7. You should see a screen like that shown in Figure 1

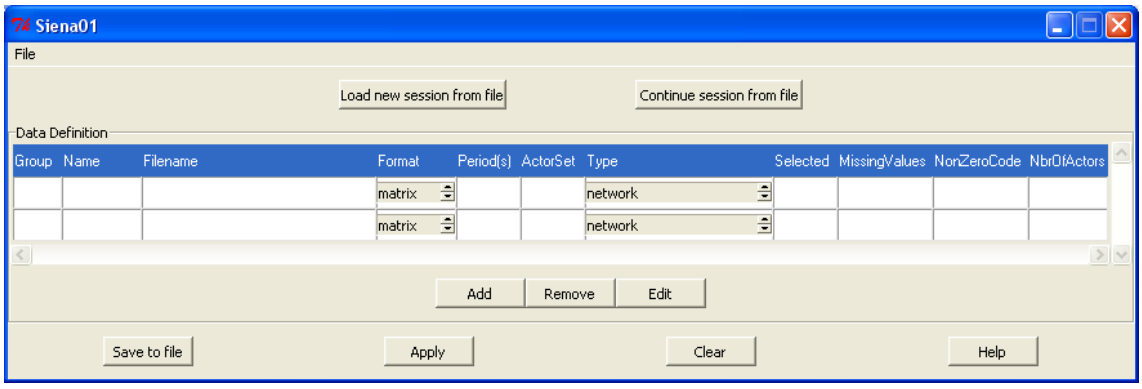


Figure 1: Siena Data Entry Screen

8. If the initial screen appears correctly, then check your working directory or folder. This is the directory that is opened immediately when clicking the Add button. Various problems can be avoided by making sure that the working directory is the directory that also contains the data files and the saved session file (see below)!
   You need to have permission to write files in the working directory, and the data files you want to use need to be in the same directory. To change the directory:

   (a) Right click on the shortcut, and select Properties. (if somehow you don't have permission to do this, try copying the shortcut and pasting to create another with fewer restrictions. (This may not work in Windows 7: you may need to copy it from the visible desktop and then paste it in Windows Explorer in your personal Desktop area.)) In the Start in: field type the name of the directory in which you wish to work, i.e., a directory in which you can both read and write files. Then click OK.

   (b) To run the examples, put the session file and the data files in the chosen directory before starting RSiena.

   (c) To use your own data, put that data in the chosen directory before starting RSiena.

2.2.1 Using the graphical user interface from Mac or Linux

1. Install R (most recent version) as appropriate for your computer.

2. Within R, type
   install.packages("RSiena")
   To use the latest beta version, use
   install.packages("RSiena", repos="http://R-Forge.R-project.org")

3. Navigate to the directory RSiena package, (which you can find from within R by running system.file(package="RSiena")) and find a file called sienascript. Run this to produce the Siena GUI screen.(You will probably have to change the permissions first (e.g.
   chmod u+x sienascript)).

4. If you want to use the GUI, you need tcl/tk installed. This is an (optional) part of the R installation on Mac. On Linux, you may need to install Tcl/tk and the extra Tcl/tk package tktable. On Ubuntu Linux, the following commands will do what is necessary (perhaps version numbers must be adapted):[3]

```
sudo apt-get install tk8.5
sudo apt-get install libtktable2.9
```

[3]Thanks to Michael Schweinberger and Krists Boitmanis for supplying these commands.

2.2.2 Running the graphical user interface: more details

Originally RSiena provided access to the GUI interface direct from Windows. This is not now possible. This section details some helpful notes about starting RSiena within R[4]. This is done by starting up R and working with the following commands. Note that R is case-sensitive, so you must use upper and lower case letters as indicated.

First, set the 'working directory' of the R session to the same directory that holds the data files; for example,
setwd('C:/SienaTest')
(Note the forward slash[5], and the quotes are necessary[6].) Windows users can use the Change dir... option on the File menu.

You can use the following commands to make sure the working directory is what you intend and see which files are included in it:
getwd()
list.files()

Assuming you see the data files, then you can proceed to load the RSiena package, with the library function:
library(RSiena)
The other packages will be loaded as required, but if you wish to examine them or use other facilities from them you can load them using:
library(snow)
library(network)
library(rlecuyer)
The following command will give a review of the functions that RSiena offers:
library(help=RSiena)
After that, you can use the RSiena GUI. It will 'launch' out of the R session.
siena01Gui()
You can monitor the R window for error messages – sometimes they are informative.

When you are done, quit R in the polite way:
q()
(Windows users may quit from the File menu or by closing the window.)

2.2.3 Entering Data.

There are two ways to enter the data.

1. Enter each of your data files using Add.
   Fill in the various columns as described in Section 2.2.5.

2. If you have earlier saved the specification of data files, e.g., using Save to file, then you can use Load new session from File.
   This requires a file in the format described at the end of Section 2.2.5; such a file

[4]We are grateful to Paul Johnson for supplying these ideas.
[5]You can use backward ones but they must be doubled: setwd('C:\\SienaTest').
[6]Single or double, as long as they match.

can be created and read in an editor or spreadsheet program, and it is created in .csv (comma separated) format by the siena01Gui() when you request Save to file.

3. If you wish to remove files, use the Remove option rather than blanking out the entries.

Once you have done this, check that the Format, Period, Type, etc., are correct, and enter any values which indicate missingness in the Missing Values column. A (minimal) complete screen is shown in Figure 2. The details of this screen are explained in Section 2.2.5.

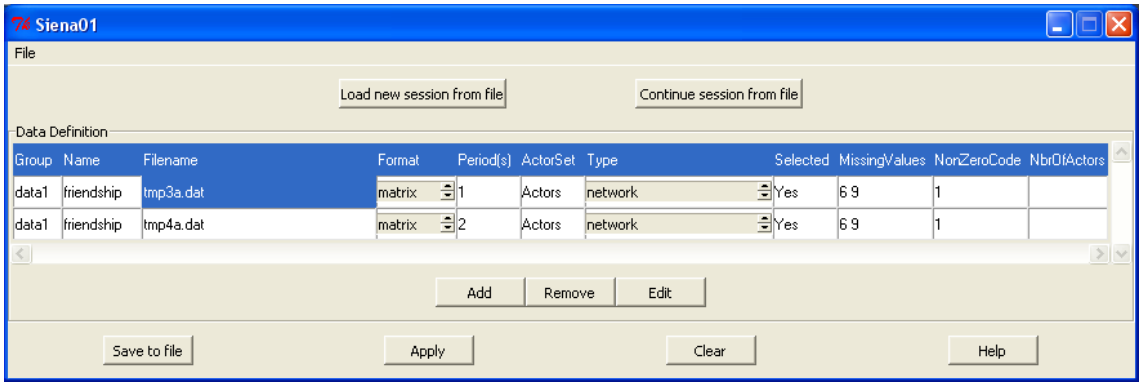


Figure 2: Example of a Completed Data Entry Screen

2.2.4 Running the Estimation Program

1. Click Apply: you will be prompted to save your work. Then you should see the Model Options screen shown in Figure 3. If this does not happen, then one possible

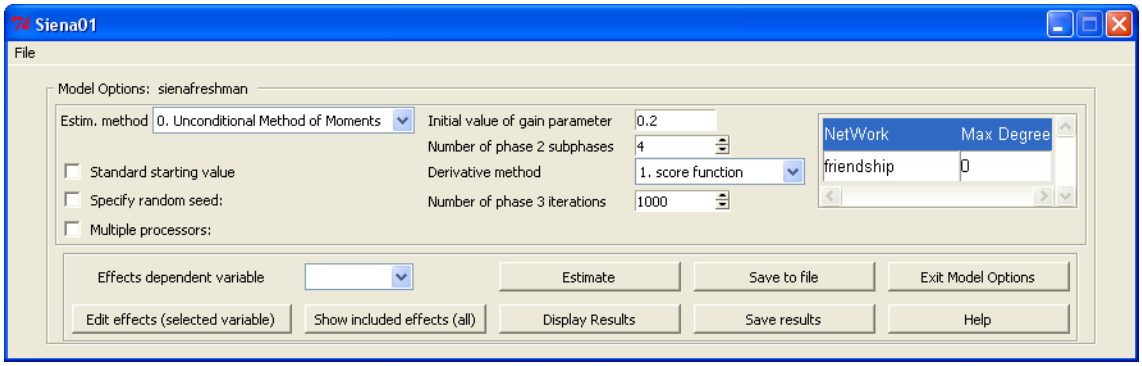


Figure 3: Model options screen

source of error is that the program cannot find your files; e.g., the files are not in the working directory (see above) but in a different directory.

If errors occur at this moment and the options screen does not appear, then you can obtain diagnostic error messages working not through the siena01Gui, but directly within R as described in Section 2.3.1. This will hopefully help you solving this problem; later on you can then work through the siena01Gui again.

2. Select the options you require.

3. Use Edit Effects to choose the effects you wish to include. Note you can edit the effects for just one dependent variable at a time if you wish by selecting one dependent variable in 'Effects dependent variable'.

4. Click Estimate.

5. You should see the SIENA screen of the estimation program.

6. When the program has finished, you should see the results. If not, click Display Results to see the results. The output file which you will see is stored, with extension `.out` in the directory in which you start siena.exe.

7. You may restart your estimation session at a later date using the Continue session from file on the Data Entry Screen.
   The restart needs a saved version of the data, effects and model as R objects. This will be created automatically when you first enter the Model Options Screen, using the default effects and model. You may save the current version at any time using the Save to file button, and will be prompted to do so when you leave this screen.

2.2.5 Details of The Data Entry Screen

Group May be left blank unless you wish to use the multi-group option described in Section 11.1. Should not contain embedded blanks.

Name Network files or dyadic covariates should use the same name for each file of the set. Other files should have unique names, a list of space separated ones for constant covariates.

File Name Usually entered by using a file selection box, after clicking Add.

Format Only relevant for networks or dyadic covariates. Can be a matrix; a single Pajek network (.net) (not for two-mode networks); or a Siena network file (an edgelist, containing three or four columns: (from, to, value, wave (optional)), not yet tested for dyadic covariates!). By specifying the waves in the fourth column in the Siena format, one file can be used to contain data for all the waves.

Period(s) Only relevant for networks and dyadic covariates. All other files cover all the relevant periods. Indicates the order of the network and dyadic covariate files. Should range from 1 to M within each group, where M is the number of time points (waves). Use multiple numbers separated by spaces for multi-wave Siena network files.

ActorSet If you have more than one set of nodes, use this column to indicate which is relevant to each file. Should not contain embedded blanks.

Type Indicate here what type of data the file contains. Options are:

network (i.e., a one-mode network, which is the usual type)

bipartite (i.e., a two-mode network, which is a network with two node sets, and all ties are between the first and the second node set)

behavior

constant covariate

changing covariate

constant dyadic covariate

changing dyadic covariate

exogenous event (for changing composition of the actor set)

Selected Yes or No. Files with Yes *or blank* will be included in the model. Use this field to remove any networks or behavior variables that are not required in the model.

Missing Values Enter any values which indicate missingness, with spaces between different entries.

Nonzero Codes Enter any values which indicate ties, with spaces between different entries.

NbrOfActors For Siena network files, enter the number of actors. For Siena net bipartite files, enter the two dimensions (number of rows, number of columns) of the network, separated by a blank space.

The details of the screen can be saved to a *session* file, from which they can be reloaded. But you can create a session file directly: it should have columns with exactly the same names and in exactly the same order as those of the Data Entry screen, and be of any of the following types:

Extension	Type
`.csv`	Comma separated
`.dat` or `.prn`	Space delimited
`.txt`	Tab delimited

The root name of this input file will also be the root name of the output file.

2.2.6 Continuing the estimation

1. Below you will see some points about how to evaluate the reliability of the results. If the convergence of the algorithm is not quite satisfactory but not extremely poor, then you can continue just by Applying the estimation algorithm again.

2. If the parameter estimates obtained are very poor (not in a reasonable range), then it usually is best to start again, with a simpler model, and from a standardized starting value. The latter option must be selected in the Model Options screen.

2.3 Using SIENA within R

There are two alternatives, depending on your familiarity with R, treated in Sections 2.3.1 and 2.3.2. Section 2.3.3 explains how to make the step from the use of the GUI to the use of R commands in the regular R way.

Section 2.4 presents an example of an R script for getting started with RSiena.

2.3.1 For those who are slightly familiar with R

1. Install R.

2. Install (within R) the package RSiena, and possibly network (required to read Pajek files), snow and rlecuyer (required to use multiple processes).

3. Set the working directory of R appropriately (setwd() within Ror via a desktop shortcut).

4. You can get help by the command

   ```
   help(RSiena)
   ```

   This will open a browser window with help information; by clicking on the 'Index' link in the bottom line of this window, you get a window with all RSiena commands. The command

   ```
   RShowDoc("RSiena_Manual", package="RSiena")
   ```

   opens the official RSiena manual.

5. Create a session file using siena01Gui() within R, or using an external program.

6. Then, within R,

   (a) Use sienaDataCreateFromSession() to create your data objects.

   (b) Use getEffects() to create an effects object.

   (c) Use fix() to edit the effects object and select the required effects, by altering the Include column to TRUE.

(d) Use sienaModelCreate() to create a model object.

(e) Use siena07() to run the estimation procedure.

Basic output will be written to a file. Further output can be obtained by using the verbose=TRUE option of siena07.

2.3.2 For those fully conversant with R

1. Add the package RSiena
2. Get your network data (including dyadic covariates) into matrices, or sparse matrices of type dgTMatrix. spMatrix() (in package Matrix) is useful to create the latter.
3. Covariate data should be in vectors or matrices.
4. All missing data should be set to NA.
5. Create SIENA objects for each network, behavior variable and covariate, using the functions sienaNet() (for both networks and behavior variables), coCovar() etc.
6. Create a SIENA data object using SienaDataCreate().
7. Use getEffects() to create an effects object.
8. Use functions such as includeEffects(), setEffect(), and includeInteraction() to further specify the model.
9. Use sienaModelCreate() to create a model object.
10. Use siena07() to run the estimation procedure.
11. Note that it is possible to use multiple processes in siena07. For details see section 2.6.
12. Also note the availability of the parameter prevAns to reuse estimates and derivatives from a previous run with similar effects.

Basic output will be written to a file. Further output can be obtained by using the verbose=TRUE option of siena07. The scripts on the scripts page of the RSiena website give many examples of how to use RSiena.

2.3.3 The transition from using the graphical user interface to commands

At some moment, for users who started learning the use of RSiena through the GUI, it can be desirable to make the transition to using commands in the regular R way. This will make available more options and integration with other R features. The transition can easily be made as follows.

After having made at least one estimation run in the GUI (which could be with the default model specification, without having made any further additions to the model; but it could also be with a more complicated model), click the button Save to file. You will be

prompted for a file name with extension .RData. Make sure that you do give a non-empty name before the dot in .RData; for the moment, let us choose the name trans.RData.

Then later in an R session you can load the trans.RData. This can be done in Windows by selecting File – Load Workspace from the drop down menu, and entering this file name. It can also be done by entering the command

```
load("trans.RData")
```

This will make available three objects, for data, model, and effects. What is currently in your workspace is shown by the command

```
ls()
```

This will probably show that the loaded objects have the names mydata, mymodel, and myeff. These are the type of objects created by the functions sienaDataCreate, sienaModelCreate, and getEffects, and discussed in the SIENA script in Section 2.4.

Now attach the package RSiena and subsequently ask for the descriptions of these three objects:

```
library(RSiena)
mydata
mymodel
myeff
```

This will give short descriptions and thereby a confirmation of what has been imported from the GUI session to the regular R session. From here on you can continue and work with R commands, as in the script, to further specify the model by working on the effects object, and then estimate the parameters by the siena07 function.

The precise point to continue from here, using the scripts in the next section and also on the RSiena website, is to use script Rscript02SienaVariableFormat.R, starting with section D (*Defining Effects*), part 3 (*Adding/removing effects using includeEffects*). After this, you are advised to continue with script Rscript03SienaRunModel.R.

