Siena algorithms

Tom A.B. Snijders

December 2, 2025

Contents

This paper gives a sketch of the main algorithms used in RSiena. It is meant as background material for understanding the code of RSiena.

For the Generalized Method of Moments ('GMoM'), there is the separate text Changes_RSiena_GMoM.pdf.

1 Notation

Logarithms (denoted $\log$) are natural logarithms.

Symbols given *in italic sf font* refer to the names of variables used in the R or C++ code.

Generic symbols for variables

There are R_N networks and R_B behavior variables.
We require $R_N + R_B \geq 1$; if the current structure of RSiena requires this, then we require $R_N \geq 1$ (but if it is easy to work with $R_N = 0, R_B \geq 1$ then it would be nice to permit this.)

i, j		actors.
m		index for time period from t_{m-1} to t_m ($m = 2, \ldots, M$).
M	*observations*	total number of observations
x		all R_N networks jointly (one outcome).
z		all R_B behaviors jointly (one outcome).
y		state: all networks and behaviors jointly (one outcome).
W		variable with values N or B, indicating whether something refers to network or behavior.
r		index number of networks or behaviors, ranging from 1 to R_W .
missing		missingness indicators for ordered triples (i, j, r) referring to networks r and for ordered pairs (i, r) referring to behaviors; values are F (*False*) and T (*True*);

		if T, then further specifications are Start / End / Both, referring to the observation period from t_{m-1} to t_m .
maxbeh$_r$		maximum of the range of the r'th behavior variable.
^{N}		as superscript: refers to network dynamics.
^{B}		as superscript: refers to behavior dynamics.

Changing variables (outcomes)

$^{\text{obs}}$		as superscript: refers to observed values.
θ	*theta*	vector of all statistical parameters.
p	*pp*	dimension of θ.
J		simulated data score function (vector of partial derivatives of log-likelihood (p-vector).
t		time.
$N_{ij}^{(r)}$		dummy tie variable indicating $i \xrightarrow{r} j$ for rth network.
$B_i^{(r)}$		behavior variable for rth behavior for actor i.

Replacing an index by + denotes summation over this index.
Toggling a number a in $\{0, 1\}$ means replacing a by $1 - a$.

Functions

$\lambda^N(r, i, x, z)$	rate function, network r. (0 for inactive actors)
$\lambda^B(r, i, x, z)$	rate function, behavior r. (0 for inactive actors)
$f^N(r, i, x, z)$	evaluation function function, network r.
$f^B(r, i, x, z)$	evaluation function, behavior r.
$g_e^N(r, i, j, x, z)$	endowment function function, network r

$g_c^N(r,i,j,x,z)$	creation function function, network r
$g_e^B(r,i,x,z)$	endowment function, behavior r.
$g_c^B(r,i,x,z)$	creation function, behavior r.
$\Delta f^N(r,i,j,x,z)$	change in $f^N(r,i,x,z)$ by toggling $N_{ij}^{(r)}$.
$\Delta f^B(r,i,v,x,z)$	change in $f^B(r,i,x,z)$ by changing $B_i^{(r)}$ to $B_i^{(r)}+v$.
$\sim E(\lambda)$	generate random variable according to exponential distribution with parameter λ (note: expected value $1/\lambda$).

Note. Whether the endowment function makes sense for behavior variables with a range of more than two values, is doubted. But we keep it included anyway, for the moment.

The R convention is followed of denoting an assignment statement by $a \leftarrow b$, meaning that the variable a gets the value b.

2 Outline of model dynamics / simulation algorithm

The tie-based model is defined as a continuous-time Markov chain by the following algorithm for generating the next change in the outcome. This is formulated here for the case that the state space includes networks as well as behavior. If there are no behavior variables B, then the steps referring to these variables can simply be dropped. In the code this is function simstats0c.

To estimate derivatives of expected values of statistics with respect to the parameters, the score function method (**?**) is used in the default method to estimate standard errors. This is indicated by 'SF only' and can be skipped if the finite differences ('FD') option, which also employs common random numbers, is used to estimate standard errors.

For each network variable numbered r the following logical (boolean) variables are defined at the moment of data entry:

- *uponly* (r) = $\big\{$for all $i, j, m : x_{ij}^{(r)\,\mathrm{obs}}(t_m) \leq x_{ij}^{(r)\,\mathrm{obs}}(t_{m+1})\big\}$;
- *downonly* (r) = $\big\{$for all $i, j, m : x_{ij}^{(r)\,\mathrm{obs}}(t_m) \geq x_{ij}^{(r)\,\mathrm{obs}}(t_{m+1})\big\}$;

For each ordered pair of network variables numbered r and $r' \neq r$, we define the following logical (boolean) variables:

- *higher* (r, r') = $\big\{$for all $i, j, m : x_{ij}^{(r)\,\mathrm{obs}}(t_m) \geq x_{ij}^{(r')\,\mathrm{obs}}(t_m)\big\}$;
- *disjoint* (r, r') = $\big\{$for all $i, j, m : \min\{x_{ij}^{(r)\,\mathrm{obs}}(t_m),\, x_{ij}^{(r')\,\mathrm{obs}}(t_m)\} = 0\big\}$;
- *atleastone* (r, r') = $\big\{$for all $i, j, m : \max\{x_{ij}^{(r)\,\mathrm{obs}}(t_m),\, x_{ij}^{(r')\,\mathrm{obs}}(t_m)\} = 1\big\}$.

Analogous definitions *uponly* , *downonly* , *higher* can be made for behaviour variables.

Model for microstep

1. Initialize time at $t = 0$; initialise networks and behaviors x, z at their observations at wave $m - 1$.
   SF only: initialise the score function at $J_m = 0$.
2. Current time, networks, behaviors, denoted by t, x, z.
3. For all r, generate $\Delta t_r^N \sim E(\lambda^N(r, +, x, z))$.
4. For all r, generate $\Delta t_r^B \sim E(\lambda^B(r, +, x, z))$.
5. Let W, r be the variable for which $\Delta t_r^W = \min_r\{t_r^N, t_r^B\}$.
   If $W = N$, goto **??**; if $W = B$, goto **??**.
   (*Note*. An alternative, mathematically equivalent, is to choose (W, r) with probabilities proportional to $\lambda^W(r, +, x, z)$ and only for this W, r generate $\Delta t_r^W \sim E(\lambda^+(+, +, x, z))$.
   This is more efficient but the gain in computation time must be negligible.)
6. Choose i with probabilities $\lambda^W(r, i, x, z)/\lambda^W(r, +, x, z)$.
7. Set $t = t + \Delta t_r^W$. (*time step*)

8. SF only: set

$$J_m \;=\; J_m + \frac{\partial \ln\left(\lambda^W(r,i,x,z)/\lambda^+(+,+,x,z)\right)}{\partial\theta} + \frac{\partial \ln \lambda^W(r,i,x,z)}{\partial\theta}\,.$$

(Note: first added term for generating W, r, i; second term for t.)

9. Define C as the set of j for which $N_{ij}^{(r)}$ is allowed to change.
This is the set of all $j \in \{1,\ldots,n\}$ from which are excluded all $j \neq i$ for which at least one of the following hold:

   (a) $N_{ij}^{(r)}$ is structurally determined;

   (b) *uponly* (r) and $N_{ij}^{(r)} = 1$;

   (c) *downonly* (r) and $N_{ij}^{(r)} = 0$;

   (d) for some $r' \neq r$, *higher* (r, r') and $N_{ij}^{(r)} = N_{ij}^{(r')} = 1$;

   (e) for some $r' \neq r$, *higher* (r', r) and $N_{ij}^{(r)} = N_{ij}^{(r')} = 0$;

   (f) for some $r' \neq r$, *disjoint* (r, r') and $N_{ij}^{(r)} = 0, N_{ij}^{(r')} = 1$;

   (g) for some $r' \neq r$, *atleastone* (r, r') and $N_{ij}^{(r)} = 1, N_{ij}^{(r')} = 0$.

   Obviously, in many cases, there are never any excluded j; and if there is only one dependent network variable, the four last conditions are never satisfied.
   If C has one element, this must be i; then go to **??**.
   (The following steps in this item then are vacuous, so they can be skipped.)
   If C is empty, this is an error, and the program must stop with an error message.

