# Effects in RSiena: The R side

*Materials for an online workshop*

Tom A.B. Snijders

University of Oxford
University of Groningen


November, 2024

These slides are meant to give an explanation of effects in RSiena, allowing you to construct effects yourself.

(The manual (Chapter 18) also contains a tutorial about creating new effects.)

# Overview of constructing effects

To construct a new effect, you will have to do the following things.

1. Choose an `effectName` and a `shortName`.
2. Define the effect statistic and the change statistic.
3. Insert a new line in the code in `\data\allEffects.csv`.
4. Construct a new effect class in `\src\model\effects`,
   or add to an existing effect class.
5. Attach the new effect by appropriate lines in the `EffectFactory`.
6. If you constructed a new effect class,
   insert the name of the `cpp` file in `\src\sources.list`
   and the name of the `h` file in `\src\model\effects\allEffects.h`
7. Check that the effect runs properly, similarly as what was done in
   `https://github.com/stocnet/rsiena/blob/main/checkEffects.R`.

Points 1–3 and 7 are explained in these slides.

# Effects in RSiena

The effects object in RSiena defines the model specification.
It is created, for a given data set, by the R function **`getEffects`**.

It is a data.frame of class **`data.frame`** and **`sienaEffects`**.

The effects object contains, as rows,
all effects that are possible for this data set.

For an effects object `effs`, the command
`effectsDocumentation(effs)`
gives all effects for data set `effs`.

The columns of the effects object
(the names of which, for an effects object `effs`, you can see from `names(effs)`)
define the effect in question (e.g., `shortName`),
how it can be handled by the software (e.g., `interactionType`),
or how it is shown in R (e.g., `effectName`);
furthermore, some columns can be changed by the user
for the model specification, (e.g., `include` and `fix`)
by functions such as **`includeEffects`** and **`setEffect`**.

Data variables in RSiena can be

- dependent directed networks `(oneMode)`
- dependent non-directed networks `(symmetric)`
- dependent two-mode networks `(bipartite)`
- dependent ordinal behavioral variables `(behavior)`
- dependent continuous behavioral variables `(continuous)`
- actor covariates `(coCovar, varCovar)`
- dyadic covariates `(coDyadCovar, varDyadCovar)`

(this leaves out variables defining composition change).

Effects are determined by their `shortName` and
the combination of data variables:
there should be one dependent variable,
and 0, 1, or 2 variables in an explanatory role;
this may be covariates or dependent variables.

In the effects object, the column `name` is the dependent variable;
the columns `interaction1` and `interaction2` are
the explanatory variables (if any).

Whether an effect for a combination of data variables is possible
depends on the length of the variables and the orders of the network
(e.g., for a two-mode network with different numbers of nodes in the first and second mode, a covariate can work either on the first or on the second mode).

# allEffects

The data.frame `allEffects` contains all effects for any data set;
it is read from `data\allEffects.csv`.
The command
`effectsDocumentation()`
prints the main information about all effects in this data.frame to a html file.

(The order in this print differs from the order in `allEffects`; see below.)

`allEffects` is not an effects object, but used as the origin of effects objects.

# getEffects

The R function **`getEffects`** creates the effects object for a data set,
using file `data\allEffects.csv` in the code
and the combination of data variables in the data set.

`allEffects.csv` is a comma-separated file with the following columns:

1. "effectGroup": defines the combination of data variables in the data set;
2. "effectName": the dependent variable is indicated here
   by the proxy `xxxxxx`;
3. "functionName": the estimation statistic;
4. "shortName"
5. "endowment": will this effect, next to the evaluation effects,
   also have a creation and endowment version?
6. "interaction1": the first variable used as explanatory variable,
   with `yyyyyy` as a provisional proxy;
7. "interaction2": the second variable used as explanatory variable,
   with `zzzzzz` as a provisional proxy;

8. "type": `"eval"`, `"rate"`, or `"gmm"`;
9. "basicRate": `FALSE` (for now);
10. "include": `FALSE` (for now);
11. "randomEffects": `FALSE` (for now);
12. "fix": `TRUE` $\Leftrightarrow$ (`"type"=="gmm"`);
13. "test": `FALSE` (for now);
14. "timeDummy": `","` (for now);
15. "initialValue": initial value of the parameter, `0` (for now);
16. "parm": initial value of the internal effect parameter;
17. "functionType": `"objective"`, `"rate"`, or `"gmm"`;
    I don't know why this is different from "type"; is it used at all?

18. "period": the period to which the effect applies (only for basic rate effects);
19. "rateType": for rate effects, `"structural"`, `"diffusion"`, or `"covariate"`;
20. "untrimmedValue": `0` (for now);
21. "effect1": `0` (for now);
22. "effect2": `0` (for now);
23. "effect3": `0` (for now);
24. "interactionType": `"ego"`, `"dyadic"`, `""`, or `"OK"`.
25. "local": the inclusion of only "local" effects allows more efficient ML estimation, see the DPhil dissertation of Charlotte Greenan; if in doubt, choose `FALSE`;
26. "setting": not used yet.