2.4 An example R script for getting started

The best way to get acquainted with RSiena is perhaps going through some existing scripts. The script below is useful for this purpose. At the 'RSiena scripts' page http://www.stats.ox.ac.uk/~snijders/siena/siena_scripts.htm of the RSiena website it is available divided into smaller pieces. The script is written so as to be useful for novice as well as experienced R users. The 'RSiena scripts' page of the RSiena website also contains some other scripts that may be useful. The appendix of this manual contains a list of RSiena functions which may be consulted in addition to this script.

```
##############################################################################
###
### ---- Rscript01DataFormat.R: a script for the introduction to RSiena -------------
###
###                                version: January 17, 2012
##############################################################################
#
# Rscript01DataFormat.R is followed by
# RScriptSNADescriptives.R, code for descriptive analysis of the data, and
# Rscript02SienaVariableFormat.R, which formats data and specifies the model, and
# Rscript03SienaRunModel.R, which runs the model and estimates parameters
# Rscript04SienaBehaviour.R, which illustrates an example of analysing the
# coevolution of networks and behaviour
#
# The entire model fitting is summarised at the end of RscriptSienaRunModel.R
# (without comments)
#
# This is an R script for getting started with RSiena, written by
# Robin Gauthier, Tom Snijders, Ruth Ripley, Johan Koskinen, and
# Paulina Preciado, with some examples borrowed from Christian Steglich.
# Lines starting with # are not processed by R but treated as comments.
# The script has a lot of explanation of R possibilities that will be
# familiar for readers well acquainted with R, and can be skipped by them.
#
# There are various really easy online introductions to R. See, for example
#
#       http://www.statmethods.net/
#       http://www.burns-stat.com/pages/Tutor/hints_R_begin.html
#       http://data.princeton.edu/R/gettingStarted.html
#       http://www.ats.ucla.edu/stat/R/sk/
#
# You can go to any of these sites to learn the basics of R
# or refresh your knowledge.
# There is a lot of documentation available at
#       http://cran.xl-mirror.nl/other-docs.html
# including some short introductions, handy reference cards,
# and introductions in many languages besides English.
#
#
# Some general points to note:
#
# R is case-sensitive. Be aware of capitalization!
#
# The left-arrow "<-" is very frequently used: it denotes an assignment,
# "a <- b" meaning that object a gets the value b.
# Often b is a complicated expression that has to be evaluated by R,
# and computes a result that then is stored as the object a.
#
# Help within R can be called by typing a question mark and the name of the
# function you need help for. For example
# ?library
# will bring up a file titled "loading and listing of packages".
# Comments are made at the end of commands after #,
# or in lines starting with # telling R to ignore everything beyond it.
```

```
# That is why everything up to now in this file is on lines starting with #.
#
# This session will be using s50 data which are supposed to be
# present in the working directory.
# You can get them from
#     http://www.stats.ox.ac.uk/~snijders/siena/siena_datasets.htm
#
# Any command in R is a function, and ends by parentheses
# that enclose the arguments of the function,
# or enclose nothing if no argument is needed, such as for the function q()
# In general the command syntax for calling R's functions is function(x) where
# function is a saved function and x the name of the object to be operated on.
#
##################### - CALLING THE DATA AND PRELIMINARY MANIPULATIONS - ############

# The library command loads the packages needed during the session.

        library(RSiena)

# Some additional packages are used by RSiena,
# the so-called required packages; these will be loaded automatically.

# You need to have INSTALLED all of them.

?install.packages

# Or click on the tab "Packages", "Install package(s)", then select a CRAN mirror
# (e.g. Bristol if you are in the UK) and finally select from the list
# the package you wish to install.

# Where are you?

getwd()

# By something like
# setwd('C:/SienaTest')
# you can set the directory but note the quotes and forward slash.
# It is also possible to set the directory using the menus if you have them.
# On a windows machine, you can predetermine the working directory
# in the <Properties> of the shortcut to R;
# these are accessible by right-clicking the shortcut.

# If you want to set the working directory to for example
# "C:\Documents and Settings\johan\My Documents\RSiena_course"
# simply copy and paste from windows explorer or type
#     setwd('C:/Documents and Settings/johan/My Documents/RSiena_course')
# or
#     setwd('C:\\Documents and Settings\\johan\\My Documents\\RSiena_course')
#
# but note that "\" has to be changed; both '/' and '\\' work!!!

# What is there?

         list.files()
```

```
# What is available in RSiena?

        ?RSiena

# (these are .htlm HELP PAGES)

# At the bottom of this page, when you click on "Index",
# a list of all the available functions is shouwn in your browser.
# The same list is shown in the gui ('graphical user interface') by requesting

        library(help=RSiena)

# Where is the manual?

        RShowDoc("RSiena_Manual", package="RSiena")

# (Note, however, that it is possible that the Siena website
# at http://www.stats.ox.ac.uk/~snijders/siena/ contains a more recent version.)

# Each data is named (for example below we name it friend.data.w1)
# so that we can call it as an object within R.
# If you read an object straight into R, it will treat it as a
# dataset, or in R terminology a "data frame".
# Here this is not what we want, therefore on reading
# we will immediately convert it to a matrix.
# R will read in many data formats, the command to read them is read.table.
      ?read.table
# In the help file for read.table, look at the section "Value",
# which is there in every help page:
# it indicates the class of the object that is produced by the function.
# For read.table, the value is a data frame; below we see what this is.
#
# If we wished to read a .csv file we would have
# used the read.csv command.
# The pathnames must have forward slashes, or double backslashes.
# If single backslashes are used, one of the error messages will be:
#   1: '\R' is an unrecognized escape in a character string

#------------------------------------------------------------------------

# Quick start(Data assignment).
# Please make sure the s50 data set is in your working directory.
# The data set is on the Siena website ("Data sets" tab) and must be
# unzipped in your working directory.

        friend.data.w1 <- as.matrix(read.table("s50-network1.dat"))
        friend.data.w2 <- as.matrix(read.table("s50-network2.dat"))
        friend.data.w3 <- as.matrix(read.table("s50-network3.dat"))
        drink <- as.matrix(read.table("s50-alcohol.dat"))
        smoke <- as.matrix(read.table("s50-smoke.dat"))

# Explanation of data structures and formats below
```

```
################## - DIFFERENT DATA FORMATS - ########################################
###
### The assignments here involve reading in data to a "data frame"
###  data <- read.table("data.dat")
### reads your text file into a "data frame"; check the class of the object "data"
###        >  class(data)
###        [1] "data.frame"
### If your data is not a ".dat" file, alternative import methods are
### ----- ".csv" ------------------------------------------------------------------
###        data <- read.table("c:/data.csv", header=TRUE,
###                                    sep=",", row.names="iden")
### ---- ".csv" -------------------------------------------------------------------
###        data <- read.csv("data.csv",header=T)
### ---- ".spss" ------------------------------------------------------------------
###        library(foreign)
###        data <- read.spss("c:/data.spss")
### ---- ".dta" -------------------------------------------------------------------
###        library(foreign)
###        data <- read.dta("c:/data.dta")
###
#############################################################################################

################## - FROM DATA FRAME TO MATRIX - ######################################
###
### ---- Data frame ---------------------------------------------------------------
### The data frame is like a spreadsheet of cases by variables,
### where the variables are the columns, and these have the names
###        names( data )
### a spreadsheet view of a data frame is given by the fix() command
###        fix( data )
###
### All changes you make in this spreadsheet will be saved automatically,
### so beware.
###
### As an example create two vectors:
      height <- c( 120, 150, 180, 200, 190, 177, 170, 125, 141, 157 )
       weight <- c( 11, 14, 17, 18, 17, 18, 11, 12, 10, 15 )
### The function c() combines its argument into a vector
### (or into a list, but we are not concerned with that possibility now.).
### These two vectors can be collected as variables in a data frame
      data <- data.frame( height, weight )
### and look at the results
    data
### The columns of a data frame may be extracted using a "$" sign and their names.
### For example:
names( data )
data$height
### Or by "[]" and column number, e.g.
data[1]
   data[  , 1]
   data[ 1,  ]
### If you wish to see the structure of an object, such as data, then request
      str(data)
### Objects often have attributes:
```

```
      attributes(data)

### ---- Matrix ------------------------------------------------------------
### A "matrix" is a numeric object ordered like a matrix with dimensions
### ( dim() ) given by the number of rows and columns, e.g.
dim( friend.data.w1 )

### If you wish to play around with a copy of the matrix, e.g. having the name "data",
### you can make the copy by the command
data <- friend.data.w1
### The earlier object that we created with the name "data" now has been lost.
### Elements if matrices can be accessed by using square brackets.
### For example, the element of "data" in row 2 and column 3 is given by
data[ 2, 3 ]
### the first three rows of a matrix called data are given by
data[ 1:3, ]
### columns 2, 5, and 6 are given by
data[ , c( 2, 5, 6) ]

### ---- Converting data frame to matrix -----------------------------------------
###
### Most classes can be converted to other classes through "as.typeofclass ()", e.g.
### if "data" would be an object with a matrix-like structure,
### then it could be converted to the class "matrix" by the command
      data <- as.matrix( data )

##############################################################################


################## - EXAMPLE FOR ARC LIST - ########################################
###
### From the Siena website you can download the data set arclistdata.dat:
###   http://www.stats.ox.ac.uk/~snijders/siena/arclistdata.dat
### Download this file and save it in your current directory.
ArcList <- read.table( "arclistdata.dat", header=FALSE ) # creates data frame
### Note the capitalization.
### Now ArcList is a data.frame
### (we saw this above in the help file for read.table).
### What are its dimensions?
            dim(ArcList)
### You can get an impression of it by looking at the start of the object:
            head(ArcList)
### This is a data file in arclist format, with columns:
### sender id, receiver id, value of tie, wave.
### Add names to the columns:
names(ArcList) <- c( "sid", "recid", "bff", "wid" )
### The bff ("best friend") variable does not have much variability:
            table(ArcList$bff)
### It may be nice to order the rows by sender, then by receiver, then by wave.
### The tedious way to do this is
            ArcList <- ArcList[ order( ArcList$sid, ArcList$recid, ArcList$wid), ]
### To understand this, you may look at the help page for function order()
### The rows of Arclist are ordered first by ArcList$sid, then by ArcList$recid,
### and then by ArcList$wid;
```

```
### these reordered rows then are put into the object ArcList,
### this overwriting the earlier contents of this object.
### This way is tedious because it is repeated all the time that the names
### sid, recid, wid refer to the ArcList object.
### The less tedious way uses the function "with".
### The with(a, b) function tells R that b must be calculated,
### while the otherwise unknown names refer to data set a:
ArcList <- with( ArcList, ArcList[ order( sid, recid, wid), ] )
### Now suppose we want to create a separate set of records for each wave.
### Select by wave:

SAff.1 <- with(ArcList, ArcList[ wid == 1, ] ) #extracts edges in wave 1
SAff.2 <- with(ArcList, ArcList[ wid == 2, ] ) #extracts edges in wave 2
SAff.3 <- with(ArcList, ArcList[ wid == 3, ] ) #extracts edges in wave 3
n <- 50 # this is the number of nodes which is not provided by the arclist

### and has to be repeated separately for each of the waves
### (you may loop over the waves if you like),

### For transforming a matrix into an adjacency list

## create indicator matrix of non-zero entries of a
        ones <- !friend.data.w1 %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(friend.data.w1)[ones]
        edges[, 2] <- col(friend.data.w1)[ones]
        edges[, 3] <- friend.data.w1[ones]
# if desired, order edge list by senders and then receivers
        edges <- edges[order(edges[, 1], edges[, 2]), ]

### For transforming an arclist into a matrix
## First remove the fourth coulmn indicating the wave, so that we are left
## with sender, receiver and value of the tie, and make it into matrix format
SAff.1.copy <- SAff.1[, 1:3]
SAff.1.copy <- as.matrix(SAff.1.copy)
# create empty adjacency matrix
        adj <- matrix(0, 50, 50)
# put edge values in desired places
        adj[edges[, 1:2]] <- edges[, 3]
### Also see Section 4.1.1 of the Siena manual.


################################################################################
################ - READING IN PAJEK DATA - #####################################
###
### If you have data in Pajek format you can use the package "network" in order
### to convert it to a network object. This example is from ?read.paj
###   require( network )
###
###   par( mfrow = c( 2, 2 ) )
###
###test.net.1 <- read.paj("http://vlado.fmf.uni-lj.si/pub/networks/data/GD/gd98/A98.net" )
```

```
###   plot( test.net.1,main = test.net.1$gal$title )
###   test.net.2 <- read.paj("http://vlado.fmf.uni-lj.si/pub/networks/data/mix/USAir97.net" )
###   plot( test.net.2,main = test.net.2$gal$title )
###
################################################################################

# Before we work with the data, we want to be sure it is correct. A simple way
# to check that our data is a matrix is the command class()

        class( friend.data.w1 )

# to list the properties of an object attributes( friend.data.w1 )
# (different classes have different attributes)

# To check that all the data has been read in, we can use the dim() command.
# The adjacency matrix should have the same dimensions as the original data
# (here, 50 by 50).

        dim(friend.data.w1)
        dim(drink)

# To check the values are correct, including missing values, we can use
# the following commands to tabulate the variables.

        table( friend.data.w1, useNA = 'always' )
        table( friend.data.w2, useNA = 'always' )
        table( friend.data.w3, useNA = 'always' )
        table( drink, useNA = 'always' )
        table( smoke, useNA = 'always' )

# NA is the R code for missing data (Not Available).
# This data set happens to have no missings
# (see the data description on the Siena website).
# If there are any missings, it is necessary to tell R about the missing data codes.
# Let us do as if the missing codes for the friendship network were 6 and 9.
# This leads to the following commands.
# (For new R users: the c() function used here as "c(6,9)" constructs
#  a vector [c for column] consisting of the numbers 6 and 9.
#  This function is used a lot in basic R.)

        friend.data.w1[ friend.data.w1 %in% c(6,9) ] <- NA
        friend.data.w2[ friend.data.w2 %in% c(6,9) ] <- NA
        friend.data.w3[ friend.data.w3 %in% c(6,9) ] <- NA

# Commands for descriptive analysis are in the script: RscriptSNADescriptives.R

############### - SELECTING SUBSETS OF DATA - ###################################

# To select a subset of the data based on an actor variable, say,
# those who have the value 2 or 3 on drinking at time 1
# (the possibilities are endless, but hopefully this will serve as a pattern)

      use <- drink[, 1] %in% c(2, 3)
```

```
# This creates a logical vector which is TRUE for the cases where the condition
# is satisfied. To view or check, display the vectors:

      drink[ , 1 ]
      use

# and the number of selected cases is displayed by

      sum( use )

# or

table( use )

# To have this arrayed more neatly side by side, you can create and display
# a matrix with the desired information,

        aa <- matrix(nrow=50, ncol=2)
        aa[,1] <- drink[,1]
        aa[,2] <- use
        aa
#the first column contains the drink level and the second whether this
#level is 2 or 3 (1) or not (0)

# or as a shorter alternative

aa <- cbind(drink[ , 1 ],use)
aa

# Given this selection, submatrices can be formed in case the analyses
# are to be done for this subset only:

        friend1.data.w1 <- friend.data.w1[ use, use ]
        friend1.data.w2 <- friend.data.w2[ use, use ]
        drink1 <- drink[ use, ]



################################################################################
###
### ---- PROCEED TO RscriptSNADescriptives.R FOR DESCRIPTIVE ANAYLSIS ----------
###
################################################################################
```

```
##############################################################################
###
### ---- RscriptSNADescriptives.R: a script for the introduction to RSiena -------
###
###                                version: January 17, 2012
##############################################################################
#
# Rscript01DataFormat.R is followed by
# RScriptSNADescriptives.R, code for descriptive analysis of the data, and
# Rscript02SienaVariableFormat.R, which formats data and specifies the model, and
# Rscript03SienaRunModel.R, which runs the model and estimates parameters
# Rscript04SienaBehaviour.R, which illustrates an example of analysing the
# coevolution of networks and behaviour
#
# The entire model fitting is summarised at the end of Rscript03SienaRunModel.R
# (without comments)
#
# This is an R script for getting started with RSiena, written by
# Robin Gauthier, Tom Snijders, Ruth Ripley, Johan Koskinen, and
# Paulina Preciado, with some examples borrowed from Christian Steglich.
# Lines starting with # are not processed by R but treated as comments.
# The script has a lot of explanation of R possibilities that will be
# familiar for readers well acquainted with R, and can be skipped by them.

# A visual inspection of the adjacency matrices can sometimes be useful.
# This will, for example, help in highlighting outliers with respect to
# outdegrees or indegrees, if there are any of such outliers.
# This requires package "sna":

        library( network )
        library( sna )
        net1 <- as.network( friend.data.w1 )
        net2 <- as.network( friend.data.w2 )
        net3 <- as.network( friend.data.w3 )
        plot.sociomatrix( net1,drawlab = F, diaglab = F, xlab = 'friendship t1' )
        plot.sociomatrix( net2,drawlab = F, diaglab = F, xlab = 'friendship t2' )
        plot.sociomatrix( net3,drawlab = F, diaglab = F, xlab = 'friendship t3' )

# The class,
class( net1 )
# with attributes
attributes( net1 )
# has special methods associated with it.
# while  plot( friend.data.w1 ) only produces a rather dull plot of
# the first two columns
#       plot( net1, xlab = 'friendship t1' )
# produces a nice sociogram
#
# Some further descriptives you can do for the data are plotting and
# calculating some statistics.

# add the attribute drink to the network object

        net1 %v% "drink" <- drink[ , 1 ]
```

```
# color the nodes by drink

        plot( net1, vertex.col = "drink", xlab = 'friendship t1' )


# Now let's color the nodes by drink and scale the vertex by degree of nodes!
#
# First calculate degree:

        deg <- rowSums( as.matrix( net1 ) )# NB:  rowSums() is defined for class matrix

# have a look at the degree distribution

        table( deg, useNA = 'always' )

# Now do the desired plot:

        plot( net1, vertex.col = "drink", vertex.cex = (deg + 1)/1.5 )

# ---- Plot the three waves of data -------------------------------------------

# Add drink to waves 2 and 3
        net2 %v% "drink" <- drink[ , 2 ]
        net3 %v% "drink" <- drink[ , 3 ]
       deg2 <- rowSums( as.matrix( net2 ) )
       deg3 <- rowSums( as.matrix( net3 ) )

# Create a set of panels ( 1 row by 3 columns, or 3 columns by 1 row)

        par( mfrow = c( 1, 3 ) )

# creating three plots after each other will place them in consecutive panels

        plot( net1, vertex.col = "drink", vertex.cex = (deg + 1)/1.5 )
        plot( net2, vertex.col = "drink", vertex.cex = (deg2 + 1)/1.5 )
        plot( net3, vertex.col = "drink", vertex.cex = (deg3 + 1)/1.5 )

# Each time we make a plot the coordinates move - because always
# the starting values are random. We can also save coordinates
# and use them for later plotting:

        par( mfrow = c( 1, 3 ) )
        coordin <-  plot( net1, vertex.col = "drink", vertex.cex = (deg +1 )/1.5 )
        plot( net2, coord = coordin, vertex.col = "drink", vertex.cex = (deg2 + 1)/1.5 )
        plot( net3, coord = coordin, vertex.col = "drink", vertex.cex = (deg3 + 1) /1.5 )

# To get coordinates based on all three waves: coordin <-  plot( net1 + net2 + net3 )
# For more plotting options, try the gplot function in the "sna" library

        ?gplot
        ?gplot.layout
```

```
# ---- Basic network statistics ------------------------------------------------

# The package "sna" can be used for a variety of descriptions and analyses.
# The following are examples.
# some important graph level statistics

        gden( net1 ) # density
        grecip( net1 ) # proportion of dyads that are symmetric
        grecip( net1, measure = "dyadic.nonnull" ) # reciprocity, ignoring the null dyads
        gtrans( net1 ) # transitivity

# dyad and triad census

        dyad.census( net1 )
        triad.census( net1 )

# out degree distribution (of course for a symmetric network outdegree=indegree)

        outdegree <- degree( net1, cmode = "outdegree" )
        outdegree #outgoing ties of each note

        hist( outdegree )
        quantile( outdegree )

# measures of connectivity and distance

        dist <- geodist(net1, inf.replace = Inf, count.paths = TRUE)
# calculate the geodesic distance (shortest path length) matrix
       dist$gd
# matrix of geodesic distances
      dist$counts
# matrix containing the number of paths at each geodesic between each pair of vertices
        dist
        reach <- reachability( net1 )  # calculate the reachability matrix
        reach
##############################################################################
###
### ---- PROCEED TO Rscript02SienaVariableFormat.R FOR PREPARING DATA FOR RSIENA -
###
##############################################################################

##############################################################################
###
### -- Rscript02SienaVariableFormat.R: a script for the introduction to RSiena --
###
###                               version January 17, 2012
##############################################################################
#
# The introductory script is divided into the following script files:
# Rscript01DataFormat.R, followed by
# RScriptSNADescriptives.R, code for descriptive analysis of the data, and
# Rscript02SienaVariableFormat.R, which formats data and specifies the model, and
# Rscript03SienaRunModel.R, which runs the model and estimates parameters
# Rscript04SienaBehaviour.R, which illustrates an example of analysing the
```

```
# coevolution of networks and behaviour
# Written with contributions by Robin Gauthier, Tom Snijders, Ruth Ripley,
# Johan Koskinen, and Paulina Preciado.
#
# This script, Rscript02SienaVariableFormat.R, sets up the variables for analysis.
# The manipulations in this script requires that you have gone through the
# first part, "CALLING THE DATA AND PRELIMINARY MANIPULATIONS",
# of the script "Rscript01DataFormat.R" beforehand;
# or that you have defined the data and the model by means of the
# siena01Gui() function and have saved and loaded the results,
# as described in Section 2.3.3 of the Siena manual.

#### FORMATING DATA ACCORDING TO THEIR ROLES AS VARIABLES IN A SIENA MODEL #####

# A number of objects need to be created in R, as preparations to letting
# siena07 execute the estimation. This will be indicated by
# A: dependent variables;
# B: explanatory variables;
# C: combination of dependent and explanatory variables;
# D: model specification.

# ---- A. ----------------------------------------------------------------------
# First we have to create objects for the dependent variables.

# sienaNet creates a Siena network object from a matrix or array or list of
# sparse matrix of triples.
# This object will have the role of a dependent variable in the analysis.
# The name of this network object (here: friendship) will be used
# in the output file.

 friendship <- sienaNet(
                     array( c( friend.data.w1, friend.data.w2, friend.data.w3 ),
                     dim = c( 50, 50, 3 ) ) )

# The integers in the dim() here refer to the number of nodes (senders,
# receivers) and the number of waves.
# This object is an array of dimension 50 x 50 x 3, representing
# three adjacency matrices, with a number of attributes.
# Note that this is an object

        class(friendship)

# with specific attributes and methods associated with it.
# You can get the detailed information by requesting

        dim( friendship )
        attributes( friendship )

# If you only are interested in the value of one particular attribute,
# you can request this by, e.g.,

        attributes( friendship )$type

# The entire contents of the object are listed by typing
```

```
#       friendship

# but this gives a lot of output which you may not want,
# hence the # sign in front.

# The function sienaNet can also be used to create a behavior variable object
# with the extra argument type = "behavior".
# (Non-mentioned attributes get the default value, and in this case
# oneMode is the default;  see below.)

# The 'drink' data (created in  RscriptDataFormat.R ) is made available as a
# dependent behavior variable by the function

        drinkingbeh <- sienaNet( drink, type = "behavior" )

# the class, class(drinkingbeh), is still sienaNet

# (NB: only use the variable in ONE role in a given model:
#  behavior variable or changing covariate!)

# The options available for a sienaNet object are displayed when typing

        ?sienaNet

# This shows that next to one-mode (unipartite) and behavior dependent variables,
# also two-mode (bipartite) dependent variables are possible.
# You can infer that oneMode is the default type from the fact
# that it is mentioned first.

# To create bipartite network objects you need two node sets and must create
# the node sets too. The following is an example
# (not really meaningful, just for the syntax):

 bfriendship <- sienaNet(array(c(friend.data.w1, friend.data.w2, friend.data.w3),
                               dim=c(50, 50, 3)),
                               "bipartite", nodeSet=c("senders", "receivers"))
 senders <- sienaNodeSet(50, nodeSetName="senders")
 receivers <- sienaNodeSet(50, nodeSetName="receivers")


# ---- B. ------------------------------------------------------------------
# Second we construct objects for the explanatory (independent) variables.
# From the help request
#       ?sienaDataCreate
# we see that these can be of five kinds:
# coCovar            Constant actor covariates
# varCovar           Time-varying actor covariates
# coDyadCovar        Constant dyadic covariates
# varDyadCovar       Time-varying dyadic covariates
# compositionChange  Composition change indicators

# You can get help about this by the following requests:
#       ?coCovar
```

```
#       ?varCovar
#       ?coDyadCovar
#       ?varDyadCovar
#       ?sienaCompositionChange

# The variables available for this data set all are changing actor covariates.
# For illustrative purposes, we use smoking as observed at the first wave
# as a constant covariate:

        smoke1 <- coCovar( smoke[ , 1 ] )

# This selects the first column of smoke,
# which contains the first wave observations,
# and makes it available as a constant covariate.
# This is the pattern for for evey covariate file, e.g.
#       Attr1 <- coCovar( Covariate1 )
# where Covariate1 is a matrix with dim(Covariate1) equal to n x 1
# Note, if Covariates is a matrix with dim(Covariates) equal to n x p
# you can create constant covariates through
#       Attr1 <- coCovar(Covariates[,1])
#       ...
#       Attrk <- coCovar(Covariates[,k])

# We use the drinking data as a changing covariate.
# The function varCovar creates a changing covariate object from a matrix;
# the name comes from 'varying covariate'.

        alcohol <- varCovar( drink )