   For all $j \in C$, calculate $h_j = \Delta f^N(r,i,j,x,z)$, and $h_i = 0$.
   For all $j \in C$ with $N_{ij}^r = 1$, calculate $h_j = h_j - g_e^N(r,i,j,x,z)$.
   For all $j \in C$ with $N_{ij}^r = 0$, calculate $h_j = h_j + g_c^N(r,i,j,x,z)$.
   Choose $j \in C \cup i$ with probabilities

$$\pi_j = \frac{\exp(h_j)}{\sum_k \exp(h_k)} \tag{1}$$

   SF only: set $J_m = J_m + \partial h_j/\partial\theta \;-\; \sum_k \pi_k\, \partial h_k/\partial\theta$.
   If $j \neq i$, toggle N_{ij}^r. (*network step*)
   Goto **??**.

10. Let C be the set of $v \in \{-1, 1\}$
for which $B_i^{(r)} + v$ is within the range of $B_i^{(r)}$.
For all $v \in C$, calculate $h_v = \Delta f^B(r, i, v, x, z)$, and $h_0 = 0$.
If $-1 \in C$, calculate $h_{-1} = h_{-1} - g_e^B(r, i, x, z)$.
If $+1 \in C$, calculate $h_{+1} = h_{+1} + g_c^B(r, i, x, z)$.
Choose $v \in C \cup 0$ with probabilities

$$\pi_v = \frac{\exp(h_v)}{\sum_u \exp(h_u)} \tag{2}$$

SF only: set $J_m = J_m + \partial h_v/\partial\theta \; - \; \sum_u \pi_u \, \partial h_u/\partial\theta$.
Add v to B_i^r. (*behavior step*)
Goto **??**.

Stopping criterion

1. In the unconditional estimation option, microsteps continue until $t \geq 1$.
Note that, by convention, time duration between waves is set to be unity.
SF only: set
$J_m = J_m - (1 - t^{\text{last}})\big(\partial \ln \lambda^+(+, +, x, z)/\partial\theta\big)$,
where t^{last} is the last generated value of t before t exceeded 1.

2. In the conditional estimation option, if the conditioning variable is network $N^{(r)}$, microsteps continue until

$$\sum_{i,j} \mid N_{ij}^{(r)} - x_{ij}^{(r)\,\text{obs}}(t_{m-1}) \mid \geq \sum_{i,j} \mid x_{ij}^{(r)\,\text{obs}}(t_m) - x_{ij}^{(r)\,\text{obs}}(t_{m-1}) \mid ,$$

where the sum is over all tie variables that are not structurally fixed at t_{m-1} or t_m (note that it is possible that tie variables are structurally fixed but have different subsequent values).
If the conditioning variable is behavior $B^{(r)}$, microsteps continue until

$$\sum_{i} \mid B_i^{(r)} - z_i^{(r)\,\text{obs}}(t_{m-1}) \mid \geq \sum_{i} \mid z_i^{(r)\,\text{obs}}(t_m) - z_i^{(r)\,\text{obs}}(t_{m-1}) \mid ,$$

where the sum is over all actors that are not structurally inactive at t_{m-1} or t_m.

Score function

The generated statistics S can be written as $S = \sum_{m=2}^{M} S_m$, where S_m is calculated in consequence of the simulation of the process in the period from t_{m-1} to t_m. Denote the value of J generated in this period by J_m. To use the score function method, we calculate

$$< SJ >= \sum_{m=2}^{M} S_m \, J'_m \; . \tag{3}$$

This is a $p \times p$ matrix, and an estimate for $\partial E_\theta S / \partial \theta$.
(Or do we work with $\sum_{m=2}^{M} (S_m - s^{\text{obs}}) J'_m$ for numerical accuracy?)

The decomposition into the $M-1$ periods is kept because it allows a more efficient variance reduction (see further down).

(Mathematical note: for simulations taking place according to parameter θ, $E_\theta J_m = 0$. We will later subtract a value $s_m J'_m$ for an 'almost constant' s_m; this does not affect the expected value, but leads to a considerable variance reduction; see **?**.)

3 Outline of Robbins-Monro algorithm for MoM and ML

This section is based on the appendix of **?**, and updated to include the algorithm changes that were incorporated after 2001. The implementation of the algorithm in RSiena is function robmon, and has a number of additional details to improve convergence.

This function is used for ML as well as MoM estimation. One difference is that for MoM, estimation statistics are used as described in Section **??**, while for ML, the statistics are the complete-data score functions. Another difference is that the covariance matrix of the estimator (which implies the standard errors) is estimated differently. In the current section, the description of the covariance matrix of the estimator is for the MoM. The description of the covariance matrix of the ML estimator is given in Section **??**.

The purpose of the algorithm is to approximate the solution of the moment equation

$$\mathsf{E}_\theta S = s\,, \tag{4}$$

where $s = s^{\mathrm{obs}}$, the observed value. The solution is denoted by θ_0. The algorithm is a multivariate version of the Robbins-Monro (1951) algorithm. It uses the idea of Polyak (1990) and Ruppert (1988) to employ a diagonal matrix $\tilde{D}$ in the iteration step (**??**)

$$\hat{\theta}_{N+1} \,=\, \hat{\theta}_N \,-\, a_N\, \tilde{D}^{-1}\,(S_N - s)\,, \tag{5}$$

and estimate the solution by partial averages of $\hat{\theta}_N$ rather than the last value; and it uses the idea of Pflug (1990) to let the values of a_N remain constant if the average products of successive values $(S_N - s)(S_{N-1} - s)$ are positive, since this suggests that the process still is drifting toward its limit value. However, the specification used here deviates from Pflug's proposal by requiring, for the premature decrease of a_N, that for *each* coordinate the partial sum of the product of successive values be negative, rather than requiring this only for the sum over the coordinates. Further, the number of steps for which a_N is constant is bounded between a lower and an upper limit to ensure that a_N is of order N^{-c}.

Under the option *doubleAveraging* , the iteration step is

$$\hat{\theta}_{N+1} \,=\, \bar{\theta}_N \,-\, a_N\, N\, \tilde{D}^{-1}\,(\overline{S}_N - s)\,, \tag{6}$$

where

$$\bar{\theta}_N \,=\, \frac{1}{N}\sum_{n\le N} \hat{\theta}_n\,, \quad \overline{S}_N \,=\, \frac{1}{N}\sum_{n\le N} s_n\,,$$

implementing a proposal of **?** studied by **?**.

Whether the algorithm yields an estimate that indeed solves the moment equation (**??**) to a satisfactory degree of precision is checked in the 'third phase' of the algorithm below.

The reason for incorporating the matrix $\tilde{D}$ is to achieve better compatibility between the scales of S and of θ. The diagonal elements of $\tilde{D}$ are defined as the estimated values of the derivatives $\partial \mathsf{E}_\theta(S_k)/\partial\theta_k$ where θ is at its

initial value. To see that this leads to compatibility of the scales of S and θ note that in the extreme case where var$(S_k) = 0$ and the diagonal elements of $\tilde{D}$ are equal to $\partial \mathsf{E}_\theta(S_k)/\partial\theta_k$, (**??**) for $a_N = 1$ is just the iteration step of the Newton-Raphson algorithm applied to each coordinate of S separately. Thus, beginning the algorithm with a_N in the order of magnitude of 1 will imply that the initial steps have an approximately right order of magnitude.

The results of Polyak and Ruppert do not point exclusively to diagonal matrices; other positive definite matrices could also be used. Therefore, the option is available instead of a diagonal matrix to use a partial diagonalization; this uses the parameter *diagonalize* set in sienaModelCreate.

The number of dimensions of θ and of S is denoted by p and the initial value is denoted θ_1. 'Generating $S \sim \theta$' means to simulate the model according to parameter value θ and calculate the statistics S.

The estimation of derivatives has two options: finite differences ('FD') and score function ('SF'). SF is more efficient and unbiased (**?**) and therefore is the default, FD is available for some models for which the derivatives of the log-likelihood needed for SF have not yet been worked out.

The FD option is based on disturbing the current parameter values by adding the value ϵ_j, and using common random numbers. Because of the discrete nature of the simulated statistics, a very small ϵ_j will yield simulated values that with high probability are equal to the values obtained without the disturbance. This is undesirable (see **?**). Good values of ϵ_j must be such that with rather high probability (say, more than .5) the simulated values are not identical to those obtained without the disturbance.