The "for now" indications mean that it may be changed later,
but you can leave it like this when constructing a new effect.

The combination of data variables is determined in `allEffects` by their **`effectGroup`**, given as the first column of `allEffects`.

What you normally do when constructing a new effect, is that you look for an effect that is similar; or, if that is not available, for an effect that uses the same combination of dependent and explanatory variables.

This effect will in any case have the same **`effectGroup`**.

You then have to modify the following columns.

2. "effectName": still use `xxxxxx` but adapt the rest;
3. "functionName": describe the estimation statistic;
4. "shortName": as you chose it;
5. "endowment": in most cases `TRUE`; only `FALSE` if the usual way of working with newly created or terminated ties will not work for estimating the creation or endowment effect;

24. "interactionType": `"ego"`, `"dyadic"`, `""`, or `"OK"`.

# Internal effect parameters

If your effect depends on an internal effect parameter,
please use the symbol ♯ in the "effectName" and "functionName"
so that the user is reminded of its current value;
follow the rules in Section 18.1 of the RSiena manual; and

16. "parm": define a suitable initial value of the internal effect parameter.

(*Up to and including version 1.4.12, the values of internal effect parameters were not mentioned correctly in effect and function names.*
*The rules in Section 18.1 of the RSiena manual are important to allow the use of numbers in effect and function names that do not represent values of internal effect parameters.* )

# Interaction effects

User-defined interactions are created (indeed, by the user) by the R function **`includeInteraction`**, and use the effects in the effects object with **`shortName`** `unspInt` (for networks) or `unspBehInt` (for behavioral variables) or `contUnspInt` (for continuous behavioral variables).

They replace the column entries for `effect1`, `effect2`, and if the interaction involves three interacting effects `effect3`, with the row numbers of the interacting effects in the effects object.

Interactions of network effects are defined by multiplying the change statistics.

The difficulty with defining interactions in RSiena is that
the product of the change statistics does not necessarily correspond
to an evaluation effect — of which this product should be the change statistic.

Therefore a condition is required,
which uses the `interactionType` of effects.

There is some explanation in the manual (Section 5.11).
Further explanations are given in Sections 4.5, 4.6, and 4.7 of

`https://www.stats.ox.ac.uk/~snijders/siena/Siena_algorithms.pdf`

# Network interaction effects

The `interactionType` of a network effect is defined to be `"ego"` if it can be written as

$$s^X_{ik}(x, z) = \sum_j x_{ij}\, c_{ki}(x, z) \ ,$$

where $c_{ki}(x, z)$ is independent of $(x_{i1}, \ldots, x_{in})$ and independent of $j$.

(The argument $z$ refers to a co-evolving variable, and might be null.)

The change statistic then is $\Delta^X_{kij}(x, z) = c_{ki}(x, z)$ .

The `interactionType` of a network effect is defined to be `"dyadic"` if it can be written as

$$s^X_{ik}(x, z) = \sum_j x_{ij}\, c_{kij}(x, z) \ ,$$

where $c_{kij}(x, z)$ is independent of $(x_{i1}, \ldots, x_{in})$.

The change statistic then is $\Delta^X_{kij}(x, z) = c_{kij}(x, z)$ .

# Network interaction effects (2)

Clearly, `"ego"` is a stronger condition for `interactionType` than `"dyadic"`.

For an effect with `interactionType="ego"`,
all interactions are allowed with arbitrary other effects;
for an effect with `interactionType="dyadic"`,
all interactions are allowed with other `"dyadic"` effects.

# Behavior interaction effects

The `interactionType` of a behavior effect is defined to be `"OK"` if it can be written as

$$s_{ik}^{Z}(x, z) = z_i \, s_{ik}^{Z0}(x, z) \, ,$$

where $s_{ik}^{Z0}(x, z)$ is a function not depending on $z_i$.

(The argument $x$ refers to a co-evolving variable, and might be null.)

This is the case for many effects.

Exceptions are, e.g., `quad` and `avSim`.

The change contribution then is

$$\Delta_{kij}^{Z}(x, z) = s_{ik}^{Z0}(x, z) \, .$$

For an effect with `interactionType="OK"`, all interactions are allowed with other `"OK"` effects.

# Conclusion for interaction effects

For a new effect, you have to define
the `interactionType` column in `allEfects.csv`.

This can be `"ego"`, `"dyadic"`, `"OK"`, or `""`,
and should be according to the definitions above.

# Elementary effects

An elementary effect is a contribution to the creation or maintenance of a tie,
defined directly, i.e.,
without expressing it based on the change in some evaluation function.
This is treated in the Siena manual.

For adding a tie, this is added to the objective function,
and for removing a tie it is subtracted.