# You need at least 3 waves of a varying covariate to use it as varCovar as
# the previous wave is used as a predictor of the next wave.

# The command

        attributes( alcohol )

# will tell you the information that RSiena now has added to the drink data.

# ---- C. ---------------------------------------------------------------------
# We now combine the dependent and independent variables.
# The function sienaDataCreate creates a Siena data object from input networks,
# covariates and composition change objects;
# the objects that earlier were created by sienaNet will have the role
# of dependent variables, and similarly the other roles are predetermined
# by creation by the functions coCovar, varCovar,
# coDyadCovar, varDyadCovar, and sienaCompositionChange.

        mydata <- sienaDataCreate( friendship, smoke1, alcohol )

# You should now understand how the result of this differs from the
# result of
#       mybehdata <- sienaDataCreate( friendship, smoke1, drinkingbeh )

# If you would like to use different names, you could request this as follows:
```

```
#        mydata <- sienaDataCreate( nominations = friendship, smoke1,
#                                   drinking = alcohol )

# For bipartite networks you would have to specify the node sets, e.g.,

        mybidata <- sienaDataCreate(bfriendship,
                                   nodeSets=list(senders, receivers))

# If you wish to use a covariate, the relevant nodeSet must match, e.g.,
        balcohol <- varCovar(drink, nodeSet="senders")
        mybidata <- sienaDataCreate(bfriendship, balcohol,
                                   nodeSets=list(senders, receivers))

# but we will not use this in these scripts.
# This finishes the data specification. Now we have to specify the model.
#

# ---- D. ---------------------------------------------------------------------
################################################################################
###
### ---- DEFINING EFFECTS ------------------------------------------------------
###
################################################################################
# The data set as combined in mydata implies a certain set of effects
# that can be included in the specification of the model.
# The function getEffects creates a dataframe of effects with a number of extra
# properties for use in RSiena:

        myeff <- getEffects( mydata )

# mydata is needed as an argument as the effects depend on the number
# and types of covariates and dependent variables.
# Before we explain the object myeff and how we shall be going to use it,
# we first produce a data description which is available now:

        print01Report( mydata, myeff, modelname = 's50_3_init' )

# This writes a basic report of the data to the file
# s50_3_init.out in the current working directory. Locate and open it!
# Inspecting this is important because it serves as a check and also contains
# a number of descriptives.
# In this description you can see that the third wave data for alcohol are not used.
# This is because changing covariates are assumed to be constant from one wave until
# immediately before the next wave, so that the values for the last wave are ignored.

# Let us now consider the myeff object, which is used to specify the model.
# It is of the class "sienaEffects", and contains the model specification.
# You can inspect the current model specification by simply requesting

       myeff

# For starting, the model specification is just a very limited default
# (including rates of change, outdegree and reciprocity only)
# To make a meaningful analysis, you will need to add to it.
```

```
# The rows of myeff correspond to the effects.
# By requesting

        names( myeff )

# you see the type of information that is stored about the effects,
# i.e., the columns (variables) defined for the effects.
# If desired, more information about these variables can be obtained
# from the help files:
#      ?getEffects

# Among these variables is the effectName.
# The set of available effects can be inspected by requesting their names:

        myeff$effectName

# This gives a lot of names: all effects defined in Section 12 of the manual,
# depending on the variables specified in mydata.
# The "include" column defines whether effects are included in the model.

        myeff$include

# Here the TRUE values correspond to the default model specification which,
# however, is not meant as a serious model, being too limited.
# There are 3 main ways to operate on myeff.
# 1. Changing myeff in spreadsheet form by the function fix();
# 2. Changing myeff directly by operating on its elements;
# 3. Using RSiena functions "includeEffects", "setEffects", etc.
# Which one to use is a matter of personal preference.
# The third way is most in line with the design philosophy of R,
# and allows you to save scripts that also can be used when
# there will be new versions of RSiena.
# Therefore, we suggest that you skip the explanations of options 1 and 2, and
# proceed directly to option 3, ' Adding/removing effects using includeEffects'.

# For identifying your effects you need the "shortName"s,
# which can be read in the manual (section "Mathematical definition of effects"),
# or obtained from the "sink" command below.

# ---- 1. Adding/removing effects using fix() ---------------------------------
# fix calls a data editor internal to R, so we can manually edit the effects.
# This operates the same as in the Gui.

        fix( myeff )

# How to use fix() is presented merely for getting to know what myeff is.
# In practical analysis it is more convenient
# to use routine "includeEffects" instead, as explained below.
# fix() may not be usable if you do not have tcl/tk available!
# Note that the top of the dataframe shows the names of the columns:
# name, effectName, etc.
# You can edit the "include" column by changing the TRUE and FALSE values
# as required; when the editor is closed, the new values are stored.
```

```
# ---- 2. Adding/removing effects by direct manipulation of myeff --------------
# Alternatively we can edit the dataframe directly by using R functions.
# You are advised to skip this part ("2.") at first reading,
# and read it only if you wish to get more understanding
# of the interal strtucture of the effects object.
# The commands below are used to set "include" to TRUE or FALSE,
# as an alternative to using the data editor.
# The "include" column with values TRUE or FALSE will always be
# ocated at the 9th column,
# but transitive triplets will not always be at the 13th row as this depends
# on the number of periods and variables; further, the list of available effects
# may change in future versions.
# Note: These row numbers may not be the current ones, as they depend on the
# list of effects implemented, which is changeable.
# Some examples are the following
# (preceded by # because not proposed to be applied).

        #myeff[13,9] <- TRUE   #transitive triples
        #myeff[17,9] <- TRUE   #3 cycles
        #myeff[19,9] <- TRUE   #transitive ties
        #myeff[29,9] <- TRUE   #indegree popularity (sqrt)
        #myeff[33,9] <- TRUE   #outdegree popularity (sqrt)
        #myeff[35,9] <- TRUE   #indegree based activity (sqrt)
        #myeff[37,9] <- TRUE   #outdegree based activity (sqrt)
        #myeff[46,9] <- TRUE   #indegree-indegree assortativity
        #myeff[69,9] <- TRUE   #alcohol alter
        #myeff[73,9] <- TRUE   #alcohol ego
        #myeff[75,9] <- TRUE   #alcohol similarity
        #myeff[83,9] <- TRUE   #alcohol ego x alcohol alter

# But in other choices of data, the effect numbers will change.
# This is a reason why this is not a convenient method.
# The following methods require more typing the first time,
# but can be re-used much more robustly.
# Several variants are given, so that you can use what suits you best.
# We give a small and meaningful model.

# To understand the R commands, recall that myeff is a matrix,
# i.e., a two-dimensional array,
# and myeff[i,j] refers to row/effect i and its characteristic j.
# A file with the effect names, for easy access to their exact wordings,
# is obtained by the following commands.
# The function sink diverts output to a file;
# the last command sink() directs it to the console again.
# The function cbind combines two columns into a matrix,
# and is used here to get effect names and short names next to each other.

     sink("effectlist.txt")
     cbind( myeff$effectName,myeff$shortName)
     sink()

# Another way to get this information is by the command
```

```
# write.table(format(cbind( myeff$effectName,myeff$shortName)),"effectlist.txt")

# Now look in the file "effectlist.txt" (in the current directory)
# for the spelling of the various effects you might wish to use.
# The following commands can be used to select the
# five mentioned effects.
# Note that == is the logical "equals", & is the logical "and",
# and what is between the square brackets [...] makes a selection
# of the specified row and column of the "myeff" data frame.

        myeff[myeff$effectName=='transitive triplets' &
              myeff$type=='eval', 'include'] <- TRUE
        myeff[myeff$effectName=='3-cycles' &
              myeff$type=='eval', 'include'] <- TRUE
        myeff[myeff$effectName=='smoke1 similarity' &
              myeff$type=='eval', 'include'] <- TRUE
        myeff[myeff$effectName=='alcohol alter' &
              myeff$type=='eval', 'include'] <- TRUE
        myeff[myeff$effectName=='alcohol ego' &
              myeff$type=='eval', 'include'] <- TRUE
        myeff[myeff$effectName=='alcohol ego x alcohol alter' &
              myeff$type=='eval', 'include'] <- TRUE

# You can similarly add other ones.
# If you make a typing error in the effect name, there will be no warning,
# but also no consequence of your command.
# Therefore it is good to check the results,
# by requesting the list of effects now included in the model:

        myeff

# ---- 3. Adding/removing effects using includeEffects -------------------------
# A third way of specifying the model is by the includeEffects function.
# This function uses short names instead of full names.
# The short names are listed in the descriptions given in
# Section 12 of the manual.
# A list of the short names can also be obtained by

        sink("effectshortlist.txt")
        myeff$shortName
        sink()

# This will create a file in your working directory named "effectshortlist.txt"
# A more general table of effect information including short names
# is also available as a pdf file in the R directory, and can be opened by

        RShowDoc("effects", package="RSiena")

# or created and opened as a html file in the current directory by the function

        # effectsDocumentation()

# For illustration, let us start from scratch with a new sienaEffects object,
# and add the transitive triples and 3-cycles effects
```

```
        myeff <- getEffects( mydata )
        myeff <- includeEffects( myeff, transTrip, cycle3 )

# Note that we can set several effects in one go!
# To remove: myeff[ myeff$shortName == '<<shortName>>', ]$include <- FALSE
# e.g. if
#       myeff <- includeEffects( myeff, cycle3 )
# and you want to remove this effect
#       myeff[ myeff$shortName == 'cycle3', ]$include <- FALSE
# Or, more conveniently, switch "include" to FALSE to turn off an effect.
#       myeff <- includeEffects( myeff, cycle3, include = FALSE )

# Check again which effects you have included:

        myeff

# ---- Adding/removing covariate related effects ------------------------------
# The short names do not differentiate between the covariates:
# e.g., the effects 'alcohol ego' and 'smoke1 ego' both have short name 'egoX',
# and the command

        myeff <- includeEffects( myeff, egoX )

# results in a message that does not (like the earlier one)
# confirm the newly included effect.
# The covariates are indicated by the variable "interaction1"
# in the sienaEffects object,
# and this has to be mentioned to include these effects:

        myeff <- includeEffects( myeff, egoX, altX, egoXaltX,
                                 interaction1 = "alcohol" )
        myeff <- includeEffects( myeff, simX, interaction1 = "smoke1" )

# We check the results again:

        myeff

# By looking at the help offered by

        ?includeEffects

# you can see how to include endowment effects and how to exclude effects.
# Effects that depend on other variables, such as egoX, altX, etc. above,
# need the specification of these variables to define them.
# This is done by the interaction1 parameter
# when only one variable name is needed,
# and by interaction2 if there is a second variable involved,
# such as AltsAvAlt (see the manual).


# ---- Creating interactions --------------------------------------------------
# As a special topic, let us show how interaction effects are created.
```

```
# A convenient method to include an interaction is offered by the
# includeInteraction function.
# This can be used to interact two or three effects
# (if the interactions are allowed; see the manual for this).
# The interaction between smoke1 ego and reciprocity, for instance,
# can be defined by the command

        myeff <- includeInteraction( myeff, egoX, recip, include = FALSE,
                                    interaction1 = c("smoke1","") )

# and, e.g., an interaction between smoke1 ego and alcohol ego is defined by

#        myeff <- includeInteraction( myeff, egoX, egoX, include = FALSE,
#                                     interaction1 = c( "smoke1", "alcohol" ) )

# where we now specified "include = FALSE" because we wish to make this effect
# available without using it in in the model to be estimated below.

# Note that the keyword 'interaction1' used by RSiena is used for identifying
# the covariate for which the ego effect is selected, and does not
# refer to the interaction effect itself.
# If at least one of the interacting effects requires the interaction1 parameter
# for it specification, then this parameter is also required for the
# includeInteraction function.
# Here the two or three interaction1 parameters must be combined using c();
# the same goes for interaction2.

# A second special topic is how to access other characteristics of effects
# without referring to the effect numbers.
# This can be done by the setEffect function.
# E.g., the dense triads effects
# counts the number of triplets with at least xx ties,
# where xx is the parameter of the effect, which can be 5 or 6
# (note that 6 is the maximum number of ties in a triplet).
# The default is 5. This is changed to 6 by the command

        myeff <- setEffect(myeff, denseTriads, include = FALSE, parameter = 6)

# The 'parameter' keyword refers to the effect parameter, described in
# Section 12 of the manual.

################################################################################
###
### -- PROCEED TO Rscript03SienaRunModel.R FOR PARAMETER ESTIMATION BY RSIENA --
###
################################################################################

################################################################################
###
###
### ---- Rscript03SienaRunModel.R: a script for the introduction to RSiena -------
###
###                          version January 17, 2012
################################################################################
```

```
#
# The introductory script is divided into the following script files:
# Rscript01DataFormat.R, followed by
# RScriptSNADescriptives.R, code for descriptive analysis of the data, and
# Rscript02SienaVariableFormat.R, which formats data and specifies the model, and
# Rscript03SienaRunModel.R, which runs the model and estimates parameters
# Rscript04SienaBehaviour.R, which illustrates an example of analysing the
# coevolution of networks and behaviour.
# Written with contributions by Robin Gauthier, Tom Snijders, Ruth Ripley,
# Johan Koskinen, and Paulina Preciado.
#
# This script, Rscript03SienaRunModel.R, runs the estimation in RSiena for the
# model set up and defined in the script Rscript02SienaVariableFormat.R.
#
# A quick version of the model fitting without comments is given at the end
# of this script

############################ ESTIMATION OF PARAMETERS ############################

# Parameters of the model are estimated by the function siena07.
# This requires the data specification; the effects specification;
# and a number of parameters, or settings, for the estimation algorithm.
# The latter are contained in an object created by the function sienaModelCreate.
# You can look at the help provided by ?sienaModelCreate
# to find out about options that you may use here;
# for beginning users, only the two options mentioned below are relevant.
#
# Output will be written to a file with name projname.out, where projname is
# whatever name is given; the default (used if no name is given) is Siena.
# This file will be written to your current directory.
# New estimation runs will append to it.
# A new call to print01Report will overwrite it!

        mymodel <- sienaModelCreate(useStdInits = FALSE, projname = 's50_3')

# The useStdInits parameter determines the initial values used for
# the estimation algorithm.
# If useStdInits = TRUE, standard initial values are used;
# if useStdInits = FALSE, the initial values are used that are contained
# in the "initialValue" column of the effects object,
# which were reported above by the information request
#       myeff
# Below we shall see how these initial values can be altered.

# The function siena07 actually fits the specified model to the data
# If you wish the pretty picture of Siena on the screen as information
# about the progress of the algorithm, type

  ans <- siena07( mymodel, data = mydata, effects = myeff)

# (ans for "answer").
# If however you do not want the pretty picture, or if this leads to
# difficulties (which may happen e.g. on a Mac), then type
```

```
#  ans <- siena07(mymodel, data=mydata, effects=myeff, batch=TRUE)

# and intermediate information will be written to the console.

# Function siena07 produces a so-called sienaFit object, here called ans;
# and it fills in a few things in the sienaEffects object myeff,
# if this is the first use of myeff in a siena07 call.
# By using various different effects objects, i.e., with different names,
# you can switch between specifications.

# The batch = FALSE parameters will give a graphical user interface being opened
# which reports on the progress of the estimation algorithm;

# verbose = TRUE leads to extensive diagnostic information being sent
# to the console during the estimation, and results after the estimation
# (these results are also copied to the output file projname.out, mentioned above);
# while batch=TRUE gives only a limited amount of printout sent to the console
# during the estimation (which is seen when clicking in the console,
# or more immediately if the Buffered Output is deselected in the Misc menu)
# which monitors the progress of the estimation algorithm in  a different way.

# The call of siena07 leads to output in the file s50_3.out
# (or more generally projname.out,
# where projname is the name given in sienaModelCreate)
# and to the creation of the object which here is called ans (for "answer").

# To use multiple processors, in the simplest case where your computer has 2
# processors, use

#         ans <- siena07( mymodel, data = mydata, effects = myeff, batch = TRUE,
#                         verbose = TRUE, nbrNodes = 2, useCluster = TRUE)

# Adjust the nbrNodes to the number available.
# If you wish to work on with other programs while running siena07,
# it is advisable to use one node less than the number of available processors.
# If you wish to use other machines as well,
# see the more detailed instructions below.
# You will then need to use the clusterString argument as well.
#
# For more advanced use, it can be helpful to have access to the networks
# simulated in the so-called third phase of the estimation algorithm.
# These networks can be used, e.g., for checking goodness of fit.
# This can be achieved by using the parameter returnDeps=TRUE.
# The fitted object ans will then have a component named "sims"
# which contains a list (each iteration) of lists (each data object)
# of lists (each dependent network or behavior variable) of edgelists for
# networks or vectors for behavior variables.
#
# This option when used with multiple processors would require
# rather a lot of communication between multiple processes,
# slowing down the computations,
# so it might be better to avoid using the two options together.
```

```
################### LOOKING AT THE RESULTS ################################

# The file "s50_3.out" will contain the results of the estimation.
# It is contained in the current directory ("getwd()").
# This file can be read by any text editor.
# A summary of the results is obtained on the screen by

           ans

# and a larger summary by

           summary(ans)

# Depending on the random seed and the model specification,
# the results could be something like the following.

# Estimates, standard errors and t-statistics for convergence
#
#                                        Estimate   Standard   t statistic
#                                                     Error
#
# Rate parameters:
#   0.1      Rate parameter period 1      6.5644  ( 1.1020   )
#   0.2      Rate parameter period 2      5.1981  ( 0.8611   )
#
# Other parameters:
#   1.  eval outdegree (density)         -2.7027  ( 0.1253   ) -0.0025
#   2.  eval reciprocity                  2.3894  ( 0.2157   ) -0.0335
#   3.  eval transitive triplets          0.6111  ( 0.1392   ) -0.0247
#   4.  eval 3-cycles                    -0.0069  ( 0.2731   )  0.0126
#   5.  eval smoke1 similarity            0.2595  ( 0.2107   )  0.0144
#   6.  eval alcohol alter               -0.0195  ( 0.0699   ) -0.0155
#   7.  eval alcohol ego                  0.0374  ( 0.0801   ) -0.0681
#   8.  eval alcohol ego x alcohol alter  0.1297  ( 0.0527   ) -0.0001
#

# The results can also be viewed externally in the output file s50_3.out
# It is advisable that you have a look at all three reports and
# understand how information is organized in each of them.

# To understand the table above, note that the "t statistic"
# is the t-statistic for convergence checking,
# not the t statistic for testing the significance of this effect!
# (See Section 6.2 of the manual.)
# In the external output file, these are called
# "t-ratios for deviations from targets".
# The rule of thumb is that all t-ratios for convergence
# should ideally be less than 0.1 in absolute value;
# this signifies good convergence of the algorithm.
# In the example here, this is the case.
# If this would not be the case, the best thing to do would be
# to continue the estimation, using the estimates produced here,
# and contained in ans, as the new initial values.
# This is explained below.
```

```
# With function siena07 we made ans as the object containing
# all the results of the estimation. For example,

        ans$theta

# contains the vector of parameter estimates while

        ans$covtheta

# contains the covariance matrix of the estimates.
# There are several "methods" available for viewing the object
# containing the results of the estimation.
# Above we already mentioned
#        ans
# and
#        summary( ans )
# The command

        xtable( ans )

# will produce a table formatted for inclusion in a LaTeX document
# or formatted in html. Use e.g.

        xtable( ans, type = 'html' )

# to get html, and e.g.

        xtable( ans, file = 'ff.tex' )

# to write the results to a file.
# At http://cran.r-project.org/web/packages/xtable you can find
# a set of vignettes for the xtable package, the xtable gallery,
# which gives more options.
# A function siena.table is available that is specially designed
# for RSiena results.

############### MORE ON INITIALIZING PARAMETERS FOR ESTIMATION ########

# If the estimation algorithm has not produced good estimates
# (it 'has not converged well'),
# as will be indicated by some of the t-ratios for convergence
# being larger than 0.1 (this threshold is not to be taken too precisely, though),
# the best thing to do is continuing the estimation,
# using the estimates produced here,
# and contained in ans, as the new initial values.
# This is done by the option prevAns ('previous ans') as in

        ans <- siena07(mymodel, data=mydata, effects=myeff, prevAns=ans)

# the parameter estimates in ans then are  extracted and
# used in the new estimation,
# and moreover Phase 1 will be omitted from the algorithm,
# as derivatives and covariance matrix are used from the previous run.
```

```
# This should be used only if the model specification in myeff
# has not changed, and if the provisional parameter estimates obtained
# in ans are reasonable; if they are not reasonable,
# omit the prevAns option, use
#       mymodel$useStdInits <- TRUE
# to get back on track, and return at the next estimation to
#       mymodel$useStdInits <- FALSE
# To understand what happens here, read on:

# Another and more flexible way for determining initial values is by
# using the useStdInits element of the model object,
# and the initial values in the effects object.
# This is done as follows.
# The option useStdInits = TRUE in sienaModelCreate, will make
# each estimation run start with standard initial values.
# The option useStdInits = FALSE makes the estimation start
# with the initial values in the effects object.
# You can switch between these by commands such as

#       mymodel$useStdInits <- FALSE
#       mymodel$useStdInits <- TRUE

# Putting the estimates from the results object ans into the
# effects object myeff is done by
#       myeff <- update(myeff, ans)
# A check that the effects object contains the desired initial values is made by

        myeff

# The initial values are in the vector
        myeff$initialValue[myeff$include]
# and this also can be initialised differently, if this is desired.
# Note that this initial vector will be used until you change it again,
# e.g., to the results of a new run,
# or until you change the useStdInits option.

################################################################################
###
### ---- Testing effects -------------------------------------------------------
###
################################################################################
#
# 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.
# 2. Score-type tests of single and multiple parameters are described
# in the manual.
# Parameters can be restricted by putting TRUE in the
# include, fix and test columns of the effects object.
# For example, to request a score test for the indegree popularity effect,
# the commands can be as follows.
```

```
#        myeff <- setEffect(myeff, inPopSqrt, fix=TRUE, test=TRUE,
#                                          initialValue=0.0)
#        ans <- siena07(mymodel, data=mydata, effects=myeff)

# After such an operation, again request
#        myeff
# to see what you have done.


################################################################################
###
### ---- Time test -------------------------------------------------------------
###
################################################################################
#
# An application of the score test is given for the special case of parameter
# heterogeneity by Lospinoso et al. (2010) and implemented in RSiena.
# To apply the test to the results obtained above. request, e.g.,
#        tt2 <- sienaTimeTest(ans)
#        plot(tt2, effects=3:4)
# If as a consequence of this analysis you wish to add time dummy terms,
# this may be done via
#        myeff <- includeTimeDummy(myeff, transTrip, cycle3, timeDummy="2")
#        ans3 <- siena07(mymodel, data=mydata, effects=myeff, batch=TRUE)
# and testing again,
#        tt3 <- sienaTimeTest(ans3)
# and so on.

################################################################################
###
### ---- Summary of model fitted -----------------------------------------------
###
################################################################################

        friend.data.w1 <- as.matrix(read.table("s50-network1.dat")) # read data
        friend.data.w2 <- as.matrix(read.table("s50-network2.dat"))
        friend.data.w3 <- as.matrix(read.table("s50-network3.dat"))
        drink <- as.matrix(read.table("s50-alcohol.dat"))
        smoke <- as.matrix(read.table("s50-smoke.dat"))

        friend.data.w1[ friend.data.w1 %in% c(6,9) ] <- NA # define missing data
        friend.data.w1[ friend.data.w2 %in% c(6,9) ] <- NA
        friend.data.w1[ friend.data.w3 %in% c(6,9) ] <- NA

        friendship <- sienaNet( array( c( friend.data.w1,
                                          friend.data.w2, friend.data.w3 ),
                                dim = c( 50, 50, 3 ) ) )

        drinkingbeh <- sienaNet( drink, type = "behavior" )
        smoke1 <- coCovar( smoke[ , 1 ] )
        alcohol <- varCovar( drink )

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