Symbols given *in italic sf font* refer to the names of variables used in the R code.

The standard initial value is calculated in function *getNetworkStartingVals()* in file *effects.r* . This uses an adapted version of (11.41) in **?**. The adaptation is a precision-based weighting of multiple periods in case $M \geq 3$ (the vector *prec* used in the function is a measure of precision).

The algorithm consists of three phases.

1. In this phase a small number n_1 of steps are made to estimate *dfra* = $D(\theta_1) = \left(\partial \mathsf{E}_\theta(S)/\partial\theta\right) |_{\theta=\theta_1}$.

This estimate is used to define $\tilde{D}$. Denote by e_j the j'th unit vector in p dimensions.

Initialise $\mathrm{Sum}_d = 0_{p\times p}$, $\mathrm{Sum}_S = 0_{p\times 1}$.
For SF, initialise additionally
$\mathrm{Sum}_{Sm} = 0_{p\times 1}$ and $\mathrm{Sum}_{Jm} = 0_{p\times 1}$ for $m = 2, \ldots, M$.
For $N = 1$ to n_1, do the following.

(FD) Generate

$$\textit{fra} = S \sim \theta_1$$
$$S_j \sim \theta_1 + \epsilon_j e_j \ (j = 1, \ldots, p),$$

where all these $p + 1$ random vectors use a common random number stream to make them strongly positively dependent and where ϵ_j are suitable constants.
Compute the difference quotients

$$\textit{sdf} = d_j = \epsilon_j^{-1}(S_j - S) \ ;$$

for small values of ϵ_j the expected value of the matrix $d = (d_1, ..., d_p)$ approximates $D(\theta_1)$. However, ϵ_j must be chosen not too small because otherwise the variances of the d_j become too large.
Update

$$\mathrm{Sum}_S = \mathrm{Sum}_S + S$$
$$\mathrm{Sum}_d = \mathrm{Sum}_d + d$$

(SF) Generate

$$\textit{fra} = S \sim \theta_1$$

its components being $S_m \, (m = 2, \ldots, M)$ (see 'Score Function' above), the complete-data score functions J_m $(m = 2, \ldots, M)$, and $d =< SJ >$ according to (**??**).

Update

$$\begin{aligned}
\mathrm{Sum}_S &= \mathrm{Sum}_S + S \\
\mathrm{Sum}_d &= \mathrm{Sum}_d + d \\
\mathrm{Sum}_{Sm} &= \mathrm{Sum}_{Sm} + S_m,\ m = 2, \ldots, M \\
\mathrm{Sum}_{Jm} &= \mathrm{Sum}_{Jm} + J_m,\ m = 2, \ldots, M
\end{aligned}$$

The differences *fra* $-s$ are stored as *sf* in procedure *doPhase1or3Iterations* .

At the end of Phase 1, calculate the following results:

(a) Estimate $E_{\theta_1} S$ by

$$\bar{s} = \frac{\mathrm{Sum}_S}{n_1} .$$

(b) In *CalculateDerivative* : Estimate $D(\theta_1)$ by

$$\text{FD: } \hat{D} = \frac{\mathrm{Sum}_d}{n_1}$$

$$\text{SF: } \hat{D} = \frac{\mathrm{Sum}_d}{n_1} - \frac{\sum_{m=2}^{M} \mathrm{Sum}_{Sm}\mathrm{Sum}_{Jm}}{n_1^2} .$$

(c) Partially (or entirely) diagonalized matrix

$$\tilde{D} = \textit{diagonalize} \times \mathrm{diag}(\hat{D}) + (1 - \textit{diagonalize}) \times \hat{D}$$

where *diagonalize* is set in function sienaAlgorithmCreate, with default value 1.

(d) Componentwise regression coefficients of statistics S on scores $\sum_m J_m$, i.e., for each coordinate j,

$$\textit{RegrCoef}_j = \frac{\widehat{\mathrm{cov}}(S_j, \sum_m J_{mj})}{\widehat{\mathrm{var}}(S_j)}$$

from the values generated in Phase 1.
(For purely descriptive purposes, the corresponding correlation coefficients are also calculated.)

(e) Make one partial estimated Newton-Raphson step,

$$\hat{\theta} = \theta_1 - a_1 \hat{D}^{-1} (\bar{s} - s) \ .$$

where

targets $= s =$ observed values.

Phase 2. *Note 1: I cannot find anything about phase2.0 in the current code – ts 23-05-13.*
Note 2: the use of observedPos and observedNeg apparently was discontinued.
The boolean variable *phase2.0* differentiates between having or not having a subphase 2.0. Such a subphase was introduced in version R-Forge 1.1-220 (August 2012).
If *phase2.0* then $k_{\min} = 0$, else $k_{\min} = 1$.
Repeat for $k = k_{\min}, \ldots, k_{\max}$ (subphases):
function *proc2subphase*

(a) Initialise *nit* = $N = 0$, $\mathrm{Sum}_{\hat{\theta}} = 0_{p\times 1}$, $\mathrm{Sum}_{\Delta} = 0_{p\times 1}$, $S_{\mathrm{prev}} = 0_{p\times 1}$, *ac* = $\mathrm{AC} = 0_{\mathrm{p}\times\mathrm{p}}$, *observedPos* = *observedNeg* $=$ FALSE$_{\mathrm{p}\times 1}$.

(b) Generate

fra $= S \sim \theta$

or, for multiple processes, as the average of *int* independent replicates of such variables;
if *dolby* , then also generate the sum of scores $\sum_m J_m$ and calculate

$$\textit{fra} \ = \ \textit{fra} \ - \textit{regrCoef} \ * \Big(\sum_m J_m \big)$$

(componentwise multiplication); this serves for reducing the variance while not affecting the expected value.

(c) Update

$$\begin{aligned}
\mathrm{Sum}_\Delta &= \mathrm{Sum}_\Delta + (S - s) \\
\text{If } \textit{doubleAveraging}\text{ , } \hat{\theta} &= \mathrm{Sum}_{\hat{\theta}}/N \; - \; a_N \, \tilde{D}^{-1} \, \mathrm{Sum}_\Delta \\
\text{else } \hat{\theta} &= \hat{\theta} \; - \; a_N \, \tilde{D}^{-1} \, (S - s) \\
N &= N + 1 \\
\mathrm{Sum}_{\hat{\theta}} &= \mathrm{Sum}_{\hat{\theta}} + \hat{\theta} \\
\text{if } N \geq 2\text{, then } \mathrm{AC} &= \mathrm{AC} + (\mathrm{S} - \mathrm{s})(\mathrm{S}_{\mathrm{prev}} - \mathrm{s})' \\
\textit{observedPos}_{\,j} &= \textit{observedPos}_{\,j} \text{ or } \textit{fra}_{\,j} \geq 0 \;\; \text{(for all } j) \\
\textit{observedNeg}_{\,j} &= \textit{observedNeg}_{\,j} \text{ or } \textit{fra}_{\,j} \leq 0 \;\; \text{(for all } j) \\
S_{\mathrm{prev}} &= S \; .
\end{aligned}$$

(updates for *observedPos* and *observedNeg* relevant for $k = 0$ only)

(d) Stopping rule for subphase $k = 0$:
If $N >= n^+_{2k}$ or *observedPos = observedNeg* $=$ TRUE$_{p\times 1}$ then goto (b).

Stopping rule for subphase $k \geq 1$:
If $N >= n^+_{2k}$ or ($N >= n^-_{2k}$ and $\max_k \mathrm{AC}_{\mathrm{kk}} \leq 0$), then
$\{$ update $\hat{\theta} = N^{-1}\mathrm{Sum}_{\hat{\theta}}$; set $a_k = a_k \times$ reductionfactor$\}$; goto (b).
(But if *phase2.0* then the update to $\hat{\theta}$ is done only for $k \geq 2$.)

In the code, *theta* $= \theta$, *gain* $= a_N$, *ac* = AC, *thav* $= \mathrm{Sum}_{\hat{\theta}}$, *nit* $= N$, *n2min* $= n^-_{2k}$, *n2max* $= n^+_{2k}$.

Phase 3. Phase 3 is used only for the estimation of $D(\theta)$ and Cov$(\hat{\theta})$, and as a check for the (approximate) validity of (**??**). The value of $\hat{\theta}$ is left unchanged in this phase and is equal to the value obtained after last subphase of phase 2. The procedure is mainly as in phase 1.