This is more general than the evaluation function,
and has lost the interpretation of ‘evaluation’.

Examples are the `gwesp` effects, `transTrip1` and `2`, and `WXX`.

Elementary effects always have `interactionType="dyadic"`,
so that they can be used freely in interactions with other `"dyadic"` effects.

# Contextual effects

Some effects do not satisfy the conditions defining "ego" or "dyadic" effects;
e.g., `outAct` and `reciAct`,
but for these it would be nice anyway to use them in interactions.

Other versions of these are created which are "ego" or "dyadic",
meant to be used as contextual effects in interactions,
where the ‘context’ represents the conditions
under which ego makes the choice in the ministep.

They are also centered for their use in interactions.
They are represented as elementary effects,
with `interactionType="dyadic"`,
and indicated by the suffix "_ego" or "_dya" in the **`shortName`**.

# Back to **`getEffects`**

**`getEffects`** creates the effects object generated by `allEffects.csv`
and the combination of variables in the data set.

The combination of data variables for a given effect
is determined in `allEffects` by their **`effectGroup`**,
given as the first column of `allEffects`.

For each variable in the data set,
and each combination of variables that is permitted,
all effects in the corresponding **`effectGroup`** are created.

Dependent behavioral variables in an explanatory role
are treated as actor covariates.

Strings `xxxxxx`, `yyyyyy`, `zzzzzz`, and ♯ in `allEffects`
are filled in by **`getEffects`** as discussed above.

These are used only for the textual representation of effects in R ,
and have no further computational consequences.

# More about effectGroups

The first column of `allEffects` is the **`effectGroup`**. By
`unique(allEffects$effectGroup)`
you can see that, currently, there are 59 **`effectGroup`**s.
Unfortunately, there is no clear system in their names.
You will have to look to the correspondence of effects and **`effectGroup`**s
to determine the meaning of an **`effectGroup`**.
*Also see Section 18.3 in the manual for effectGroups and two-mode networks.*

---

In the created effects object, as well as in the print by
**`effectsDocumentation()`**,
the effects are ordered by **`effectGroup`**.

An effect may be useful for several **`effectGroup`**s.
(E.g., `altX` is used in **`covarSymmetricObjective`**,
**`covarBipartiteObjective`**, and **`covarNonSymmetricObjective`**.)

# Operation of **getEffects**

The R function **getEffects** is defined in the file `effects.r`,
which you will find in
`https://github.com/stocnet/rsiena/blob/main/R/effects.r`;
this calls internal function **createEffects**
for each **effectGroup** separately,
handling all variables in the **Siena** data set.

Mostly you can just take file `effects.r` for granted.

However, if your new effect is not covered by any existing **effectGroup**,
you have to add a section to the code for **getEffects** in `effects.r`,
and specify its rank order in `effectsDocumentation.r`.

# But sometimes...

(Mostly, the contents of this slide are not needed!)

But sometimes you do have to do something to file `effects.r`,
because the definition of **`effectGroup`** does not cover everything
for effects to be included by **`getEffects`** in the effects object.

If you made a new effect following an existing effect as a model,
you have to check whether
the **`shortName`** of this effect occurs in `effects.r`.
If it does, see how it occurs and see if you have to do something similar.

To be prepared, it is helpful to look at the file `effects.r`,
and search for the string `shortName`, and see how it is being treated.

The effects object is an important part of the information transferred from the R part to the C++ part of RSiena.

This happens in the R function **`initializeFRAN`**.
Only those columns are transferred that are relevant for C++.
These are the variables in the C++ class **`EffectInfo`**.

# Checking what you did

My limited way of checking a newly created effect (if it compiles)
is accumulated in script `checkEffects.r`
in the base directory of the source code.

For a data structure to which new effect can be applied,
the internal data sets (or simulated data) of RSiena
are used to create a data object, without any missings.

A model is constructed that contains the new effect and perhaps some others.
The default estimation algorithm is used.
Normally, convergence should be excellent.
Then the estimation statistic is checked
by comparing the result of `targets` with what is manually calculated.

# Summary: Overview of constructing effects

To construct a new effect, you have to do the following things.

1. Choose an `effectName` and a `shortName`.
2. Define the effect statistic and the change statistic.
3. Insert a new line in the code in `\data\allEffects.csv`.
4. Construct a new effect class in `\src\model\effects`,
   or add to an existing effect class.
5. Attach the new effect by appropriate lines in the `EffectFactory`.
6. If you constructed a new effect class,
   insert the name of the `cpp` file in `\src\sources.list`
   and the name of the `h` file in `\src\model\effects\allEffects.h`
7. Check that the effect runs properly, similarly as what was done in
   `https://github.com/stocnet/rsiena/blob/main/checkEffects.R`.

Points 4–6 are treated in the slides "*Effects in RSiena on the C++ side*".