```
        myeff <- getEffects( mydata )# create effects structure

        print01Report( mydata, myeff, modelname = 's50_3_init' )

        myeff <- includeEffects( myeff, transTrip, cycle3 )
        myeff <- includeEffects( myeff, egoX, altX,
                                 egoXaltX, interaction1 = "alcohol" )
        myeff <- includeEffects( myeff, simX, interaction1 = "smoke1" )

        mymodel <- sienaModelCreate( useStdInits = TRUE, projname = 's50_3' )
        ans <- siena07( mymodel, data = mydata, effects = myeff)

################################################################################
###
### -- PROCEED TO Rscript04SienaBehaviour.R FOR
###                                 MODELING NETWORKS AND BEHAVIOUR BY RSIENA --
###
################################################################################

################################################################################
###
### ---- Rscript04SienaBehaviour.R: a script for the introduction to RSiena ------
###
###                                     version January 17, 2012
################################################################################
#
# The introductory script is divided into the follwing script files:
# Rscript01DataFormat.R, followed by
# RScriptSNADescriptives.R, code for descriptive analysis of the data, and
# Rscript02SienaVariableFormat.R, that formats data and specifies the model, and
# Rscript03SienaRunModel.R, that runs the model and estimates parameters
# Rscript04SienaBehaviour.R, that illustrates an example of analysing the
# coevolution of networks and behaviour
# Written with contributions by Robin Gauthier, Tom Snijders, Ruth Ripley,
# and Johan Koskinen.
#

# Here is a short script for analysing the co-evolution of the
# friendship network and drinking behaviour for the 50 girls in the
# Teenage Friends and Lifestyle Study data
# (http://www.stats.ox.ac.uk/~snijders/siena/s50_data.zip), described in
# http://www.stats.ox.ac.uk/~snijders/siena/s50_data.htm

# Read in the adjacency matrices, covariates and dependent behavioural variable
# assuming data are in current working directory

        friend.data.w1 <- as.matrix(read.table("s50-network1.dat"))# network
        friend.data.w2 <- as.matrix(read.table("s50-network2.dat"))
        friend.data.w3 <- as.matrix(read.table("s50-network3.dat"))
        drink <- as.matrix(read.table("s50-alcohol.dat"))# behaviour
        smoke <- as.matrix(read.table("s50-smoke.dat"))# covariate

# At this point it is a good idea to use the sna package to plot the networks
# and the behavioural variable. Descriptive measures of the similarity of
```

```
# "friends" with respect to behaviour (like Moran's I) are given by the function
# nacf() in the sna package

# Tell RSiena that the adjacency matrices are network data and in what order
# they should be treated

      friendship <- sienaNet( array( c( friend.data.w1, friend.data.w2,
                                        friend.data.w3 ),
                            dim = c( 50, 50, 3 ) ) )# create dependent variable

# Tell RSiena that the variable "drink" should be treated as a dependent variable

        drinkingbeh <- sienaNet( drink, type = "behavior" )

        smoke1 <- coCovar( smoke[ , 1 ] )

        myCoEvolutionData <- sienaDataCreate( friendship, smoke1, drinkingbeh )

        myCoEvolutionEff <- getEffects( myCoEvolutionData )

# Run reports to check that data is properly formated and
# to get some basic descriptives

        print01Report( myCoEvolutionData, myCoEvolutionEff,
                       modelname = 's50_3_CoEvinit' )

# Define the effects to include in the coevolution model
# Start with some structural effects (use the shortnames that you find in
# RShowDoc("effects", package="RSiena") )

        myCoEvolutionEff <- includeEffects( myCoEvolutionEff, transTrip, cycle3)

# Include a homophily effect for the constant covariate smoking

        myCoEvolutionEff <- includeEffects( myCoEvolutionEff, simX,
                                            interaction1 = "smoke1" )

# If we want to parse out whether there is a selection or influence (or both)
# effect for drinking behaviour,
# we need to first include sender, receiver and homophily effects
# of drinking for friendship formation:

        myCoEvolutionEff <- includeEffects(myCoEvolutionEff, egoX, altX, simX,
                                           interaction1 = "drinkingbeh" )

# For the influence part, i.e. the effect of the network on behaviour,
# we specify the following effects:
# Now indegree, outdegree and assimilation effects for drinking

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

# Check what effects you have decided to include:
```

```
        myCoEvolutionEff

# Define the model as data + effects:

        myCoEvModel <- sienaModelCreate( useStdInits = TRUE,
                                         projname = 's50CoEv_3' )

# Estimate model

        ans <- siena07( myCoEvModel, data = myCoEvolutionData,
                        effects = myCoEvolutionEff)

# THE RESULTS

# To look at the results, type

        ans

# or, somewhat more extensive,

        summary(ans)

# Note that the column 't statistic' gives the t-ratio for convergence,
# not the t statistic for testing that the parameter is 0.
# For good convergence, the t-ratios for convergence
# all should be less than .1 in aboslute value (see the manual).

# For this small data set, the model for behavior dynamics is over-specified,
# leading to some very large standard errors.
# Running a model modified by
#          myCoEvolutionEff <- includeEffects( myCoEvolutionEff,
#                                name = "drinkingbeh", indeg, outdeg,
#                                interaction1 = "friendship" ,include = FALSE)
# without degree effects on behaviour gives better results.
```

2.5 Outline of estimation procedure

SIENA estimates parameters by the following procedure:

1. Certain statistics are chosen that should 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:
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.
(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 `t statistics for deviations from targets`.

2.6 Using multiple processes

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

2. In Phases 1 and 3 the simulations are performed in parallel. In Phase 2, multiple simulations are done with the same parameters, and the resulting statistics are averaged. The gain parameter is increased and the number of iterations in phase 2 reduced to take advantage of the increased accuracy.

3. 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.

4. 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.

5. 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.

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

7. Currently RSiena uses sockets for inter-process communication.

8. 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.

9. 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.

2.7 Steps for looking at results: Executing SIENA.

1. Look at the start of the output file for general data description (degrees, etc.), to check your data input.

2. When parameters have been estimated, first look at the `t ratios for deviations from targets`. These are good if they are all smaller than 0.1 in absolute value, and reasonably good if they are all smaller than 0.2.
   We say that the algorithm has converged if they are all smaller than 0.1 in absolute value, and that it has nearly converged if they are all smaller than 0.2.
   These bounds are indications only, and may be taken with a grain of salt.

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 sienaModelCreate. 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.

4. If all this is of 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 6.4 of this manual.

2.8 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.

Ripley, Ruth M., Snijders, Tom A.B., and Preciado, Paulina. 2011. Manual for SIENA version 4.0 (version January 29, 2012). Oxford: University of Oxford, Department of Statistics; Nuffield College. http://www.stats.ox.ac.uk/siena/

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 (2011) for the score-type goodness of fit tests and Schweinberger and Snijders (2007b) for the calculation of standard errors of the Method of Moments estimators. For assessing and correcting time heterogeneity, and associated model selection considerations, refer to Lospinoso et al. (2011).

A tutorial is Snijders et al. (2010b).

2.9 Getting help with problems

If you have a problem running RSiena, please read through the following hints to see if any of them help. If not, please 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.

For queries about the modelling aspects of SIENA, or interpretation, please continue to use the StOCNET / RSiena mailing list.

Check your version 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. 1.0.9) and an R-forge revision number. You can find both the 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 (use print01Report to get the relevant output file if you are not using the GUI.)

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.

Part II
Users' manual

3 Parts of the program

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

1. input of basic data description,
2. model specification,
3. estimation of parameter values using stochastic simulation,
4. testing parameters and assessing goodness of fit,
5. simulation of the model with given and fixed parameter values.

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 is written to a text file named *pname*.out, where *pname* is the name specified in the call of sienaModelCreate().

4 Input data

SIENA is a program for the statistical analysis of repeated measures of social networks, and requires network data collected at two or more time points. It is possible to include changing actor variables (representing behavior, attitudes, outcomes, etc.) which also develop in a dynamic process, together with the social networks. As repeated measures data on social networks, at the very least, *two or more repeated observations of a network* are required. The number of time points is denoted M.

In addition, various kinds of variables are allowed:

1. *actor-bound* or *individual variables*, also called *actor attributes*, which can be symbolized as v_i for each actor i; these can be constant over time or changing;
   the changing individual variables can be dependent variables (changing dynamically in mutual dependence with the changing network; then they are also called dependent behavior variables) or independent variables (exogenously changing variables; then they are also called individual covariates).

2. *dyadic covariates*, which can be symbolized as w_{ij} for each ordered pair of actors (i, j); these likewise can be constant over time or changing.

Dependent variables (which can be networks or behavior variables) all must be available for the same set of time points.

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.

The data specification in RSiena consists of two steps. First, the role of each variable to be used must be defined using the functions sienaNet, coCovar, varCovar, coDyadCovar, varDyadCovar, or sienaCompositionChange. Second, the variables must be combined into one RSiena data set by the function sienaDataCreate.

4.1 Digraph data

For data specification by the sienaNet function, the network must be specified as a matrix or array or list of sparse matrix of triples. The help file for sienaNet shows by examples how the specification can be given by sparse matrices. Here we only discuss the specification by matrices.

For data specification by the graphical interface siena01Gui 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. The three possible formats for digraph input are as follows.

1. *Adjacency matrices.*
   These can be used in sienaNet and in sienaDataCreateFromSession.
   In the usual case of a one-mode network the adjacency matrix consists of n lines

each with n integer numbers, separated by blanks or tabs, each line ended by a hard return. 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).

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.

4.1.1 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(Matrix(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.

4.1.2 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. Another way (more complicated but it gives possibilities to represent actors entering or leaving at specified moments between observations) is described in Section 4.8.

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.

Structural zeros offer the possibility of analyzing several networks simultaneously under the assumption that the parameters are identical. Another 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 functions 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 (i.e., has the missing value data code for this covariate), 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 structurally fixed at time t_{m+1} at some value $x_{ij}(t_{m+1})$, then in the course of the simulation process from t_m to t_{m+1} this tie variable can be changed as part of the process in the usual way, but after the simulation is over and before the statistics are calculated it will be fixed to the value $x_{ij}(t_{m+1})$.

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

4.2 Dependent behavioral variables

Dependent actor-level variables, also called dependent behavior variables, are changing variables at the actor level, analysed according to the "networks and behavior" actor-oriented model of Snijders et al. (2007); Steglich et al. (2010). This means they change dynamically in mutual dependence with the changing network. These variables also are defined using the sienaNet function, now requiring the `type = behavior` parameter.

4.3 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, $M-1$ covariate matrices are needed.

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

The mean is always subtracted from the covariates. See Section 4.9 on centering.

4.4 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 observation moments, because for 2 observation moments the distinction between constant and changing covariates is not meaningful.

Each changing individual covariate must be specified in a separate call fo 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 4.9 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.

4.5 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 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;

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.

4.6 Dependent action variables

SIENA also allows dependent action variables, also called 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 action variables represent the actors' behavior, attitudes, beliefs, etc. The difference between dependent action variables and changing actor covariates is that the latter 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. In the current implementation only one dependent network variable is allowed, but the number of dependent action variable can be larger than one. Unlike the changing individual covariates, the values of dependent action variables are not assumed to be constant between observations.

Dependent action 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. 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 and the values will be truncated towards zero.

4.7 Missing data

SIENA allows that there are some missing data on network variables, on covariates, and on dependent action variables. Missing data in changing dyadic covariates are not yet implemented. Missing data must be indicated by missing data codes, *not* by blanks in the data set.

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.

The basic idea is the following.

NOTE: This may not be a correct representation. To be modified.

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' is the modal value of the tie variable. In the further observations, for any 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 the same principle is used: if there is a previous observation of the same variable then this value is imputed, if there is none then the observationwise mode of the variable is imputed. Missing covariate data are replaced by the variable's average score at this observation moment. In the course of the simulations, however, the adjusted values of the dependent action 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 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 action 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 target statistics.

4.8 Composition change

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 4.1.2. This is straightforward and not further explained here. This subsection explains the method of Huisman and Snijders (2003), which uses the information about composition change in a slightly more efficient way.

Network composition change, due to actors joining or leaving the network, is handled separately from the treatment of missing data. The digraph 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 lines, 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.

At these times, 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(s). 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 Section 2.1.

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.

4.9 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. For the changing covariates, this is the global mean (averaged over all periods). The values of these subtracted means are reported in the output.

For the dyadic covariates and the similarity variables derived from the individual covariates, the grand mean is calculated and stored by function SienaDataCreate, and 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 exactly the same.) Unlike the 'covariate similarity' effect, the 'same covariate' effect is not centered but keeps its 0-1 values.

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$vCovars$alocohol, "mean")
```

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

```
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.

4.10 Monotone 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, 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.

If a dependent variable is only increasing or only decreasing for all periods, then two of the basic effects defined below 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,

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. 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[7] 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'[8] 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.

- *creation function*
  The creation function[9] distinguishes between new and old network ties (when evaluating possible network changes) and between increasing or decreasing behavioral

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

[8] 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.

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

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.
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 function*
  The endowment function[10] 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: termination of existing ties, decrease of values of the behavior dependent variable.
  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.

5.1.1 Specification in SIENA

The model specification can be defined in SIENA by the functions includeEffects and setEffects. The scripts in Section 2.4 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 example, the command

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

```
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`. For the practical use of SIENA, the shortNames are important; they can be found in Chapter 12.