Initialise $\mathrm{Sum}_d = 0_{p\times p}$, $\mathrm{Sum}_S = 0_{p\times 1}$, $\mathrm{SumSq}_S = 0_{p\times p}$.
For SF, initialise additionally $\mathrm{Sum}_{Sm} = 0_{p\times 1}$ and $\mathrm{Sum}_{Jm} = 0_{p\times 1}$ for $m = 2, \ldots, M$.
For $N = 1$ to n_3, do the following.

(FD) Generate

$$\begin{aligned} \textit{fra} &= S \sim \theta \\ & S_j \sim \theta + \epsilon_j e_j \ (j = 1, \ldots, p), \end{aligned}$$

where all the $p + 1$ random vectors use a common random number stream to make them strongly positively dependent and where ϵ_j are suitable constants. Compute the difference quotients

$$\textit{sdf} = d_j = \epsilon_j^{-1}(S_j - S) \ .$$

Update

$$\begin{aligned} \mathrm{Sum}_S &= \mathrm{Sum}_S + S \\ \mathrm{SumSq}_S &= \mathrm{SumSq}_S + S\,S' \\ \mathrm{Sum}_d &= \mathrm{Sum}_d + d \end{aligned}$$

(SF) Generate

$$\textit{fra} = S \sim \theta$$

its components being $S_m \, (m = 2, \ldots, M)$ (see 'Score Function' above), the complete-data score functions J_m $(m = 2, \ldots, M)$, and $d =< SJ >$ according to (**??**).
Update

$$\begin{aligned} \mathrm{Sum}_S &= \mathrm{Sum}_S + S \\ \mathrm{SumSq}_S &= \mathrm{SumSq}_S + S\,S' \\ \mathrm{Sum}_d &= \mathrm{Sum}_d + d \\ \mathrm{Sum}_{Sm} &= \mathrm{Sum}_{Sm} + S_m, \ m = 2, \ldots, M \\ \mathrm{Sum}_{Jm} &= \mathrm{Sum}_{Jm} + J_m, \ m = 2, \ldots, M \end{aligned}$$

At the end of Phase 3, calculate the following results:

(a) Estimate $E_{\hat{\theta}} S$ and $\mathrm{Cov}_{\hat{\theta}} S$ by

$$\bar{s} = \frac{\mathrm{Sum}_S}{n_3} \ , \quad \Sigma = \frac{\mathrm{SumSq}_S}{n_3} \ - \ \bar{s}\,\bar{s}' \ .$$

(b) To check (approximate) validity of (**??**) compute the t-ratios for convergence,

$$\textit{tstat} \ = t_j = \frac{\bar{s}_j - s_j^{\text{obs}}}{\sigma_j} \ , \tag{7}$$

where σ_j is the square root of the j'th diagonal element of Σ.

(c) In *CalculateDerivative3* : Estimate $D(\hat{\theta})$ by

$$\text{FD: } \hat{D} = \frac{\text{Sum}_d}{n_3}$$
$$\text{SF: } \hat{D} = \frac{\text{Sum}_d}{n_3} - \frac{\sum_{m=2}^{M} \text{Sum}_{Sm}\text{Sum}_{Jm}}{n_3^2} \ .$$

(d) Estimate the covariance matrix of $\hat{\theta}$ by

$$\text{Cov}(\hat{\theta}) \ = \ \hat{D}^{-1}\Sigma\hat{D}'^{-1} \ . \tag{8}$$

The standard errors are the square roots of the diagonal elements of $\text{Cov}(\hat{\theta})$.

(e) For possible later use with the *prevAns* option, recalculate the componentwise regression coefficients

$$\textit{RegrCoef}_{\ j} \ = \ \frac{\widehat{\text{cov}}(S_j, \sum_m J_{mj})}{\widehat{\text{var}}(S_j)}$$

(all j) from the values generated in Phase 3.

This algorithm contains various constants that can be adapted so as to achieve favorable convergence properties. Experience with various data sets led to the following values.

The number of steps in Phase 1 is

$$\textit{n1} \ = n_1 = 7 + 3p \ .$$

In the *dolby* option, at least 50 steps are taken in Phase 1.
The minimum number of steps in subphase $2.k$ is

$$\textit{n2minimum[1]} \ = n_{2k}^{-} = ((2.52)^k) \ \times \ (7 + p)$$

which is meant to approximate $n_{2k}^- = 2^{4(k+2)/3}(7+p)$; the maximum number is

$$\textit{n2maximum[1]} = n_{2k}^+ = n_{2k}^- + 200 \ .$$

For multiple processes, we use

$$n_{2k}^- = ((2.52)^{k-1}) \times \max\{5, (7+p) \times 2.52/\textit{int}\}$$

where *int* is the number of processes. These bounds n_{2k}^- and n_{2k}^+ are determined so that $N^{3/4}a_N$ tends to a finite positive limit.
For large p they are rather conservative (i.e., unnecessarily large).
The default number of steps in phase 3 in the SF option is *n3* = $n_3 = 1000$.
For the FD option, 500 is a good default.

The default number of subphases is *nsub* = 4; more or fewer subphases could be used to obtain smaller or larger precision, but 4 seems really a good number.
The initial value of a_N in phase 2 is *firstg* = 0.2, and for multiple processes $0.2 \times \sqrt{\textit{int}}$.
The reductionfactor at the end of subphases in phase 2 is for MoM estimation parameter *reduceg* set by *sienaAlgorithmCreate* (default 0.5), but for ML estimation it is 0.25.

The values of *epsilon* = ϵ_j in the FD option are chosen initially as 0.1, but in Phase 1 a check is made and if the j'th coordinate of $d - d_j$ is exactly 0 for all or most of the simulations then ϵ_j is adaptively increased. The variability obtained by the use of small values of ϵ_j is more serious than the bias obtained by the use of a large value. An ideal value would be to have ϵ_j slightly less than the standard error of $\hat{\theta}_j$. However, this is known only after the estimation has finished. (Of course in many cases there have been done earlier estimations, and the information obtained from them might be used for this purpose.)

3.1 prevAns: using the previous answer

The *prevAns* option in *siena07* does the following.

1. Using the function *updateTheta* , for the requested effects the initial parameter values are taken from the previous answer object.

2. If the specifications of the previous answer and the current effects object correspond, then the following objects are taken from Phase 3 of the previous answer, and used to substitute for the calculations done at the end of Phase 1:
   dfra ; matrix *dinv* which is used to calculate matrix *D* ; *sf* ; *regrCoef* ; *regrCor* .
   This substitution is determined by the flag *haveDfra* .

3.2 Convergence criterion

Up to version 1.1-284, the proposed convergence criterion focused on the t-ratios for convergence *tstat* defined in (**??**), of which the maximum absolute value

$$\textit{tmax} \;=\; \max_j \{\textit{tstat}_{\,j}\}$$

should be less than 0.10. This was supported by simulations of the estimators using this convergence criterion, for which expected values and coverage rates of confidence intervals were good. Since then it has appeared that the convergence criterion should be improved, and the following conclusion was reached.

The *overall maximum convergence ratio* should be used as an additional convergence criterion. It is defined as the maximum t-ratio for convergence for any linear combination of the parameters,

$$\textit{tconv.max} \;=\; \max_b \left\{ \frac{b'\left(\bar{s} - s^{\mathrm{obs}}\right)}{\sqrt{b'\,\Sigma\,b}} \right\} \;, \tag{9}$$

where $\bar{s}$ is the average simulated vector of statistics and s^{obs} is its observed value. This is equal to (use Cauchy-Schwarz)

$$\max_c \left\{ \frac{c'\,\Sigma^{-1/2}\left(\bar{s} - s^{\mathrm{obs}}\right)}{\sqrt{c'c}} \right\} \;=\; \sqrt{\left(\bar{s} - s^{\mathrm{obs}}\right)'\,\Sigma^{-1}\left(\bar{s} - s^{\mathrm{obs}}\right)} \;. \tag{10}$$

The definition implies that

$$\textit{tconv.max} \;\geq\; \textit{tmax} \;.$$

Further, the vector b^* of weights that has the largest convergence t-ratio (i.e., the largest value of the expression in braces in (**??**)) is

$$b^* = \Sigma^{-1/2} c^* = \Sigma^{-1/2} \Sigma^{-1/2} \left(\bar{s} - s^{\text{obs}} \right) = \Sigma^{-1} \left(\bar{s} - s^{\text{obs}} \right) . \tag{11}$$

In case of difficult convergence, the latter vector of weights may be used to diagnose which is the linear combination giving most trouble to the algorithm. It may be even more instructive to study the weights of standardized statistics (i.e., statistics divided by their standard deviation).