5.1.2 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

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

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

$$u_i(x^0, x) \;=\; \big(f_i(x) - f_i(x^0)\big) + \Delta^+(x^0, x)\,\big(c_i(x) - c_i(x^0)\big) + \Delta^-(x^0, x)\,\big(e_i(x) - e_i(x^0)\big) \;.$$

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 ($\Delta^-(x^0, x) = 1$).

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).

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 or transitive ties; 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 two effects selected under (2) represent the dynamics in local (triadic) structure; 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.

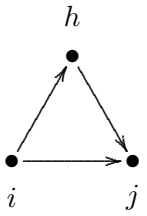
The basic list of these and other effects is as follows.

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

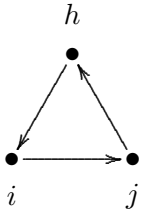


   b. The *balance effect*, which may also be called *structural equivalence with respect to outgoing ties*. This expresses a preference of actors to have ties to those other actors who have a similar set of outgoing ties as themselves. Whereas the transitive triplets effect focuses on how many same choices are made by ego (the focal actor) and alter (the other actor) — the number of h for which $i \to h$ and $j \to h$, 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 $i \to h \to j$ there are, it is only considered whether there is at least one such indirect connection. Thus, one indirect tie suffices for the network embeddedness.

   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.
   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.

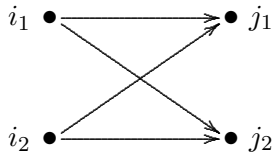
13. The *out-out degree assortativity effect* (with parameters 2 or 1 in similar roles) reflects tendencies for actors with high out-degrees to preferably be tied to other actors with high out-degrees.

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

For the structural part of the model for network dynamics, for two-mode (or bipartite) networks, treated in Koskinen and Edling (2010), 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 sameconnected 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 the are 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 of W on X, representing the extent to which the existence of a tie $i \overset{W}{\rightarrow} j$ promotes the creation or maintenance of a tie $i \overset{X}{\rightarrow} j$.

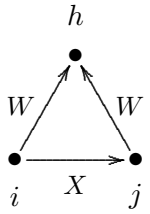
2. If both networks are one-mode, then a next effect is the reciprocity effect with W on X, representing the extent to which the existence of a tie $j \overset{W}{\rightarrow} i$ promotes the creation or maintenance of a tie, in the reverse direction, $i \overset{X}{\rightarrow} j$.

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

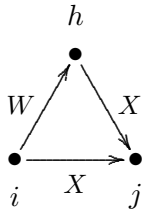
4. The *outdegree W activity effect* (where parameter 2 is the sqrt version, while parameter 1 is the non-sqrt version – see above for explanations of this) reflects the extent to which actors with high outdegrees on W will make more choices in the X network.

⊙ Several mixed transitivity effects can be important.

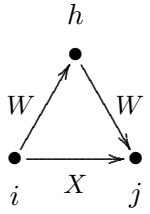
5. If X is a one-mode network, the *from W agreement* effect represents the extent to which agreement between i and j with respect to outgoing W-ties promotes the creation or maintenance of a tie $i \overset{X}{\to} j$.



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

7. 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, and average alter 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.

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. 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 sienaModelCreate. Value `modelType = 1` is for directed networks, values `2-6` for non-directed networks.

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

2. 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.

3. 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.

4. 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.

5. 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 utilities 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` of the effects that are interacting. The interaction effect must also be 'included', but the underlying effects need only be 'included' if they are also required individually.

One way to specify an interaction is to take one of the unspecified interaction effects, work directly on these columns, and set the `include` value to `TRUE`; but it is more convenient to work with includeInteraction, an R function provided to facilitate the definition of interaction effects. Such effects can be specified simply by short names and the names of any variables required to identify the underlying effects: it is not necessary to know the effectNumbers of the effects. (The effectNumbers would change if new effects are introduced to RSiena.) Information about short names of effects can be found in the file 'effects.pdf' in the doc directory of the library, accessible from within R using the command

```
RShowDoc("effects", package="RSiena")
```

Alternatively a new version of this list can be displayed in a browser by using the function:

```
effectsDocumentation()
```

Chapter 12 of this manual also gives the short names of all effects.

5.8.1 Interaction effects for network dynamics

The following kinds of user-defined interactions are possible for the 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 can be inspected by using the fix editor. Another way of seeing the values of the `interactionType` is by requesting, if the network variable is called, e.g., `friendship`, the following:

```
cbind(myeff[myeff$name=="friendship","effectName"],
      myeff[myeff$name=="friendship","shortName"],
      myeff[myeff$name=="friendship","interactionType"])
```

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 either 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, include = FALSE,
                                    interaction1 = c("smoke1","") )
myeff <- includeInteraction( myeff, egoX, egoX, include = FALSE,
                                    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, `""`.

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

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`[11] or empty. You can see the values of the `interactionType` by requesting, if the behavior variable is called, e.g., `drinkingbeh`, the following:

```
cbind(myeff[myeff$name=="drinkingbeh","effectName"],
      myeff[myeff$name=="drinkingbeh","shortName"],
      myeff[myeff$name=="drinkingbeh","interactionType"])
```

The behavioral effects with non-`OK` (i.e., empty) `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

[11] 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.

to the following objective function; where it is assumed that also the linear and quadratic shape effects are included in the model.

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

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

$$f_{ij}^{(m)}(\mathbf{x}) = \sum_k \left(\beta_k + \delta_k^{(m)} h_k^{(m)}\right) s_{ik}\big(\mathbf{x}(i \rightsquigarrow j)\big) \tag{1}$$

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.

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 sienaModelCreate 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).

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 following script gives a walkthrough for this functionality. The interface may change in the future, as this function is still in progress. See the working paper by Lospinoso (2011) for technical information about the test.

```
##############################################################################
## Model selection and RSiena                    #
## Script prepared by Josh Lospinoso (stats.ox.ac.uk/~lospinos)               #
## Date: 5/29/2011                    #
## Version: 4                    #
##############################################################################

# In this script, we will load the s50 data (type ?s50 for more information)
# and go through an iteration of forward model selection. The idea is to specify
# a restricted model containing a conservative number of effects and then
# evaluate evidence for inclusion of more elaborate terms. This approach
# contrasts with backward model selection, which would entail inclusion of a
# large number of terms and subsequent evaluation for omission of terms.

# It is strongly emphasized here that theory should always guide model
# selection! In this script, our intent is to illustrate some of the tools that
# are available to you and not necessarily to provide a "cookie cutter" solution
# to fitting SAOMs to all datasets:
#
# 1) sienaTimeTest
# 2) sienaGOF
# 3) The score type test of Schweinberger

# First, we will install the experimental "RSienaTest" from R-Forge.
```

```
# This version is updated more often than the RSiena package on CRAN.
# Currently, one of the functions (sienaGOF) is not yet available in the
# CRAN version.
install.packages("RSienaTest", repos="http://R-Forge.R-project.org")

# Later, we will use the sna and network packages to show the new GOF
# functionality, as well as a few other packages to help us later on:
install.packages(c("network", "sna"))

# After these packages have been installed, you must tell R to load the
# RSienaTest package:
library(RSienaTest)

# First, give some estimation settings. Here we will use 4 phase 2 subphases
# and 1000 phase 3 iterations. These settings come generally recommended for
# most datasets.
estimationSettings <- sienaModelCreate(fn=simstats0c, nsub=4, n3=1000)

# The s50 dataset is already loaded into the R environment once the
# library(.) function is called. See e.g. ?s501 for information on
# each data object.

# This will load three waves of 50 actors into a sienaNet
friendship <- sienaNet(array(c(s501, s502, s503), dim=c(50, 50, 3)))
s50data <- sienaDataCreate(friendship)

# Here we begin to build up a preliminary model selection. By default, this
# model will have density and reciprocity effects included from getEffects(.)
model1 <- getEffects(s50data)
model1 <- includeEffects(model1,
density,
recip,
transTrip, name="friendship")

# Here, we can add a few score type tests for more elaborate transitive closure
# terms. The choice of which cocktail of closure terms can have profound effects
# on the global structures of the network.
model1 <- setEffect(model1, cycle3,
fix=TRUE, test=TRUE, include=TRUE)
model1 <- setEffect(model1, nbrDist2,
fix=TRUE, test=TRUE, include=TRUE)

# Check what is specified in model1:
model1

# We use the siena07() function to estimate the parameters of model1
# NOTE: you MUST specify returnDeps=TRUE in order to use the goodness of
# fit testing functionality.
( results1 <- siena07(estimationSettings, data=s50data, effects=model1,
batch=TRUE, returnDeps=TRUE) )

# Next, we perform a test for time heterogeneity. This will help us see
# if the parameters have shifted over time. It comes highly
# recommended to perform this step before checking other goodness of fit!
```

```
( timetest1 <- sienaTimeTest(results1) )
# With a p value of about .55, there is very little joint evidence against
# the hypothesis of time homogeneity.

# To get a more informal feel for how time heterogeneity may be present in
# the data, we can plot the test for each effect:
plot(timetest1, effects=1:5)
# Gray bands indicate the base periods (or, alternately, time dummies that
# have been estimated using siena07), and red bands indicate potential time
# dummies. These bands are 95% confidence bands. Where they do not overlap,
# there is (individual) evidence against the time homogeneity hypothesis.

# There is some information on sienaTimeTest available in the help file,
# where you can find the citation for Lospinoso et. al (2010)
# where the test is developed.
?sienaTimeTest

# We can now check the score type test results for more involved
# selection terms. Summary of a siena07 object gives us a wealth
# of information about estimation, but it will also give us a score-type
# test of Schweinberger (2007) calculated for those effects which we have
# denoted fix=TRUE, test=TRUE above:
summary(results1)
# In the section "Generalised score test <c>", we are presented with the
# results. It shows that, with a p value of (<.0001), the actors at distance
# two effect should be included in the next model.

# We have checked time heterogeneity and the goodness of fit of a few
# alcohol-related statistics, but we perform one last check using the
# sienaGOF function. Lospinoso and Snijders (2011) propose to calculate
# auxiliary statistics, i.e. statistics that are important to represent
# with the model but that are not strictly fit by the parameters of the
# model. There is a simple battery of auxiliary statistics that is already
# implemented in RSiena: indegree and outdegree distributions, triad census,
# and average nearest neighboor degree (KNN) distributions. First, let us
# calculate the following using our results:

# - Indegree distribution ( # of nodes with indegree < A for A in 1:MAX)
( id1 <- sienaGOF(results1, IndegreeDistribution, verbose=TRUE) )

# - Outdegree distribution ( # of nodes with outdegree < A for A in 1:MAX)
( od1 <- sienaGOF(results1, OutdegreeDistribution, verbose=TRUE) )

# - Geodesic distribution  ( # of geodesces < A for A in 1:MAX)
( ge1 <- sienaGOF(results1, GeodesicDistribution, verbose=TRUE) )

# We can also write our own custom functions. Here we take all of the transitive
# triplets and intransitive triplets (omitting the vacuously transitive enumerations)
# (See Wasserman and Faust 1994, p. 243):
#
# Transitive:
# 030T a  -> b  -> c, a  -> c
# 120D a  -> b <-> c, a  -> c
# 120U a <-  b <-> c, a <-  c
```

```
# 300  a <-> b <-> c, a <-> c
#
# Intransitive:
# 021C a  -> b  -> c, a     c
# 030C a  -> b  -> c, c  -> a
# 111D a  -> b <-> c, a     c
# 111U a <-  b <-> c, a     c
# 120C a  -> b  -> c, a <-> c
# 201  a <-> b <-> c, a     c
# 210  a <-> b <-> c, a  -> c
#
# The triad.census function is available from the package "sna" (see ?triad.census)
# It will return a vector of length 16 with the following configurations:
#
#1  "003"
#2  "012"
#3  "102"
#4  "021D"
#5  "021U"
#6  "021C"
#7  "111D"
#8  "111U"
#9  "030T"
#10 "030C"
#11 "201"
#12 "120D"
#13 "120U"
#14 "120C"
#15 "210"
#16 "300"
#
# So, we wish to screen out the transitive triplets c(9,12,13,16)
# and the intransitive triplets c(6,10,7,8,14,11,15)
#
CustomTriadCensus <- function(i, data, sims, groupName, varName, wave,
extractor=snaEdgelistExtraction) {
require(sna)
x <- extractor(i, data, sims, groupName, varName, wave)
triad.census(x)[6:16]
}
# Execute the test:
( tc1 <- sienaGOF(results1, CustomTriadCensus, verbose=TRUE) )

# For now, we will not worry about the contents of the text output for these
# objects, but we will seek to get an intuitive feel for how well these features
# are fitting by using the plot.sienaGOF functionality:
plot(id1)
# These plots give a solid red line denoting the observed value of the auxiliary
# statistic (in this case, # of nodes with outdegree < x). The "violin plots"
# show the simulated values of these statistics as both a box plot and a
# kernel density estimate. The dashed gray line represents a 95% confidence
# band for the simulations. Finally, a p value is displayed on the X axis
# label. This p value comes from the Monte Carlo Mahalanobis Distance Test
# of Lospinoso and Snijders (2011) (see ?sienaGOF for more information).
```

```
# The null hypothesis for this p value is that the auxiliary statistics
# for the observed data are distributed according to the statistics shown in
# the plot. For id1, we see that the p value is about .40, so we are left
# with the impression that fit is quite good. (This can be seen visually also!)

# Try outdegree also:
plot(od1)
# We see good on the outdegree also (p is .48).

# We can use the keys given in ?triad.classify of the sna package to help
# us understand where fit is lacking on triad census:
triad.keys <- c("021C","111D","111U","030T","030C","201","120D",
"120U","120C","210","300")
plot(tc1, key = triad.keys)
# The following triads are not being represented well in the model, based
# on a 95% confidence interval cutoff:
# 111U: a <-> b -> c ; a   c  (Too many being simulated)

# And geodesic distance:
plot(ge1)
# The fit is poor (p is less than 0.01)

# The observations are on the bottom of the confidence interval.
# Since there are a finite number of dyads in the graph, this means that the
# simulations are too connected--that is, there are more geodesces
# less than, say, 8 in the simulations than in the observations, so the
# observed geodesces must be more frequent at higher numbers.

# For now, let us follow the suggestsions from the score test of Schweinberger
# and include the actors at distance two effect.
model2 <- setEffect(model1, nbrDist2, fix=FALSE, test=FALSE, include=TRUE)

# And add another candidate for improving triad census (three cycles is still
# on the effects object):
model2 <- setEffect(model2, balance, fix=TRUE, test=TRUE, include=TRUE)

# Estimate our model (note the use of prevAns to speed up convergence):
( results2 <- siena07(estimationSettings, data=s50data, effects=model2,
batch=TRUE, returnDeps=TRUE, prevAns=results1) )

# Next, we perform a test for time heterogeneity:
( timetest2 <- sienaTimeTest(results2) )
# Which still indicates that there is little evidence against the
# time homogeneity assumption.

# Again, we perform sienaGOF on the indegree, outdegree, and geodesic
# distributions as well as triad census:
( id2 <- sienaGOF(results2, IndegreeDistribution, verbose=TRUE) )
( od2 <- sienaGOF(results2, OutdegreeDistribution, verbose=TRUE) )
( tc2 <- sienaGOF(results2, CustomTriadCensus, verbose=TRUE) )
( ge2 <- sienaGOF(results2, GeodesicDistribution, verbose=TRUE) )
# We will treat each of these in turn.

# First, we will look at indegree distribution:
```

```
summary(id2)
# The summary output tells us information first about the Monte Carlo
# Mahalanobis Distance goodness of fit test of Lospinoso and Snijders (2011).
# The p value of .40 tells us that fit is still good on indegree distribution.
# The next item to notice in this summary
# are the outdegree - popularity and outdegree - activity displays.
# These two effects had test=TRUE selected on the effects object, so
# sienaGOF treats these specially. It considers a polynomial approximation
# to the Mahalanobis distance with respect to the parameters of the model
# so that we can posit various specifications and compare their expected
# Mahalanobis distances (for details see Lospinoso and Snijders 2011):
#
# MM(Full) -- For "method of moments  (full)" corresponds to the one step
#  estimates of Schweinberger (2007). This specification allows the most
# drastic differences from the current parameter estimates, and is our
# best guess at what the new parameter estimates would be at the proposed
# specification. Since the one step estimates can be erratic for values
# far away from the current parameters, we also provide some other estimates
# below.
#
# MM(Par.) -- For "method of moments  (partial)" corresponds to an estimate
# where we only take the one step estimate coordinate from MM(full) for
# the effect under consideration. E.g. for "outdegree - popularity",
# all parameters are equivalent to the current estimates, but we replace
# the 0 for outdegree - popularity with its one step estimate.
#
#     GMM -- If MM(Full) and MM(Par.) are not well behaved, there is another
# alternative to consider. GMM, for "generalized method of moments", does
# not depend on the one step estimates at all. It is the parameterization
# for the particular model under consideration that minimizes the quadratic
# approximation for the Mahalanobis distance. It, by definition, will have
# a MHD less than both MM(Full) and MM(Par.).
#
# These estimates can give us some indication of which effects are best to includ
# in our model, if we wish to reduce the distance between the observations and the
# simulations. Since outdegree and indegree distributions already fit quite well,
# Because fit is sufficient for the indegree distribution, we do not need to do
# much more analysis of this output.

# For outdegree,
summary(od2)
# we likewise still have good fit.

# For geodesic distances,
summary(ge2)
# we have achieved quite good fit by simply including actors at distance two.

# For triad census,
summary(tc2)
# where fit is the worst across all of the statistics, outdegree popularity seems
# to be the best candidate, although the magnitude of these estimates cannot be
# trusted (we take the ranking rather than the real value)
plot(tc2, key = triad.keys)
# Here we can use the suggestions of the MM one step estimates. It seems that
```

```
# we can expect a decrease from about 58 to 52 if we include balance:
model3 <- setEffect(model2, balance, fix=FALSE, test=FALSE, include=TRUE)
# While three cycles has not yet had occasion to enter into our model, we continue
# to use it as a candidate, since triad census is not fit well and the three cycles
# is one way to directly fix one of the statistics (030C). Let us compare it
# against dense triads (6), i.e. (300) as we consider model 4:
model3 <- setEffect(model3, denseTriads, fix=TRUE, test=TRUE, include=TRUE)
model3[model3$shortName=="denseTriads","parm"] <- 6

# Estimate our model 3
( results3 <- siena07(estimationSettings, data=s50data, effects=model3,
batch=TRUE, returnDeps=TRUE, prevAns=results2) )
# Convergence could be improved (convergence statistics should be <.1 in abs. val.):
( results3 <- siena07(estimationSettings, data=s50data, effects=model3,
batch=TRUE, returnDeps=TRUE, prevAns=results3) )

# Time test
( tt3 <- sienaTimeTest(results3) )
# All is still well with time homogeneity

# Check goodness of fit again:
( id3 <- sienaGOF(results3, IndegreeDistribution, verbose=TRUE) )
( od3 <- sienaGOF(results3, OutdegreeDistribution, verbose=TRUE) )
( tc3 <- sienaGOF(results3, CustomTriadCensus, verbose=TRUE) )
( ge3 <- sienaGOF(results3, GeodesicDistribution, verbose=TRUE) )

# All of the fits are very good aside from a rather marginal fit for triad census:
plot(tc3, key=triad.keys)
# which have persisted throughout the two model updates we have proposed.
summary(tc3)
# The summary suggests that we consider denseTriads:
model4 <- setEffect(model3, denseTriads, fix=FALSE, test=FALSE, include=TRUE)
model4[model4$shortName=="denseTriads","parm"] <- 6
# We are rapidly saturating the triadic configurations in our model. We take the
# opportunity to try degree related effects as an competing candidate to improve fit:
model4 <- setEffect(model4, inPop, fix=TRUE, test=TRUE, include=TRUE)

( results4 <- siena07(estimationSettings, data=s50data, effects=model4,
batch=TRUE, returnDeps=TRUE, prevAns=results3) )
# Check goodness of fit again:
( id4 <- sienaGOF(results4, IndegreeDistribution, verbose=TRUE) )
( od4 <- sienaGOF(results4, OutdegreeDistribution, verbose=TRUE) )
( tc4 <- sienaGOF(results4, CustomTriadCensus, verbose=TRUE) )
( ge4 <- sienaGOF(results4, GeodesicDistribution, verbose=TRUE) )

# Fit seems adequate across all of the statistics.
#
# Continuing along in this fashion, it is possible to improve fit even more, although
# we again stress that theory should be the principle reason for effect additions.
# This model has very nice fit, and was achieved after a few more iterations.
model5 <- includeEffects(
getEffects(s50data),
balance,
inPop,
```

```
outPop,
cycle3,
denseTriads,
transTrip,
nbrDist2,
name="friendship")

model5[model5$shortName=="denseTriads","parm"] <- 6

( results5 <- siena07(estimationSettings, data=s50data, effects=model5,
batch=TRUE, returnDeps=TRUE, prevAns=results4) )
( id5 <- sienaGOF(results5, IndegreeDistribution, verbose=TRUE) )
( od5 <- sienaGOF(results5, OutdegreeDistribution, verbose=TRUE) )
( tc5 <- sienaGOF(results5, CustomTriadCensus, verbose=TRUE) )
( ge5 <- sienaGOF(results5, GeodesicDistribution, verbose=TRUE) )
```

6 Estimation

The model parameters are estimated under the specification given during the model specification part, using an iterative stochastic approximation algorithm. Three estimation procedures are implemented: the Method of Moments (MoM) (Snijders, 2001; Snijders et al., 2007); 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 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 function siena07 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.

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

```
model1    <-  sienaModelCreate(projname = "trypro", useStdInits = FALSE)
results1  <-  siena07(model1, data = mydata, effects = myeff)
```

The function sienaModelCreate defines a model 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(model1, 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(model1, 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       5.9379  ( 1.0007   )
  1. eval outdegree (density) -2.4742  ( 0.1316   )   0.1328
  2. eval reciprocity          2.0198  ( 0.2505   )  -0.0260
  3. eval transitive triplets  0.5137  ( 0.1726   )  -0.0053
  4. eval 3-cycles             0.1278  ( 0.3462   )   0.0189
  5. eval smoke1 similarity    0.4477  ( 0.2576   )   0.0244
Total of 1399 iteration steps.
```

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.

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(model1, 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 sienaModelCreate, 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*. This is the last column, denoted `t statistic` in the summary given 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). Convergence is excellent when these t-ratios are less than 0.1 in absolute value, reasonable when they are less than 0.2, and moderate when they are less

than 0.3. For published results, it is suggested that estimates presented come from runs in which all t-ratios for convergence are less than 0.1 in absolute value – or nearly so. (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 0.1328, which is reasonable but not as small as desired. The best way to continue then 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(model1, 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 1981 iterations.
Parameter estimates based on 981 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.   0.9140  16.0020   0.0571
  2.   1.0200  14.3691   0.0710
  3.   2.5880  44.3047   0.0584
  4.   1.1270  14.5022   0.0777
  5.   0.1994   4.9865   0.0400

Good convergence is indicated by the t-ratios being close to zero.
```

For example, for the fourth parameter (3-cycles), the average deviation from the target value was 1.1270, and the standard deviation across the 1000 simulations in Phase 3 was 14.5022. This yields a t-ratio of $1.1270/14.5022 = 0.0777$. Large values of the averages and standard deviations are in themselves not at all a reason for concern; only the t-ratio is important.

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.

6.2 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 (which is maintained up to date better than this manual).

`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$pp`	number of parameters
`ans$targets`	targets (observed statistics) for Method of Moments estimation
`ans$tconv`	t-ratios for convergence
`ans$sf`	generated statistics in Phase 3
`ans$msf`	covariance matrix of `ans$sf`
`ans$dfra`	estimated derivative of expected statistics w.r.t. parameters, for Methods of Moments estimation
`ans$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.

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.

To get some further understanding, one could investigate some of the components of this object as follows. Note that the `(... <- ...)` statements are just a way for constructing the object 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))
```

6.3 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 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.

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 sienaModelCreate.

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.

6.4 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. 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 sienaModelCreate().

   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.

6.4.1 Convergence check

This was discussed in Section 6.1.2 above.

6.4.2 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_{i40}. Therefore the estimated evaluation function here is

$$-2.53\, s_{i1}(x) + 2.11\, s_{i2}(x) + 0.54\, s_{i3}(x) + 0.08\, s_{i5}(x) + 0.45\, s_{i40}(x)$$

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.

6.4.3 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.

6.5 Maximum Likelihood and Bayesian estimation

SIENA can estimate models by three estimation methods: the (unconditional or conditional) Method of Moments ('MoM', the default; Snijders, 2001; Snijders et al., 2007), the Maximum Likelihood method ('ML', see Snijders et al., 2010a), and Bayesian methods (see Koskinen, 2004; Koskinen and Snijders, 2007; Schweinberger and Snijders, 2007a). The implementation of maximum likelihood and Bayesian procedures in RSiena still is incomplete. Currently these methods can be used only for complete data sets (i.e., data without any missing values). In nice situations (relatively small and large network data sets, and large network and behavior data sets), the three 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 slightly 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.

For ML and Bayesian estimation, an important parameter for tuning the algorithm is the so-called Multiplication factor. 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.

6.6 Other remarks about the estimation algorithm

6.6.1 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 sienaModelCreate 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 4.8), only the unconditional estimation procedure is available.

6.6.2 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 Jr 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.

6.6.3 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).

6.6.4 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 6.6.1). The solution is either to use different (perhaps standard) initial values or to go over to unconditional estimation.

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 6.3) 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 6.3). 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 sienaModelCreate() that creates the object with specifications for the algorithm.

7.1 Multicollinearity

The estimated standard errors are less reliable in cases with strong multicollinearity, i.e., high correlations[12] between the parameter estimates. 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 6.4.3). Strong collinearity may in practice lead to large differences between the estimated standard errors, and also to considerable differences between the parameter estimates, when comparing the results produced by different estimation runs. The remedy is to reduce the model to a more parsimonious one by excluding non-significant effects of which the parameter estimates are highly correlated with others.

Provisionally (until further experience has been collected), the following may be a reasonable guideline. High parameter correlations with the outdegree effect are not a reason for worry, but high parameter correlations with other effects are a reason for checking the stability of the estimated standard errors. The threshold for finding a parameter correlation 'too high' in this respect can be rather high, such as 0.95, with 0.90 or 0.80 as a secondary threshold. 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.

[12] More precisely, 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.

7.2 Precision of the finite differences method

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

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

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 6.2 we saw that, for a sienaFit object `ans`, the estimates are given in `ans$theta` and the covariance matrix in `ans$covtheta`. 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

$$\frac{\hat{\theta}_k}{\text{s.e.}(\hat{\theta}_k)} = \texttt{ans\$theta[k] /sqrt(ans\$covtheta[k,k])} \quad . \tag{2}$$

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

In some cases, however, the t-statistic (2) 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 Jr 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). This occurs when the data indicates that the parameter should be very large in absolute value, but not how large. The parameter estimate as well as the standard error then are large, and the ratio (2) does not need to be large. The Wald test then cannot be used for significance testing. In Section 6.6.2 we proposed that in such cases, it may be helpful to fix the parameter at some large value, without estimating it. 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.

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

$$A = \begin{pmatrix} 0, 1, 0, 0, 0 \\ 0, 0, 1, 0, 0 \end{pmatrix} .$$

Then the following function 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.

```
Wald.RSiena <- function(A, ans)
{
    if (is.vector(A))
    {
        A  <- matrix(A, nrow=1)
    }
    th     <- A %*% ans$theta
    covmat <- A %*% ans$covtheta %*% t(A)
    chisq  <- drop(t(th) %*% solve(covmat) %*% th)
    d.f.   <- nrow(A)
    pval   <- 1 - pchisq(chisq, d.f.)
    print(c(chisquare = chisq, df = d.f., pvalue = pval), digits=3)
}
```

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 <- function(ans, ...)
   {
    p <- length(ans$theta)
    k <- length(c(...))
    A <- matrix(0, nrow=k, ncol=p)
    A[cbind(1:k,c(...))] <- 1
    Wald.RSiena(A, ans)
    }
```

The test of the preceding example is then produced by the command

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

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 (2011). 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 (2011) 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 (called myeff in Section 2.4). 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=(value to be used for test))
```

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.

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 use the following sequence: 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

$$\frac{\hat{\beta}_a - \hat{\beta}_b}{\sqrt{s.e_a^2 + s.e_b^2}}\ , \tag{3}$$

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 $\mathbf{h}$:

$$h_k^{(m)} = \begin{cases} 1 & \{m : w_m \in \mathcal{W}, m \neq 1\} \\ 0 & \text{elsewhere} \end{cases}, \tag{4}$$

where k corresponds to an effect included in the model.[14] 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 (7):

$$f_{ij}^{(m)}(\mathbf{x}) = \sum_k \left(\beta_k + \delta_k^{(m)} h_k^{(m)}\right) s_{ik}\big(\mathbf{x}(i \rightsquigarrow j)\big) \tag{5}$$

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:

$$\begin{aligned} &H_0 : \delta_k^{(m)} = 0 \text{ for all } k, m \\ &H_1 : \delta_k^{(m)} \neq 0 \text{ for some } k, m. \end{aligned} \tag{6}$$

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):

```
mymodel <- sienaModelCreate(fn=simstats0c, nsub=2, n3=100)
mynet1  <- sienaNet(array(c(s501, s502, s503), dim=c(50, 50, 3)))
mydata  <- sienaDataCreate(mynet1)
myeff   <- getEffects(mydata)
myeff   <- includeEffects(myeff, transTrip, balance)
ans2    <- siena07(mymodel, 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

[14] 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(mymodel, data=mydata, effects=myeff, batch=TRUE)
```

and testing again,

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

and so on.

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 6.3). This is done by requesting `nsub = 0` in the model specification in function sienaModelCreate.

```
sim_model  <-  sienaModelCreate( projname = 'sim_model', cond = FALSE,
                                 useStdInits = FALSE, nsub = 0 )
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]
```

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 6.2, 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( mymodel, 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 4.1.1 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)
```

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 6.6.1. The choice between conditional and unconditional simulation is made in the sienaModelCreate 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 6.6.1), 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. These scripts are also included in this manual in Section 2.4.

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 sienaModelCreate, 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 whether the t-ratios for convergence are small enough (ideally less than .10; less than .15 is acceptable for provisional results). 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.

10.2 Convergence problems

If there are persisting convergence problems even after repeated estimations as suggested in Section 6.1.2, 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.

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.

- 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.)

If the algorithm is unstable, with parameter values (the left hand list in the SIENA window) changing too wildly, or with the algorithm suddenly seeming stuck and not moving forward, the a solution may be to simplify the model (perhaps later on making it more complex again in forward parameter estimation steps); another solution may be to decrease the initial gain parameter (parameter firstg in function sienaModelCreate()).

11 Multilevel network analysis

For combining SIENA results of several independent networks, there are three 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. The first and third options require that the number of observations is the same for the different networks. This is not required for the second option. These methods can be applied for two or more networks.

The three 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 4.1.2. 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. 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.

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.

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. Therefore the arguments usual in statistical modeling apply: as far as assumptions is concerned, option 3 is 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. However, 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 options 1 or 2 are 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. This procedure may work less easily when the number of different networks is relatively high, because it may then lead to too many interactions with dummy variables.

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 observations 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. 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.

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.

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).

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

$$\begin{aligned} \mu_\theta &= \mathcal{E}\,\theta_j\ , \\ \sigma^2_\theta &= \mathrm{var}\,\theta_j\ . \end{aligned}$$

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^2_θ of the distribution of θ, and tests several hypotheses concerning these 'meta-parameters':

1. Test $H_0^{(0)}:\ \mu_\theta = \sigma^2_\theta = 0$ (all $\theta_j = 0$), i.e., effect θ is nil altogether.
   This is done by means of a chi-squared test statistic T^2 with N d.f.

2. Estimate μ_θ.

3. Test $H_0^{(1)}:\ \mu_\theta = 0$.
   This is done by means of a standard normal test statistic t_{μ_θ}, being the ratio of the estimate for μ_θ to its standard error.

4. Test $H_0^{(2)}:\ \sigma^2_\theta = 0$, 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^2_θ .

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 6.6.2.) 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).

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[15] 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 (1999, 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.

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:

1. $p_r > \alpha/2,\ p_l > \alpha/2$:
   No evidence for any nonzero parameter values;

2. $p_r \leq \alpha/2,\ p_l > \alpha/2$:
   Evidence that some networks have a positive parameter value, no evidence for any negative parameter values;

3. $p_r > \alpha/2,\ p_l \leq \alpha/2$:
   Evidence that some networks have a negative parameter value, no evidence for any positive parameter values;

4. $p_r \leq \alpha/2,\ p_l \leq \alpha/2$:
   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.

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

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 larger. 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 $N = 2$, cannot be determined with any degree of precision, so the confidence interval for this mean μ_θ will be huge, and the result for testing the null hypothesis $H_0^{(1)}$ will not be significant. However, the results for testing $H_0^{(0)}$ and $H_0^{(2)}$ will be significant.

12 Mathematical definition of effects

Here, the mathematical formulae for the definition of the effects are given. In Snijders (2001, 2005) and Steglich et al. (2010), further background to these formulae can be found. 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.)

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

$$f_i^{\text{net}}(x) = \sum_k \beta_k^{\text{net}} s_{ik}^{\text{net}}(x) \tag{7}$$

where β_k^{net} are parameters and $s_{ik}^{\text{net}}(x)$ are effects as defined below.

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. 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.

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, $s_{i3}^{\text{net}}(x) = \sum_{j,h} x_{ij}\, x_{ih}\, x_{jh}$;
   and for non-directed networks, $s_{i3}^{\text{net}}(x) = \sum_{j<h} x_{ij}\, x_{ih}\, x_{jh}$;
   there was an error here until version 3.313, which amounted to combining the transitive triplets and transitive mediated triplets effects;

4. *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,
   $s_{i4}^{\text{net}}(x) = \sum_{j,h} x_{ji}\, x_{ih}\, x_{jh}$;
   this cannot be used together with the transitive triplets effect in Method of Moments estimation, because of perfect collinearity of the fit statistics;

5. *number of three-cycles* (`cycle3`),
   $s_{i5}^{\text{net}}(x) = \sum_{j,h} x_{ij}\, x_{jh}\, x_{hi}$;

6. for two-mode networks: the *number of four-cycles* (`cycle4`),
   $s_{i6}^{\text{net}}(x) = \sum_{i_1,i_2,j_1,j_2} x_{i_1 j_1}\, x_{i_1 j_2}\, x_{i_2 j_1}\, x_{i_2 j_2}$;
   for parameter $p = 2$ the square root is taken;

7. *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,
   $s_{i7}^{\text{net}}(x) = \sum_j x_{ij}\, \max_h(x_{ih}\, x_{hj})$;

8. *betweenness count* (`between`),
   $s_{i8}^{\text{net}}(x) = \sum_{j,h} x_{hi}\, x_{ij}\, (1 - x_{hj})$;

9. *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,

$$s_{i9}^{\text{net}}(x) = \sum_{j=1}^{n} x_{ij} \sum_{\substack{h=1 \\ h \neq i,j}}^{n} (b_0 - \mid x_{ih} - x_{jh} \mid) \ ,$$

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.)

10. *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,

$$s_{i10}^{\text{net}}(x) = \sum x_{ij} d_{ij} \tag{8a}$$

with

$$d_{ij} = \sum_{\substack{h=1 \\ h\neq i,j}}^{n} \left(b_0 - \mid x_{hi} - x_{hj} \mid \right) . \tag{8b}$$

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.

11. *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),
$s_{i11}^{\text{net}}(x) = \#\{j \mid x_{ij} = 0, \max_h(x_{ih}\, x_{hj}) > 0\}$;
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.