These can be obtained from RSiena, if the answer object is called `ans`, as

```
(solve(ans$msf) %*% apply(ans$sf,2,mean)) * sqrt(diag(ans$msf))
```

A more readable representation may be given by

```
round((solve(ans$msf) %*% apply(ans$sf,2,mean)) * sqrt(diag(ans$msf))
```

A study was made, using several data sets and model specifications, in which estimations were first run until *tmax* ≤ 0.10, and then continued (with *prevAns*) until *tconv.max* ≤ 0.20 and then on until *tconv.max* ≤ 0.15. After each of these three endpoints, the estimate was retained. This led to a set of estimates $\check{\theta}_i$ each with their estimated covariance matrices $\check{\Sigma}_i$ and values *tmax* $_i$ and *tconv.max* $_i$.

To summarize these estimations and covariance matrices, only those i were used for which *tconv.max* $_i \leq 0.10$, to obtain robust estimates (elementwise) of the mean $\check{\theta}_i$, and of the mean $\check{\Sigma}_i$; these are denoted $\bar{\theta}$ and $\bar{\Sigma}$. It was checked that $\bar{\Sigma}$ still was positive definite. The quality of all estimates $\check{\theta}_i$ then was assessed by the squared Mahalanobis-type distances

$$d_i = (\check{\theta}_i - \bar{\theta})' \, \bar{\Sigma}^{-1} \, (\check{\theta}_i - \bar{\theta}) .$$

Regarding $\bar{\theta}$ as the true estimate, this measures the how well $\check{\theta}_i$ approximates the true estimate, and therefore can be regarded as a measure of convergence.

This led to plots such as those in Figure **??** (obtained for a model with some highly correlated effects, for which the algorithm was expected to encounter some difficulties). We see clearly that, although the great majority

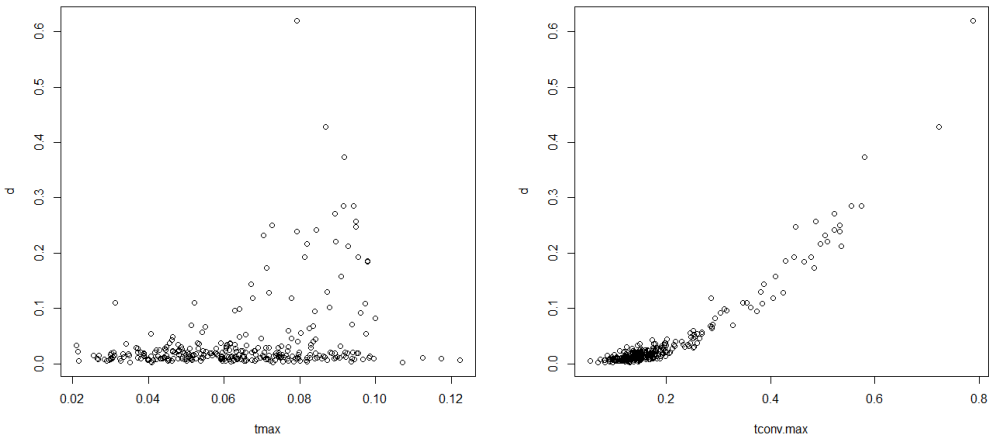

Figure 1: Distances d_i as a function of *tmax* (left) and *tconv.max* (right)

of distances for *tmax* $_i \leq 0.10$ are small, some distances are too large. By contrast, the distances are well approximated by a function of *tconv.max* $_i$, and for *tconv.max* $_i \leq 0.25$ all deviations are small. This supports the rule to require *tconv.max* $_i \leq 0.25$ or (on the safe side) *tconv.max* $_i \leq 0.20$ as a signal of convergence.

3.3 Some remarks about GMoM and sienacpp

The Generalized Method of Moments ('*GMoM*') as decribed in **?** was first implemented in *sienacpp()* , now also in *RSiena* .

The following indicates how some of the matrices in the description of the algorithm in **?** can be obtained from a *sienaFit* object *ans* produced by *sienacpp()* .

The dimension of the parameter is P = `ans$pp`, of the statistics Q = `ans$qq`, with $Q \geq P$.

First, let

```
fit <- ans$sienafit[[1]]
```

To get some understanding of what is in this object, look for `rifySienaFit` in file *RInterface.cpp* .

The matrix `z$sf = fit$phase3_statistics` is a sample in Phase 3 from the distribution of S^*. Hence Cov(S^*) is estimated by

```
cov0 <- cov(fit$phase3_statistics)
```

which is given as `ans$msf`. As indicated in (8) of **?**, the inverse $W = \big(\text{Cov}(S^*)\big)^{-1}$ is used :

```
W <- solve(cov0)
```

The $Q \times P$ matrix of derivatives Γ is estimated by `t(fit$gamma)`. For the matrix B we use `B1` as defined by

```
B0 <- t(fit$gamma) %*% W
B1 <- solve(diag(sqrt(rowSums(B0*B0)))) %*% B0
```

Hence the covariance matrix of $\hat{\theta}$ under the GMoM can be obtained as

```
D0 <- B1 %*% fit$gamma
D0inv <- solve(D0)
cov2 <- B1 %*% cov1 %*% t(B1)
covtheta <- D0inv %*% cov2 %*% t(D0inv)
```

For the GMoM, the t-ratios for convergence are calculated for the default (main) statistics for the effects. The maximum convergence ratio is calculated for all linear combinations of BS^*. Therefore, the maximum convergence ratio refers to the result of the estimation algorithm, and is not necessarily larger than the absolute values of all t-ratios for convergence. When some of the latter are large, this is an indication of poor model fit.

4 Statistics for MoM

The statistics used for the MoM are proposed in **?** and **?**.

4.1 Multi-group projects

In the following sections, statistics are often added for all periods from $m = 2$ to $m = M$. For multi-group data sets the situation is different. Such data sets can be implemented by 'glueing' the data sets as one sequence after each other. This is done by the function siena07GroupCreate. Another implementation is by considering the data set as nested, with periods nested in groups. In the sequential implementation, we can denote statistics for period m by S_m; in the nested implementation, statistics for period m in group g are denoted by S_{gm}. Denote the number of groups by G, and the number of waves for group g by M_g; and the cumulative sums of these by

$$C_g = \sum_{h=1}^{g} M_h \ .$$

In the sequential implementation, summations of the form

$$\sum_{m=2}^{M} S_m$$

should be replaced by

$$\sum_{g=1}^{G} \sum_{m=C_{g-1}+1}^{C_g} S_m \ .$$

In the nested implementation they should be replaced by

$$\sum_{g=1}^{G} \sum_{m=2}^{M_g} S_{gm} \ .$$

4.2 Rate function for networks

For an effect in the rate function

$$\lambda^X(\alpha^X, i, y) = \rho_m^X \exp\Big(\sum_k \alpha_k^X s_{ki}^X(y)\Big) \ , \tag{12}$$

the statistic for the method of moments to estimate α_k^X is

$$\sum_{m=2}^{M} \sum_i \left\{ s_{ki}^X\big(y(t_{m-1})\big) \sum_j \mid x_{ij}(t_m) - x_{ij}(t_{m-1}) \mid \right\}. \tag{13}$$

In the unconditional method of moments, the statistic to estimate ρ_m^X is

$$\sum_i \sum_j \mid x_{ij}(t_m) - x_{ij}(t_{m-1}) \mid \ . \tag{14}$$

In the conditional method of moments, this statistic is used for the stopping criterion for the simulations in period $m-1$.

4.3 Evaluation function for networks

For an effect in the evaluation function $s_{ik}^X(x,z)$, the change statistic or change contribution is defined by

$$\Delta_{kij}^X(x,z) = s_{ik}^X(x^{(+ij)},z) - s_{ik}^X(x^{(-ij)},z) \tag{15}$$

where $x^{(+ij)}$ is x to which the tie $i \to j$ has been added, and $x^{(-ij)}$ is x from which the tie $i \to j$ has been deleted. (Note that this definition implies that (**??**) is not affected by the value of x_{ij} in x as used in the left hand side.)