12. *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,
$s_{i12}^{\text{net}}(x) = \#\{j \mid x_{ij} = 0, \sum_h(x_{ih}\, x_{hj}) \geq 2\}$;
endowment effect only likelihood-based;

13. *number of dense triads* (`denseTriads`), defined as triads containing at least c ties,
$s_{i13}^{\text{net}}(x) = \sum_{j,h} x_{ij}\, I\{x_{ij} + x_{ji} + x_{ih} + x_{hi} + x_{jh} + x_{hj}) \geq c\}$,
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);

14. *number of (unilateral) peripheral relations to dense triads*,
$s^{\text{net}}_{i14}(x) = \sum_{j,h,k} x_{ij}(1-x_{ji})(1-x_{hi})(1-x_{ki})I\{(x_{jh}+x_{hj}+x_{jk}+x_{kj}+x_{hk}+x_{kh}) \geq c\}$,
where c is the same constant as in the *dense triads* effect;
for symmetric networks, the 'unilateral' condition is dropped, and the definition is
$s^{\text{net}}_{i14}(x) = \sum_{j,h,k} x_{ij}(1-x_{hi})(1-x_{ki})I\{(x_{jh}+x_{hj}+x_{jk}+x_{kj}+x_{hk}+x_{kh}) \geq c\}$;

15. *in-degree related popularity effect* (`inPop`) (earlier called *popularity* or *popularity of alter effect*), defined by the sum of the in-degrees of the others to whom i is tied,
$s^{\text{net}}_{i15}(x) = \sum_j x_{ij}\, x_{+j} = \sum_j x_{ij} \sum_h x_{hj}$;
until version 3.313, this effect was multiplied by a factor $1/n$;

16. *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,
$s^{\text{net}}_{i16}(x) = \sum_j x_{ij}\, \sqrt{x_{+j}} = \sum_j x_{ij} \sqrt{\sum_h x_{hj}}$;
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;

17. *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,
$s^{\text{net}}_{i17}(x) = \sum_j x_{ij}\, x_{j+} = \sum_j x_{ij} \sum_h x_{jh}$;
until version 3.313, this effect was multiplied by a factor $1/n$;

18. *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,
$s^{\text{net}}_{i18}(x) = \sum_j x_{ij}\, \sqrt{x_{j+}} = \sum_j x_{ij} \sqrt{\sum_h x_{jh}}$;
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;

⊙ 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;

19. *in-degree related activity effect*, (`inAct`) defined as the cross-product of the actor's in- and out-degrees,
$s^{\text{net}}_{i19}(x) = x_{i+}\, x_{+i}$;
endowment effect only likelihood-based;

20. *in-degree related activity (sqrt) effect*, (`inActSqrt`) defined by
$s^{\text{net}}_{i20}(x) = x_{i+}\, \sqrt{x_{+i}}$;

21. *out-degree related activity effect* (`outAct`), defined as the squared out-degree of the actor, $s^{\text{net}}_{i21}(x) = x^2_{i+}$;
endowment effect only likelihood-based;

22. *out-degree related activity (sqrt) effect* (`outActSqrt`) (earlier called *out-degree^(1.5)*), defined by
$s_{i22}^{\text{net}}(x) = x_{i+}^{1.5} = x_{i+}\sqrt{x_{i+}}$
endowment effect only likelihood-based;

23. *out-degree up to c* (*truncated out-degree*), where c is some constant (internal effect parameter, see above), (`outTrunc`), defined by
$s_{i23}^{\text{net}}(x) = \min(x_{i+}, c)$;

24. *square root out-degree*, defined by
$s_{i24}^{\text{net}}(x) = \sqrt{x_{i+}}$;
this is left out in later versions of SIENA;

25. *squared (out-degree – c)*, where c is some constant, defined by
$s_{i25}^{\text{net}}(x) = (x_{i+} - c)^2$,
where c is chosen to diminish the collinearity between this and the density effect;
this is left out in later versions of SIENA;

26. *sum of (1/(out-degree + c)* (`outInv`), where c is some constant, defined by
$s_{i26}^{\text{net}}(x) = 1/(x_{i+} + c)$;
endowment effect only likelihood-based;

27. *sum of (1/(out-degree + c)(out-degree + c + 1))* (`outSqInv`), where c is some constant, defined by
$s_{i27}^{\text{net}}(x) = 1/(x_{i+} + c)(x_{i+} + c + 1)$;
endowment effect only likelihood-based.

28. *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,
$s_{i28}^{\text{net}}(x) = \sum_j x_{ij}\, x_{i+}^{1/c} x_{j+}^{1/c}$;
c can be 1 or 2 (the latter value is the default);

29. *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,
$s_{i29}^{\text{net}}(x) = \sum_j x_{ij}\, x_{i+}^{1/c} x_{+j}^{1/c}$;
c can be 1 or 2 (the latter value is the default);

30. *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,
$s_{i30}^{\text{net}}(x) = \sum_j x_{ij}\, x_{+i}^{1/c} x_{j+}^{1/c}$;
c can be 1 or 2 (the latter value is the default);

31. *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,
$s^{\text{net}}_{i31}(x) = \sum_j x_{ij}\, x_{+i}^{1/c}\, x_{+j}^{1/c}$;
c can be 1 or 2 (the latter value is the default);

Dyadic covariate effects

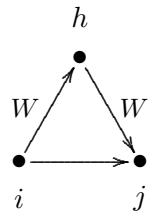
The effects for a dyadic covariate w_{ij} are

32. *covariate (centered) main effect* `(X)`,
$s^{\text{net}}_{i32}(x) = \sum_j x_{ij}\,(w_{ij} - \bar{w})$
where $\bar{w}$ is the mean value of w_{ij} ;

33. *covariate (centered)* $\times$ *reciprocity* `(XRecip)`,
$s^{\text{net}}_{i33}(x) = \sum_j x_{ij} x_{ji}\,(w_{ij} - \bar{w})$.

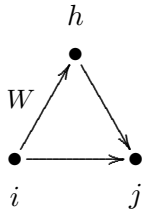
⊙ Three different ways can be modeled in which a triadic combination can be made between the dyadic covariate and the network. In the explanation, the dyadic covariate is regarded as a weighted network (which will be reduced to a non-weighted network if w_{ij} only assumes the values 0 and 1). By way 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.

34. WW => X *closure of covariate* `(WWX)`,
$s^{\text{net}}_{i34}(x) = \sum_{j\neq h} x_{ij}\, w_{ih}\, w_{hj}$;
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.

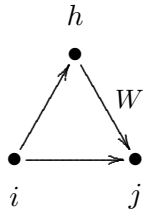


Since the dyadic covariates are represented by square arrays and not by edgelists, this will be a relatively time-consuming effect if the number of nodes is large.

35. WX => X *closure of covariate* `(WXX)`,
$s^{\text{net}}_{i35}(x) = \sum_{j\neq h} x_{ij}\, w_{ih}\, x_{hj}$;
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;



36. *XW => X closure of covariate* (`XWX`),
$s_{i36}^{\text{net}}(x) = \sum_{j \neq h} x_{ij}\, x_{ih}\, w_{hj}$;
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.



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:

37. *covariate-alter* or *covariate-related popularity* (`altX`), defined by the sum of the covariate over all actors to whom i has a tie,
$s_{i37}^{\text{net}}(x) = \sum_j x_{ij}\, v_j$;

38. *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)
$s_{i38}^{\text{net}}(x) = \sum_j x_{ij}\, v_j$;

39. *covariate-ego* or *covariate-related activity* (`egoX`), defined by i's out-degree weighted by his covariate value,
$s_{i39}^{\text{net}}(x) = v_i\, x_{i+}$;

40. *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,
$s_{i40}^{\text{net}}(x) = \sum_j x_{ij}(\text{sim}_{ij}^{v} - \widehat{\text{sim}^{v}})$,
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");

41. *covariate-related similarity* $\times$ *reciprocity* (`simRecipX`), defined by the sum of centered similarity scores for all reciprocal dyads in which i is situated,
$s_{i41}^{\text{net}}(x) = \sum_j x_{ij} x_{ji} (\text{sim}_{ij}^{v} - \widehat{\text{sim}^{v}})$;

42. *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,
$s_{i42}^{\text{net}}(x) = \sum_j x_{ij}\, I\{v_i = v_j\}$,
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;

43. *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, $s^{\text{net}}_{i43}(x) = \sum_j x_{ij}x_{ji}\, I\{v_i = v_j\}$;

44. *covariate-ego* × *alter* (`egoXaltX`), defined by the product of i's covariate and the sum of those of his alters, $s^{\text{net}}_{i44}(x) = v_i \sum_j x_{ij}\, v_j$;

45. *covariate-ego* × *alter* × *reciprocity* (`egoXaltXRecip`), defined by the product of i's covariate and the sum of those of his reciprocated alters, $s^{\text{net}}_{i45}(x) = v_i \sum_j x_{ij}\, x_{ji}\, v_j$;

46. *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, $s^{\text{net}}_{i46}(x) = \sum_j x_{ij}\, \text{dsign}(v_i - v_j)$, where $\text{dsign}(d) = 0$ for $d < 0$, 0.5 for $d = 0$, and 1 for $d > 0$.

47. *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), $s^{\text{net}}_{i47}(x) = \sum_j (1 - x_{ij})\big(\max_h x_{ih}x_{hj}\big)v_j$.

The following group of effects uses an auxiliary variable $\breve{v}_i$ which can be called "alters' v-average". It is described as the average value of v_j for those to whom i is tied, and defined mathematically by

$$\breve{v}_i = \begin{cases} \dfrac{\sum_j x_{ij}v_j}{x_{i+}} & \text{if } x_{i+} > 0 \\ 0 & \text{if } x_{i+} = 0. \end{cases} \tag{9}$$

(Since v is centered, the value of 0 in case $x_{i+} = 0$ is also the mean value of the original variable.)
(It may be noted that this constructed variable $\breve{v}_i$ will not itself have exactly a zero mean generally.)

Note that this value is being updated during the simulations. Network changes will change $\breve{v}_i$; if v_i is a dependent behavior variable, then behaviour changes will also change $\breve{v}_i$.

In the following list, there is no ego effect, because the ego effect of $\breve{v}_i$ would be the same as the alter effect of v_i.

48. *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,

$$s^{\text{net}}_{i48}(x) = \sum_j x_{ij}\breve{v}_j \qquad \text{(parameter 1)}$$

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

$$s_{i48}^{\text{net}}(x) = \sum_j x_{ij} \breve{v}_j^{(-i)} \qquad \text{(parameter 2)}$$

where

$$\breve{v}_j^{(-i)} = \begin{cases} \dfrac{\sum_{h \neq j} x_{jh} v_h}{x_{j+} - x_{ji}} & \text{if } x_{j+} - x_{ji} > 0 \\ 0 & \text{if } x_{j+} - x_{ji} = 0. \end{cases} \tag{10}$$

To compute the contribution for this effect, note that

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

This shows that, given that $\breve{v}_j$ is being updated for all j, the contribution for this effect for parameter 2 can be computed as

$$\frac{x_{j+} \breve{v}_j - x_{ji} v_i}{x_{j+} - x_{ji}}$$

(where 0/0 is interpreted as 0).

49. *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,

$$s_{i49}^{\text{net}}(x) = \sum_j x_{ij} (\text{sim}(\breve{v})_{ij} - \widehat{\text{sim}(\breve{v})}) \ ,$$

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 $\widehat{\text{sim}(\breve{v})}$ is the *observed* mean of all these similarity scores; this observed mean is defined by calculating the $\breve{v}_i$ values for each of the observations t_1 to t_{M-1}, and averaging these
$(M-1)n(n-1)$ (or $(M-1)n(n-1)/2$) similarity values.

12.1.2 Multiple network effects

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 five 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.
THIS VALUE SHOULD BE GIVEN AS THE AVERAGE DEGREE IN THE INITIAL PART OF THE OUTPUT.

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:
   $s_{i4}^{\text{net}}(x) = \sum_j x_{ij}\, (w_{+j} - \bar{w})^{1/p}$;

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:
$s^{\text{net}}_{i5}(x) = \sum_j x_{ij}\,(w_{+i} - \bar{w})^{1/p} = x_{i+}\,(w_{+i} - \bar{w}_{+.})^{1/p}$;

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:
$s^{\text{net}}_{i6}(x) = \sum_j x_{ij}\,(w_{j+} - \bar{w})^{1/p}$;

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:
$s^{\text{net}}_{i7}(x) = \sum_j x_{ij}\,(w_{i+} - \bar{w})^{1/p} = x_{i+}\,(w_{i+} - \bar{w})^{1/p}$;

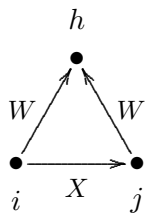
8. *Effect of both in-degrees in W on X-popularity* (X: both indegrees$^{1/p}$ W) `(both)`
defined by the sum of the W-in-degrees of the others to whom i is tied multiplied by the centered W-in-degree of i, for parameter $p = 2$ the square roots of the W-in-degrees:
$s^{\text{net}}_{i8}(x) = \sum_j x_{ij}\,(w_{+i} - \bar{w})^{1/p}\,(w_{+j} - \bar{w})^{1/p}$;
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.

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.

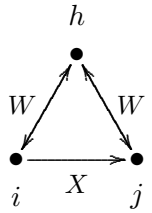
9. *Effect of W-betweenness on X-popularity* (X: betweenness$^{1/p}$ W popularity) `(betweenPop)`
defined by the sum of the W-betweenness counts of the others to whom i is tied:
$s^{\text{net}}_{i9}(x) = \sum_j x_{ij}\,\left(\sum_{h,k;h\neq k} w_{hj}\,w_{jk}\,(1 - w_{hk})\right)^{1/p}$;

Further there are four mixed triadic effects.

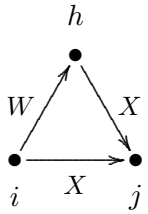
10. *agreement along W leading to X*, (X: from W agreement) `(from)`
$s^{\text{net}}_{i10}(x) = \sum_{j\neq h} x_{ij}\,w_{ih}\,w_{jh}$;
this refers to agreement of actors with respect to their W-choices (structural equivalence with respect to outgoing W-choices); the contribution of the tie $i \overset{X}{\rightarrow} j$ is proportional to the number of joint W choices of others, $i \overset{W}{\rightarrow} h \overset{W}{\leftarrow} j$.



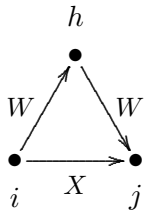
11. *agreement along mutual W-ties leading to X*, (X: from W mutual agreement) `(fromMutual)`
$s^{\text{net}}_{i11}(x) = \sum_{j \neq h} x_{ij}\, w_{ih}\, w_{hi}\, w_{jh}\, w_{hj}$;
this refers to agreement of actors with respect to their mutual W-choices (structural equivalence with respect to mutual W-choices); the contribution of the tie $i \overset{X}{\rightarrow} j$ is proportional to the number of joint mutual W choices of others, $i \overset{W}{\leftrightarrow} h \overset{W}{\leftrightarrow} j$.



12. *W leading to agreement along X*, (X: W to agreement) `(to)`
$s^{\text{net}}_{i12}(x) = \sum_{j \neq h} x_{ij}\, w_{ih}\, x_{hj}$;
this refers to the closure of mixed $W - X$ two-paths; the contribution of the tie $i \overset{X}{\rightarrow} j$ is proportional to the number of mixed $W - X$ two-paths $i \overset{W}{\rightarrow} h \overset{X}{\rightarrow} j$.
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 to whom they have a W-tie.

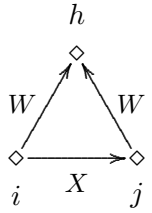


13. *mixed WW => X closure*, (X: closure of W) `(closure)`
$s^{\text{net}}_{i13}(x) = \sum_{j \neq h} x_{ij}\, w_{ih}\, w_{hj}$;
this refers to the closure of $W - W$ two-paths; the contribution of the tie $i \overset{X}{\rightarrow} j$ is proportional to the number of $W - W$ two-paths $i \overset{W}{\rightarrow} h \overset{W}{\rightarrow} j$.
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'.



Then there is a mixed triadic effect restricted to triples with the same value on a covariate.

14. *agreement along W leading to X, for same V*, `(covNetNet)`
$s^{\text{net}}_{i14}(x) = \sum_{j \neq h} x_{ij}\, w_{ih}\, w_{jh}\, I\{v_i = v_j\}$;
this refers to agreement of actors with respect to their W-choices (structural equivalence with respect to outgoing W-choices), but only for actors and choices sharing the same value of a covariate V; the contribution of the tie $i \overset{X}{\rightarrow} j$ is proportional to the number of joint W choices of others, $i \overset{W}{\rightarrow} h \overset{W}{\leftarrow} j$, counting only those for which $v_i = v_j = v_h$.



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* $\breve{v}_i^W$ is defined by

$$\breve{v}_i^W = \begin{cases} \dfrac{\sum_j w_{ij} v_j}{w_{i+}} & \text{if } w_{i+} > 0 \\ 0 & \text{if } w_{i+} = 0. \end{cases} \tag{11}$$

15. *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,

$$s_{i15}^{\text{net}}(x) = \sum_j x_{ij} \breve{v}_j^W \qquad \text{(parameter 1).}$$

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

$$s_{i15}^{\text{net}}(x) = \sum_j x_{ij} \breve{v}_j^{W(-i)} \qquad \text{(parameter 2)}$$

where

$$\breve{v}_j^{W(-i)} = \begin{cases} \dfrac{\sum_{h \neq j} w_{jh} v_h}{w_{j+} - w_{ji}} & \text{if } w_{j+} - w_{ji} > 0 \\ 0 & \text{if } w_{j+} - w_{ji} = 0. \end{cases} \tag{12}$$

16. *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,

$$s_{i16}^{\text{net}}(x) = \sum_j x_{ij} (\text{sim}(\breve{v}^W)_{ij} - \widehat{\text{sim}(\breve{v}^W)}) \ ,$$

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 $\widehat{\text{sim}(\breve{v}^W)}$ is the *observed* mean of all these similarity scores; this observed mean is defined by calculating the $\breve{v}_i^W$ values for each of the observations t_1 to t_{M-1}, and averaging these
$(M-1)n(n-1)$ (or $(M-1)n(n-1)/2$) similarity values.

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

$$c_i^{\text{net}}(x) = \sum_k \zeta_k^{\text{net}} s_{ik}^{\text{net}}(x) \tag{13}$$

for creation of ties. In this formula, the ζ_k^{net} are the parameters for the creation function. The potential effects $s_{ik}^{\text{net}}(x)$ in this function, and their formulae, are the same as in the evaluation function; except that not all are available, as indicated in the preceding subsection. For further explication, consult Snijders (2001, 2005); (here, the 'gratification function' is used rather than the creation function), Snijders et al. (2007), and Steglich et al. (2010) (here only the endowment function is treated and not the creation function, but they are similar in an opposite way).

The *network endowment function* is another way of modeling effects which operate in different strengths for the creation and the dissolution of relations. The network endowment function is zero for creation of ties, and is given by

$$e^{\text{net}}(x) = \sum_k \gamma_k^{\text{net}} s_{ik}^{\text{net}}(x) \tag{14}$$

for dissolution of ties. In this formula, the γ_k^{net} are the parameters for the endowment function. The potential effects $s_{ik}^{\text{net}}(x)$ in this function, and their formulae, are the same as in the evaluation function; except that not all are available, as indicated in the preceding subsection. For further explication, consult Snijders (2001, 2005); (here, the 'gratification function' is used rather than the endowment function), Snijders et al. (2007), and Steglich et al. (2010).

These functions are combined in the following way. For the creation of ties, the objective function used is

$$f_i^{\text{net}}(x) + c_i^{\text{net}}(x) \ , \tag{15}$$

in other words, the parameters for the evaluation and creation effects are added. For the dissolution of ties, on the other hand, the objective function is

$$f_i^{\text{net}}(x) + e_i^{\text{net}}(x) \ , \tag{16}$$

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 $\gamma_k = \beta_k$.