The quantity introduced in the beginning, called $\Delta f^N(r,i,j,x,z)$, is a linear combination of the change statistics:

$$\Delta f^N(r,i,j,x,z) = \pm \sum_k \beta_k^{X(r)} \Delta_{kij}^{X(r)}(x,z) \tag{16}$$

where $\beta_k^{X(r)}$ is the appropriate element of the parameter vector θ, and where $\pm = +1$ if the toggle means that tie $i \to j$ is added, while $\pm = -1$ if the toggle means that this tie is dropped.

In the C++ code, the change contribution is the function *calculateContribution()* , of which the basic instance is defined in *NetworkEffect()* and specific instances in all functions defining specific effects.

The statistic used for estimation, also called the target statistic, is defined as follows. To be explicit, denote all changing covariates (monadic and/or

dyadic) by v, with value $v(t_m)$ for wave m, and all constant covariates by w. The sum of the effect over all actors is defined by

$$s_k^X(x, z, v, w) = \sum_i s_{ik}^X(x, z, v, w) \ . \tag{17}$$

If there is only one network as dependent variable then there is no z so we can write $s_k^X(x, v, w)$. The target statistic then is

$$\sum_{m=2}^{M} s_k^X(x(t_m), v(t_{m-1}), w) \ . \tag{18a}$$

Note that v is taken at wave $m-1$, because for changing covariates the assumption is that the value observed at wave $m-1$ remains valid up to just before wave m.
If there are two dependent variables, one network and one behavior, then the target statistics for the network are

$$\sum_{m=2}^{M} s_k^X(x(t_m), z(t_{m-1}), v(t_{m-1}), w) \ . \tag{18b}$$

and for the behavior

$$\sum_{m=2}^{M} s_k^Z(x(t_{m-1}), z(t_m), v(t_{m-1}), w) \ . \tag{18c}$$

Note the cross-lagged way of using the waves in these equations. This is explained in **?**.

If the number of dependent variables is more than one for other configurations of dependent variables, like multiple networks or a network with multiple behaviors, the same cross-lagged principle is used: for wave $m \in \{2, \ldots, M\}$, the dependent variable is taken as observed in wave m and all explanatory ('independent') variables as observed in wave $m-1$.

In the C++ code, the summand in (**??**) is the function *evaluationStatistic()* . It is used in *StatisticCalculator.cpp* . It is defined in *NetworkEffect.cpp* and *BehaviorEffect.cpp* as the sum of *egoStatistic(i)* over all actors i, which are the terms in (**??**). For *NetworkEffect.cpp* , these are computed as

$$\text{egoStatistic}(i) = \sum_j x_{ij} \, \text{tieStatistic}(i, j) \ . \tag{19}$$

Note that these are virtual effects, and will be refined in specific network effects, which are descendants of class *networkEffect* . It is necessary to redefine either *egoStatistic* or *tieStatistic* .

For main effects, the redefined *tieStatistic* is not used if there also is a redefined *egoStatistic* . However, for network interaction effects of dyadic and ego effects, defined in Sections **??** and **??**, the *tieStatistics* are calculated as the products of the corresponding main effects (this happens in *NetworkInteractionEffect.cpp*). Therefore, to be able to use a network effect in interactions, it is necessary to redefine the *tieStatistic* .

The "number of distances equal to 2" effect is an example of an effect where decomposition (**??**) is not straightforward, and the *egoStatistic()* is defined using *initializeStatisticCalculation* and *cleanupStatisticCalculation* . The basic instances of all these functions also are defined in *NetworkEffect()* , and specific instances of *egoStatistic()* or *tieStatistic()* are defined in all functions defining specific effects.

A special case of the construction is given by so-called generic effects, see the documentation in *classdesign* .

Another construction uses the function *statistic()* ; see *networkEffect.cpp* :
'A convenience method for implementing statistics for both evaluation and endowment function. It assumes that the statistic can be calculated by iterating over ties (i, j) of a network Y and summing up some terms $s_{ij}(X)$ with respect to another network X, namely, $s(X, Y) = \sum_{(i,j)\in Y} s_{ij}(X)$. For evaluation function, $X = Y$. For endowment function, X is the initial network of the period, and Y is the network of ties that have been lost during the network evolution.'

In *networkEffect.cpp* , *creationStatistic()* uses *endowmentStatistic()* ; *endowmentStatistic()* uses *statistic()* ; *statistic()* uses *egoStatistic()* , and the latter uses *tieStatistic()* . If one of these is redefined in a given model or model class, then this chain is broken and the lower elements in the chain will be replaced by the redefinitions.

Preprocessing and postprocessing is possible using the functions *initializeStatisticCalculation()* and *cleanupStatisticCalculation()* . For examples, see *balanceEffect* . *InStructuralEquivalenceEffect* also is an example of interesting constructions in this respect.

The estimation statistics for the creation and endowment effects are calculated by applying the effect statistic to the network of gained ties, and the network of lost ties, respectively. This happens in functions *StatisticCalculator::calculateNetworkCreationStatistics* and *StatisticCalculator::calculateNetworkEndowmentStatistics* .

4.4 Actor statistics

StatisticCalculator::StatisticCalculator has an argument *returnActorStatistics* , with the result that all calls of *evaluationStatistic* , *endowmentStatistic* and *creationStatistic* also return the contributions by actor.

Functions *getTheActorStatistics* and *getActorStatistics* use *returnActorStatistics* =`TRUE`.

siena07setup.cpp has a non-exported function *getTargetActorStatistics* that is used in *getTargets* .

In *simstats.c* , if *returnActorStatistics* is present with value `TRUE` for the *sienaAlgorithm* object, the *actorStatistics* are returned as `ans[[10]]`. This possibility is not documented.

Currently, the *sienaAlgorithm* object never is given a component *returnActorStatistics* .

4.5 Interaction effects for networks

User-defined interactions for network change are defined as follows.

Consider two network effects $s_{ia}^X(x,z)$ and $s_{ib}^X(x,z)$. Denote their change statistics = change contributions by

$$\Delta_{aij}^X(x,z) \text{ and } \Delta_{bij}^X(x,z) \ .$$

Then the interaction is defined by the change contribution

$$\Delta_{a\times b,ij}^X(x,z) = \Delta_{aij}^X(x,z) \times \Delta_{bij}^X(x,z) \ . \tag{20}$$

For interactions between three effects it is analogous, with change statistic

$$\Delta^X_{a\times b\times c,ij}(x,z) = \Delta^X_{aij}(x,z) \times \Delta^X_{bij}(x,z) \times \Delta^X_{cij}(x,z) \ .$$

The question is, when does this makes sense, and what is the appropriate MoM estimation statistic. The change statistic and the estimation statistic must hang together according to (**??**) and (**??**), as mentioned above. For user-defined interactions to be properly defined by (**??**), we must indicate the estimation statistic to be used and prove that this satisfies (**??**) and (**??**).

Sufficient conditions for user-defined interactions are the following.

4.5.1 Dyadic effects

An effect is defined to be *dyadic* if it can be written as

$$s^X_{ik}(x,z) = \sum_j x_{ij}\, c_{kij}(x,z) \tag{21}$$

where $c_{kij}(x,z)$ is independent of x_{i*} defined as the row $x_{i*} = (x_{i1}, \ldots, x_{in})$. For a dyadic effect we have

$$\Delta^X_{kij}(x,z) = c_{kij}(x,z)$$

and

$$s^X_k(x,z) = \sum_{ij} x_{ij}\, c_{kij}(x,z) \ .$$

The interaction between two dyadic effects is defined by

$$s^X_{a\times b,i}(x,z) = \sum_j x_{ij}\, c_{aij}(x,z)\, c_{bij}(x,z) \ . \tag{22}$$

We then have

$$\begin{aligned}\Delta^X_{a\times b,ij}(x,z) &= c_{aij}(x,z)\, c_{bij}(x,z)\\ s^X_{a\times b}(x,z) &= \sum_i x_{ij}\, c_{aij}(x,z)\, c_{bij}(x,z) \ .\end{aligned}$$

This indeed satisfies (**??**) and (**??**). The same holds for interactions between three (or more) dyadic effects. One could say that the interaction between two dyadic effects is again a dyadic effect.

4.5.2 Ego effects

An effect is defined to be an *ego* effect if it can be written as

$$s_{ik}^X(x,z) = \sum_j x_{ij}\, c_{ki}(x,z) \tag{23}$$

where $c_{ki}(x,z)$ is independent of x_{i*} defined above, and independent of j (as the notation indicates). (This definition implies that an ego effect is also a dyadic effect.)
For an ego effect we have

$$\Delta_{kij}^X(x,z) = c_{ki}(x,z)$$

and

$$s_k^X(x,z) = \sum_{ij} x_{ij}\, c_{ki}(x,z)\ .$$

An interaction between an ego effect a and any effect b is defined by

$$s_{a\times b,i}^X(x,z) = c_{ai}(x,z)\, s_{ib}^X(x,z)\ . \tag{24}$$

This definition implies

$$\Delta_{a\times b,ij}^X(x,z) = c_{ai}(x,z)\, \Delta_{bij}^X(x,z)$$
$$s_{a\times b}^X(x,z) = \sum_{ij} c_{ai}(x,z)\, s_{ib}^X(x,z)\ .$$

Given that effect b satisfies (**??**) and (**??**), this interaction also satisfies (**??**) and (**??**). The same holds for interactions between two ego effects and any third effect, because the interaction between two ego effects is again an ego effect.

Whether effects are ego effects or not is defined separately on the R side and the C++ side; it is not transferred from R to C++. In C++, whether an effect is an ego effect is used only in *NetworkInteractionEffect.cpp* for the calculation of the estimation function. This uses the property that ego effects always have a *tieStatistic()* , and that this is independent of alter.

4.5.3 Contextual effects

Some effects that do not satisfy the conditions defining ego or dyadic effects are defined as 'ego' or 'dyadic' anyway, because they are meant to be used as contextual effects in interactions, where the 'context' is meant to represent the conditions under which ego makes the choice in the ministep. They are represented as elementary effects (see below). Such effects are indicated by the suffix "_ego" or "_dya" in the *shortName* .

4.5.4 Implementation

For the implementation of network interaction effects, see function class *NetworkInteractionEffect.cpp* . A dyadic or ego effect should be defined with a *tieStatistic()* rather than an *egoStatistic()* .

4.6 Interaction effects for behavior

Interaction effects can be defined for those behavior effects where the effect is defined as a product of the behavior itself (z_i) and something independent of z_i itself, although it may depend on the other dependent variables and even on z_j for $j \neq i$. In mathematical terms, for effects defined as

$$s^Z_{ik}(x, z) = z_i \, s^{Z0}_{ik}(x, z) \tag{25}$$

where $s^{Z0}_{ik}(x, z)$ is a function not depending on z_i. Most effects are of this kind; if I am right, all except for the quadratic tendency effect and effects involving similarity with respect to Z.

Note that this concerns effect for the same behavior variable Z. (I drop the index r here.)

For such effects, the change contribution is $s^{Z0}_{ik}(x, z)$ and the evaluation statistic is (**??**).

The interaction of two such effects, with indices k_1 and k_2 (where it is allowed that $k_1 = k_2$) is defined by the evaluation statistic

$$s^Z_{i(k_1 \circ k_2)}(x, z) = z_i \, s^{Z0}_{ik_1}(x, z) \, s^{Z0}_{ik_2}(x, z) \tag{26}$$

(I just used a circle $\circ$ to denote the interaction) and the change contribution

$$s_{ik_1}^{Z0}(x,z)\, s_{ik_2}^{Z0}(x,z)\ .$$

??????? RSiena is written in terms of an arbitrary allowed change. I have coded the change as

$$\text{difference} * s_{ik_1}^{Z0}(x,z)\, s_{ik_2}^{Z0}(x,z)\ .$$

I have I hope this is correct. I have multiplied together the terms for each ego and then divided by the value or difference until only one value or difference is retained.

For interactions between three effects it is just the same. The same effects qualify, the evaluation statistic is

$$s^{Z}_{i(k_1\circ k_2\circ k_3)}(x,z) = z_i\, s_{ik_1}^{Z0}(x,z)\, s_{ik_2}^{Z0}(x,z)\, s_{ik_3}^{Z0}(x,z)$$

and the change contribution

$$s_{ik_1}^{Z0}(x,z)\, s_{ik_2}^{Z0}(x,z\, s_{ik_3}^{Z0}(x,z)\ .$$

4.7 Effects: interactionType

The effects object has a variable (column) that is called `interactionType`. See the manual. For network effects, the interaction type is `"ego"`, `"dyadic"`, or `""` (blank); for behaviour effects, it is `"OK"` or `""`. This indicates, using the rules above, whether a interaction is allowed. This is checked in the internal RSiena function *fixUpEffectNames* . Since `"ego"` is stronger than `"dyadic"`, the former is used when the property is satisfied.

For elementary network effects (see the manual) the change statistics are not derived from an evaluation function. Therefore the requirement (**??**) does not apply. Elementary effects have `interactionType` set to ”dyadic”.

4.8 Weighted effects for behavior

Another concept, which is similar to interaction, is weighted effects. This is defined for effects involving a network X. Again I drop the index r.

The effect can be weighted by an actor variable; which can be an actor covariate as well as a dependent actor behavior (the same behavior as the one under consideration or a different one). Denote this variable by V. The weighting is carried out by multiplying each x_{ij} (but not other tie variables; only ties with i as a sender) by v_j; thus, in the interactions of the preceding subsection we are working with v_i but now with v_j. (The weighted indegree effect below is an exception to this general description.) This multiplication is done in the change contribution as well as the evaluation statistic.

Let me give the following examples (going through the list in the manual).

3. *weighted average similarity effect*, defined by the weighted average of centered similarity scores sim_{ij}^z between i and the other actors j to whom he is tied,

$$s_{i3\circ V}^{\mathrm{beh}}(x) = \frac{\sum_j v_j\, x_{ij}(\mathrm{sim}_{ij}^z - \widehat{\mathrm{sim}^z})}{\sum_j v_j\, x_{ij}} \;;$$

   (and 0 if $\sum_j v_j\, x_{ij} = 0$) ;

4. *weighted total similarity effect*, defined by the weighted sum of centered similarity scores sim_{ij}^z between i and the other actors j to whom he is tied,
   $s_{i4\circ V}^{\mathrm{beh}}(x) = \sum_j v_j\, x_{ij}(\mathrm{sim}_{ij}^z - \widehat{\mathrm{sim}^z})$;

5. *weighted indegree effect*,
   by an exception to the rule this could be defined as $s_{i5\circ V}^{\mathrm{beh}}(x) = z_i \sum_j v_j\, x_{ji}$;

6. *weighted outdegree effect*,
   $s_{i6\circ V}^{\mathrm{beh}}(x) = z_i \sum_j v_j\, x_{ij}$;

8. *weighted average similarity* $\times$ *reciprocity effect*, defined by the sum of centered similarity scores sim_{ij}^z between i and the other actors j to whom he is reciprocally tied,

$$s_{i8\circ V}^{\mathrm{beh}}(x) = \frac{\sum_j v_j\, x_{ij} x_{ji}(\mathrm{sim}_{ij}^z - \widehat{\mathrm{sim}^z})}{\sum_j v_j\, x_{ij} x_{ji}} \;;$$

(and 0 if $\sum_j v_j\, x_{ij} x_{ji} = 0$) ;

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

Let us omit the popularity-interaction effects, which already contain a weight.

14. *weighted average alter effect*, defined by the product of i's behavior multiplied by the average behavior of his alters (a kind of ego-alter behavior covariance),

$$s_{i14\circ V}^{\mathrm{beh}}(x) = z_i \, \frac{\sum_j v_j\, x_{ij}\, z_j}{\sum_j v_j\, x_{ij}}$$

(and the mean behavior, i.e. 0, if the ratio is 0/0) ;

15. *weighted average reciprocated alter effect*, defined by the product of i's behavior multiplied by the average behavior of his reciprocated alters,

$$s_{i15\circ V}^{\mathrm{beh}}(x) = z_i \, \frac{\sum_j v_j\, x_{ij}\, x_{ji}\, z_j}{\sum_j v_j\, x_{ij}\, x_{ji}}$$

(and 0 if the ratio is 0/0) ;

19. *weighted reciprocated degree effect*,
$s_{i19\circ V}^{\mathrm{beh}}(x) = z_i \sum_j v_j\, x_{ij}\, x_{ji}$.