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

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

of factors depending, respectively, on period m, actor covariates, and actor position (see Snijders, 2001, p. 383). The corresponding factors in the rate function are the following:

1. The dependence on the period `(Rate)` can be represented by a simple factor

   $$\lambda_{i1}^{\text{net}} = \rho_m^{\text{net}}$$

   for $m = 1, ..., M-1$. 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.

   Also an exponential dependence on reciprocals of out-degrees `(outRateInv)` can be specified; this can be meaningful because the rate effect of the out-degree becoming a value 1 higher might become smaller and smaller as the out-degree increases. 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

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

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

$$f_i^{\text{beh}}(x, z) = \sum_k \beta_k^{\text{beh}} s_{ik}^{\text{beh}}(x, z) \tag{17}$$

where β_k^{beh} are parameters and $s_{ik}^{\text{beh}}(x, z)$ 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`),
   $s_{i1}^{\text{beh}}(x, z) = z_i$,
   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:
   $s_{i2}^{\text{beh}}(x, z) = z_i^2$.

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[16] to specify the dependent behavior variable in the keyword name as well as the network in the keyword interaction1. For example,

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

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

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

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

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

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

8. *average similarity* × *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,
$s_{i8}^{\text{beh}}(x,z) = x_{i(r)}^{-1} \sum_j x_{ij}x_{ji}(\text{sim}_{ij}^z - \widehat{\text{sim}^z})$;
(and 0 if $x_{i(r)} = 0$) ;

9. *total similarity* × *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,
$s_{i9}^{\text{beh}}(x,z) = \sum_j x_{ij}x_{ji}(\text{sim}_{ij}^z - \widehat{\text{sim}^z})$;

10. *average similarity* × *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,
$s_{i10}^{\text{beh}}(x,z) = x_{i+}^{-1} \sum_j x_{ij}x_{+j}(\text{sim}_{ij}^z - \widehat{\text{sim}^z})$;
(and 0 if $x_{i+} = 0$) ;

11. *total similarity* × *popularity alter effect* (`totSimPopAlt`), 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,
$s_{i11}^{\text{beh}}(x,z) = \sum_j x_{ij}x_{+j}(\text{sim}_{ij}^z - \widehat{\text{sim}^z})$;

12. *average similarity* × *reciprocity* × *popularity alter effect* (`avSimRecPop`), defined by the sum of centered similarity scores sim_{ij}^z between i and the other actors j to whom he is reciprocally tied, multiplied by their indegrees,
$s_{i12}^{\text{beh}}(x,z) = x_{i(r)}^{-1} \sum_j x_{ij}x_{ji}x_{+j}(\text{sim}_{ij}^z - \widehat{\text{sim}^z})$;
(and 0 if $x_{i(r)} = 0$) ;

13. *total similarity* × *reciprocity* × *popularity alter effect* (`totSimRecPop`), defined by the sum of centered similarity scores sim_{ij}^z between i and the other actors j to whom he is reciprocally tied, multiplied by their indegrees,
$s_{i13}^{\text{beh}}(x,z) = \sum_j x_{ij}x_{ji}x_{+j}(\text{sim}_{ij}^z - \widehat{\text{sim}^z})$;

14. *average alter effect* (`avAlt`), defined by the product of i's behavior multiplied by the average behavior of his alters (a kind of ego-alter behavior covariance),
$s^{\text{beh}}_{i14}(x,z) = z_i\big(\sum_j x_{ij}\, z_j\big)/\big(\sum_j x_{ij}\big)$
(and the mean behavior, i.e. 0, if the ratio is 0/0) ;

15. *average reciprocated alter effect* (`avRecAlt`), defined by the product of i's behavior multiplied by the average behavior of his reciprocated alters,
$s^{\text{beh}}_{i15}(x,z) = z_i\big(\sum_j x_{ij}\, x_{ji}\, z_j\big)/\big(\sum_j x_{ij}\, x_{ji}\big)$
(and 0 if the ratio is 0/0) ;

16. *dense triads effect* (`behDenseTriads`), defined by the number of dense triads in which actor i is located,
$s^{\text{beh}}_{i16}(x,z) = z_i \sum_{j,h} I\{x_{ij} + x_{ji} + x_{ih} + x_{hi} + x_{jh} + x_{hj}) \geq c\}$,
where c is either 5 or 6;

17. *peripheral effect*, defined by the number of dense triads to which actor i stands in a unilateral-peripheral relation,
$s^{\text{beh}}_{i17}(x,z) = z_i \sum_{j,h,k} x_{ij}(1-x_{ji})(1-x_{hi})(1-x_{ki}) I\{x_{ij}+x_{ji}+x_{ih}+x_{hi}+x_{jh}+x_{hj}) \geq c\}$,
where c is the same constant as in the *dense triads* effect;
for symmetric networks, the unilateral condition is dropped, and the effect is
$s^{\text{beh}}_{i17}(x,z) = z_i \sum_{j,h,k} x_{ij}(1-x_{hi})(1-x_{ki}) I\{x_{ij} + x_{ji} + x_{ih} + x_{hi} + x_{jh} + x_{hj}) \geq c\}$;

18. *reciprocated degree effect* (`recipDeg`),
$s^{\text{beh}}_{i18}(x,z) = z_i \sum_j x_{ij}\, x_{ji}$;

19. *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,
$s^{\text{beh}}_{i19}(x,z) = x_{+i}\, x_{i+}^{-1} \sum_j x_{ij}(\text{sim}^z_{ij} - \widehat{\text{sim}^z})$;
(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.

Covariate effects

For each actor-dependent covariate v_j (recall that 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.

20. *main covariate effect* (`effFrom`),
$s^{\text{beh}}_{i20}(x,z) = z_i v_i$;
here too, the other dependent behavioral variables are centered so that they have overall mean 0;

21. *alter's covariate average* effect on behavior z (`AltsAvAlt`),
defined as the product of i's behavior z_i and i's alters' covariate-average $\breve{v}_i$ as defined in (9),
$s^{\text{beh}}_{i21}(x, z) = z_i\, \breve{v}_i$.
This is similar to the 'average alter' effect; for $v_i = z_i$ it would reduce to the latter effect.

There are also the following interaction effects between actor covariates (which includes other dependent behavior variables) and influence effects.

22. *interaction of actor variable with average similarity*, (`avSimEgoX`)
$s^{\text{beh}}_{i22}(x, z) = (v_i/x_{i+}) \sum_j x_{ij}(\text{sim}^z_{ij} - \widehat{\text{sim}^z})$;
(and the mean behavior, i.e. 0, if $x_{i+} = 0$) ;

23. *interaction of actor variable with total similarity*, (`totSimEgoX`)
$s^{\text{beh}}_{i23}(x, z) = v_i \sum_j x_{ij}(\text{sim}^z_{ij} - \widehat{\text{sim}^z})$;

24. *interaction of actor variable with average alter*, (`avAltEgoX`)
$s^{\text{beh}}_{i24}(x, z) = v_i\, z_i\big(\sum_j x_{ij}\, z_j\big)/\big(\sum_j x_{ij}\big)$
(and the mean behavior, i.e. 0, if the ratio is 0/0) ;

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

$$s^{\text{beh}}_{i[a*b]}(x, z) \;=\; \frac{1}{z_i}\, s^{\text{beh}}_{ia}(x, z)\, s^{\text{beh}}_{ib}(x, z) \;. \tag{18}$$

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

$$s^{\text{beh}}_{i[a*b*c]}(x, z) \;=\; \frac{1}{z_i^2}\, s^{\text{beh}}_{ia}(x, z)\, s^{\text{beh}}_{ib}(x, z)\, s^{\text{beh}}_{ic}(x, z) \;. \tag{19}$$

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

$$s^{\text{beh}}_{i[v_1*v_2]}(x, z) \;=\; \frac{1}{z_i}\,(z_i\, v_{1i})\,(z_i\, v_{2i}) \;=\; z_i\, v_{1i}\, v_{2i} \;, \tag{20}$$

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

$$\sum_{m=1}^{M-1}\sum_{i=1}^{n} I\{z_i(t_{m+1}) > z_i(t_m)\}\left(s_{ik}^{\text{beh}}(x(t_{m+1})) - s_{ik}^{\text{beh}}(x(t_m))\right), \tag{21}$$

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

$$\sum_{m=1}^{M-1}\sum_{i=1}^{n} I\{z_i(t_{m+1}) < z_i(t_m)\}\left(s_{ik}^{\text{beh}}(x(t_m)) - s_{ik}^{\text{beh}}(x(t_{m+1}))\right), \tag{22}$$

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,

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

for $m = 1, ..., M-1$.

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 (7) 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[17] 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

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

$$s_{ik}^{\text{beh}}(x, z) \;=\; z_i\, s_{ik}^0(x, z) \;, \tag{23}$$

where $s_{ik}^0(x, z)$ is not dependent on z_i, 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 β_k^{beh}, and a change by +1 will increase it by the same amount. Therefore, the log-odds-ratio of an increase in behavior compared to staying constant, due to a difference of 1 in the value of the 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 are not of the form (23). 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

$$\beta_{\text{ego}}\, v_i\, x_{i+} \;+\; \beta_{\text{alter}} \sum_j x_{ij}\, v_j \;+\; \beta_{\text{sim}} \sum_j x_{ij}(\text{sim}_{ij}^v - \widehat{\text{sim}^v}) \;, \tag{24}$$

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 (24) of the tie from i to j, represented by the single tie variable x_{ij} — i.e., the difference between the values of (24)

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 4.9. 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

$$\begin{aligned} &\beta_{\text{ego}}\,(v_i - \bar{v}) \;+\; \beta_{\text{alter}}\,(v_j - \bar{v}) \;+\; \beta_{\text{sim}}\,(\text{sim}_{ij}^{v} - \widehat{\text{sim}^{v}}) \\ &= \beta_{\text{ego}}\,(v_i - \bar{v}) \;+\; \beta_{\text{alter}}\,(v_j - \bar{v}) \;+\; \beta_{\text{sim}}\left(1 - \frac{|v_i - v_j|}{\Delta_V} - \widehat{\text{sim}^{v}}\right) . \end{aligned} \tag{25}$$

From this equation a table can be made that gives the outcome of (25) for some values of v_i and v_j.

This can be concretely carried 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 (25) 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.

$v_i \;\backslash\; v_j$	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

$$\beta_{\text{ego}}\,(v_i{-}\bar{v})\,x_{i+} + \beta_{\text{alter}}\sum_j x_{ij}\,(v_j{-}\bar{v}) + \beta_{\text{sq. alter}}\sum_j x_{ij}\,(v_j{-}\bar{v})^2 + \beta_{\text{e}\times\text{a}}\sum_j x_{ij}\,(v_i{-}\bar{v})\,(v_j{-}\bar{v})\;. \tag{26}$$

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

$$\beta_{\text{ego}}\,(v_i-\bar{v}) \;+\; \beta_{\text{alter}}\,(v_j-\bar{v}) \;+\; \beta_{\text{sq. alter}}\,(v_j-\bar{v})^2 \;+\; \beta_{\text{e}\times\text{a}}\,(v_i-\bar{v})\,(v_j-\bar{v})\;. \tag{27}$$

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*(vi-v_av)*(vi-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 \ 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 \ 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 (27) 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

$$v_j \;=\; \bar{v} \;-\; \frac{\beta_{\text{alter}} \;+\; \beta_{\text{e×a}}\,(v_i - \bar{v})}{2\,\beta_{\text{sq. alter}}} \;. \tag{28}$$

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 (28) 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 (28).

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

[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.

differences even though they are tiny, then they are most attracted to others with $v_j = 2$; 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 4.9.

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

$$f_i^{\mathrm{beh}} \;=\; \beta_{\mathrm{trend}}\,(z_i - \bar{z}) \;+\; \beta_{\mathrm{drink}}\,(z_i - \bar{z})^2 \;+\; \beta_{\mathrm{av.\ sim}}\,\frac{1}{x_{i+}}\sum_j x_{ij}\big(\mathrm{sim}_{ij}^{z} - \widehat{\mathrm{sim}^{z}}\big)\ . \quad (29)$$

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

$$f_i^{\mathrm{beh}} \;=\; \beta_{\mathrm{trend}}\,(z_i - \bar{z}) \;+\; \beta_{\mathrm{drink}}\,(z_i - \bar{z})^2 \;+\; \beta_{\mathrm{av.\ alter}}\,(z_i - \bar{z})(\bar{z}_{(i)} - \bar{z})\ , \quad (30)$$

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

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

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

Equation (30) is simpler than equation (29), because (30) 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 (29) depends on the entire distribution of the Z values of i's friends.

Suppose that, in model (29), 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 (30), 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 (29) 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].

$\bar{z}_{(i)}$ \ 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

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

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 (30) 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 filled; new error messages will be added as questions about them arise.

14.1 During estimation

Unlikely to terminate this epoch: more than 1000000 steps.

> This can happen during conditional estimation in function siena07, when the rate parameter provisionally has hit a value such that the desired number of changes will never be reached. See Section 6.6.1.
> 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 6.6.1).

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

```
sienaModelCreate
```

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 first one to look at is `RSienaDeveloper.tex`. This will guide you to the further documentation.

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/.
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.

1. Work out the definition of the effect and the contribution or change statistic. For network effects, the change statistic is

$$\Delta_{kij}(x) = s_{ki}(x^{+ij}) - s_{ki}(x^{-ij}) \tag{31}$$

   where x^{+ij} is the network with the tie $i \to j$ and x^{-ij} is the network without this tie.
   Determine an existing effect that is most similar to this effect (or perhaps more than one).

2. Open the file allEffects.csv located in "RSiena\data"

   - Identify the effect group (e.g. nonSymmetricObjective, bipartiteSymmetricObjective) where this effect belongs. The names of the groups should be self-explanatory.
   - Insert a new row in that group. Copy a row that corresponds best to your new effect and modify effectName, functionName, shortName, and more if this seems necessary. 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.

3. 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++.

4. 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

$$s_i^{\text{net}}(x) = \min(x_{i+}\,,\, c) \tag{32}$$

where c is an internal effect parameter.

1. The change statistic (31) is

$$\Delta_{ij}(x) \;=\; \begin{cases} 1 & \text{if } \{x_{i+} < c, x_{ij} = 0\} \text{ or } \{x_{i+} \le c, x_{ij} = 1\} \\ 0 & \text{else.} \end{cases} \tag{33}$$

   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

$$s_i^{\text{net}}(x) = \sum_j x_{ij}\, r_{ij}(x) \ . \tag{34}$$

The latter defines the effect through a calculateContribution function and an egoStatistic function, which is the effect as defined in (32).

Since our new effect cannot be expressed in a straightforward way by an equation of the type (34), 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 (32).

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 'outdegreeactivitysqrt-effect' 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 (33), 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 (33), 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   <- sienaNet(array(c(s501, s502), dim=c(50, 50, 2)))
mydata  <- sienaDataCreate(mynet)
mymodel <- sienaModelCreate(projname="s50_12")
myeff   <- getEffects(mydata)
myeff   <- setEffect(myeff, outTrunc, parameter = 3)
ans     <- siena07(mymodel, 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.

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 1 and 2 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 1: Useful functions from R in execution order

* Also available via a menu option

Table 2: 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 2.2.3 and 2.2.4.
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	sienaNet	sienaNet (netarray, type= c("oneMode", "bipartite", "behavior"), nodeSet="Actors", sparse=is.list (netarray))	sienaNet(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 2 – 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 2 – 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 "sienaNet", "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 2 – 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 2 – 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 2 – continued from previous page

Stage	Name	Syntax	Examples	Description
5	print01Report	print01Report(data, myeff, modelname="Siena", session=NULL, getDocumentation=FALSE)	print01Report(MyData, MyEff)	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 2 – continued from previous page

Stage	Name	Syntax	Examples	Description
5	sienaModelCreate	sienaModelCreate(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)	MyModel <– model.create (projname = "MyProject")	Creates a siena model 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 sienaModelCreate

Continued...

Table 2 – 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 model 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 2 – 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 begins at end October 2009, and only details changes which affect the user. (Programmers should consult the changeLog file on CRAN or in the R-forge repository.)

- 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, inlcude effect1-3 etc on diaplay 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.

Butts, C. (2008). Social network analysis with `sna`. *Journal of Statistical Software*, 24(6).

Cochran, W. (1954). The combination of estimates from different experiments. *Biometrics*, 10:101–129.

de Federico de la Rúa, A. (2004). L'analyse longitudinale de réseaux sociaux totaux avec siena - méthode, discussion et application. *Bulletin de Méthodologie Sociologique*, 84:5–39.

de Federico de la Rúa, A. (2005). El análisis dinámico de redes sociales con siena. método, discusión y aplicación. *Empiria*, 10:151–181.

Fisher, R. A. (1932). *Statistical Methods for Research Workers*. Oliver & Boyd, 4th edition.

Frank, O. (1991). Statistical analysis of change in networks. *Statistica Neerlandica*, 45:283–293.

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.

Hauck Jr, W. W. 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.

Huisman, M. and Snijders, T. A. B. (2003). Statistical analysis of longitudinal network data with changing composition. *Sociological Methods & Research*, 32:253–287.

Huisman, M. and Steglich, C. (2008). Treatment of non-response in longitudinal network data. *Social Networks*, 30:297–308.

Jariego, I. M. and de Federico de la Rúa, A. (2006). El análisis dinámico de redes con siena. In Molina, J. L., Quiroga, A., Martí, J., Jariego, I., and de Federico, A., editors, *Talleres de autoformación con programas informáticos de análisis de redes sociales*, pages 77–93. Bellaterra: Universität Autonoma de Barcelona.

Koskinen, J. and Edling, C. (2010). Modelling the evolution of a bipartite network–peer referral in interlocking directorates. *Social Networks*, In Press, Corrected Proof. http://dx.doi.org/10.1016/j.socnet.2010.03.001.

Koskinen, J. H. (2004). *Essays on Bayesian Inference for Social Networks*. PhD thesis, Department of Statistics, Stockholm University.

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.

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.

Lospinoso, J. A. (2011). Goodness of fit for stochastic actor oriented models. *Working paper*.

Lospinoso, J. A., Schweinberger, M., Snijders, T. A. B., and Ripley, R. (2011). Assessing

and accounting for time heterogeneity in stochastic actor oriented models. *Advances in Data Analysis and Computation*, 5:147–176.

Pearson, M. 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.

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.

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(3):400–407.

Robins, G. and Alexander, M. (2004). Small worlds among interlocking directors: network structure and distance in bipartite graphs. *Computational & Mathematical Organization Theory*, 10:69–94.

Schweinberger, M. (2011). Statistical modeling of network panel data: Goodness-of-fit. *British Journal of Statistical and Mathematical Psychology*, page in press.

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. (2001). The statistical evaluation of social network dynamics. In Sobel, M. E. and Becker, M. P., editors, *Sociological Methodology – 2001*, volume 31, pages 361–395. Boston and London: Basil Blackwell.

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. 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. (1999). *Multilevel Analysis: An introduction to basic and advanced multilevel modeling*. London: Sage.

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., 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. E. G. (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. Berlin: Springer.

Steglich, C. E. G., Snijders, T. A. B., and Pearson, M. (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.

van Duijn, M. A. J., Zeggelink, E. P. H., Huisman, M., Stokman, F. N., and Wasseur, F. W. (2003). Evolution of sociology freshmen into a friendship network. *Journal of Mathematical Sociology*, 27:153–191.

Viechtbauer, W. (2010). Conducting meta-analyses in r with the metafor package. *Journal of Statistical Software*, 36(3):1–48.