4.9 Calculation of cross-lagged statistics in C++

For coevolution models, evaluation effects of one dependent variable on another dependent variable (such effects are called 'mixed effects') are estimated in the Method of Moments by cross-lagged statistics (**?**, formulae 26 and 27).

Evaluation effects are calculated in
`StatisticCalculator::calculateNetworkEvaluationStatistics`

and in the first part of
`StatisticCalculator::calculateBehaviorStatistics`.

In these functions, two states of the entire data set (with all dependent variables and covariates) are used:

`lpPredictorState` is the state of everything at the start of the wave;

`pCurrentLessMissingsEtc` is the state of everything at the end of the simulations, changed in a way that deals with missingness of data and structurally fixed values (this is documented in `missingsEtc.pdf`).

(Naming conventions in `siena\src` are that private variable names start with the letter `l` – for `local` – and names of pointer variables have the first or second (after `l`) letter `p`.)

In `StatisticCalculator::calculateNetworkEvaluationStatistics`, the simulated state of this dependent network in `pCurrentLessMissingsEtc` is used to replace the component of `lpPredictorState` corresponding to this dependent network. The modified `lpPredictorState` is then used to initialize all evaluation effects.

4.10 Contemporaneous statistics for GMoM

Note that the use of the variable `pSimulatedState` in the C++ code always is for the GMoM. It indicates that the simulated state instead of the preceding state should be used for the evaluation statistics.

Estimation of co-evolution models by the Generalized Method of Moments ('GMoM') operates as follows. For estimating the mixed effects, in which one dependent variable is being explained by another dependent variable, the Method of Moments uses cross-lagged estimation statistics, as discussed in the preceding section. **?** proposed a GMoM estimator in which the cross-lagged statistics are supplemented with contemporaneous statistics.

GMoM statistics are effects with `type=gmm`. Contemporaneous effects for co-evolution models all have shortNames ending in `_gmm`.

Computing the contemporaneous statistics is achieved in `StatisticCalculator::calculateNetworkGMMStatistics` by initializing the `gmm` effects using function `effect::initialize` with five parameters (for other effects they have four), adding parameter `State * pSimulatedState`. This initialization is prepared in classes `NetworkEffect` and `BehaviorEffect` by defining `initialize` as virtual functions

```
void NetworkEffect::initialize(const Data * pData,
     State * pState, State * pSimulatedState,
     int period, Cache * pCache)
```

and

```
void BehaviorEffect::initialize(const Data *pData,
     State *pState, State *pSimulatedState,
     int period, Cache *pCache)
```

A similar construction is used for generic effects in class `GenericNetworkEffect`.

These virtual functions `initialize` then are redefined for the contemporaneous `gmm` statistics by making `effect::evaluationStatistic` depend on the contemporaneous instead of the cross-lagged statistics. This is done, for the currently implemented contemporaneous `gmm` statistics, in effect classes `CovariateDependentNetworkEffect`, `NetworkDependentBehaviorEffect`, and `NetworkAlterFunction`. Because of the inheritance property this needs to be done only once in each chain of classes, and will work for all effect classes inheriting from these three.

The actual contemporaneous `gmm` statistics then are defined by modifying the corresponding effects of `type=eval` such that a constructor is added with the parameter `bool simulatedState`, and the five-parameter version of `effect::initialize` is added. The same is done for functions defining generic effects.

Examples of this can be seen in effect classes `CovariateEgoEffect`, `CovariateSimilarityEffect`, and `OutTieFunction`. These then are specified as `gmm` statistics in `effectFactory` by using them with `simulatedState=true`.

5 Rate effects

Rate effects are currently of three types, defined by the rateType column of `allEffects.csv`. The values are ”structural”, ”diffusion”, ”covariate”, or NA.

Function *DependentVariable::initializeRateFunction* in *Dependentvariable.cpp* initializes the rate function; here also the functions *model/effects/StructuralRateEffect.h* and *model/effects/DiffusionRateEffect.h* are used.

Functions *StatisticCalculator::calculateNetworkRateStatistics* and *StatisticCalculator::calculateBehaviorRateStatistics* in *StatisticCalculator.cpp* calculate the rate functions. In addition to calculating rate statistics, RSiena also computes scores for rate effects. These scores are calculated in functions such as *getStatistics* in `siena07internals.cpp` and in methods of *DependentVariable* and *EpochSimulation* .

The `RateX` effect is defined in a funny way. If you look for the string ”RateX”, it occurs nowhere in the cpp directory. It is identified by being the only rate effect of type ”covariate”, together with the dependent variable name and the explanatory variable, given by the parameter ”interaction1”.

5.1 Diffusion rate effects

Diffusion rate effects can be added and modified more easily than other rate effects, as they are implemented as a distinct class, *DiffusionRateEffect* , which inherits from *BehaviorRateEffect* . The method *DiffusionRateEffect::calculateRateContribution* returns the value used in the calculation of rates, scores, and statistics in both *DependentVariable* and *StatisticCalculator* , each of which creates its own instances of diffusion rate effects.

The contribution to the rate function is calculated by *DiffusionRateEffect::proximityValue* , which must be defined in each diffusion rate class (or is otherwise pure virtual). To create a new *DiffusionRateEffect* , one should define a new class inheriting from *DiffusionRateEffect* and implement the mathematical definition of the rate effect in the *proximityValue* method. New diffusion rate effects must also be explicitly

constructed and initialized in *DependentVariable::initializeRateFunction* and *StatisticCalculator::calculateBehaviorRateStatistics* . Additionally, they must be registered as known diffusion rate effects in the *getStatistics* function in `siena07internals.cpp`.

Further refactoring of all rate effects and their construction and storage may be possible to make the code more uniform and easier to maintain.

6 Likelihood-based calculations: Chain structures

This algorithm follows the definitions in **?**. The notation also is taken from that paper.

The basic data structure for likelihood-based calculations is called a *chain*. This is a sequence of changes that can take one ('observed') value of y to a next one.

To allow later generalization to valued networks as easily as possible, we define a condition D (for dichotomous) that is defined on the level of variables (networks or behavioral variables); in our current system D is *True* for networks and *False* for behavioral variables, but this can be different in future uses.

One change is called a *ministep* , denoted *ms*, and is defined as:

$$ms = (w, i, j, r, d, pred\ , succ\ , lOptionSetProb\ , lChoiceProb\ , rRate\ ) \tag{27}$$

where

w ('aspect') = 'network' or 'behavior' (abbreviated to $N - B$);
i ('actor') = actor if w = B, sending actor if w = N;
j ('actor') = meaningless 0 if w = B, receiving actor if w = N;
r ('variable number') = number of variable ($1 \leq r \leq R_w$);
d ('difference') = meaningless 0 if D, amount of change if not D (where D depends on w, r);
currently we require $d \in \{-1, 0, 1\}$, but at some later moment exceptions to this rule may be allowed;
pred ('predecessor') = pointer to preceding ministep;
succ ('successor') = pointer to next (succeeding) ministep;
lOptionSetProb ('log OptionSet probability')
= log probability of making a ministep of this OptionSet, where the OptionSet is defined below as (w, i, r);
lChoiceProb ('log choice probability')
= log probability of making a ministep of this choice, where the choice is (j, d), given that (w, i, r);

rRate ('reciprocal rate') = reciprocal of aggregate (summed) rate function immediately before this ministep.

To indicate the components/fields of a ministep we use the notation *ms*.w, *ms*.i, etc.
The precise definitions of *lOptionSetProb* , *lChoiceProb* , and *rRate* are given below in the specification of function *StepProb* . The values (w, i, j, r, d) may also be called the coordinates of the ministep.

In Siena 3, d, *pred* and *succ* are called *difh*, *predh* and *such*; and the program uses rates instead of reciprocal rates, but this was implemented only very incompletely anyway.

The ministep is practically the same as what is called a microstep in Section **??**, but used here in a more precise way. These words are not intentionally different.
The log probability and reciprocal rate depend not only on the chain and the ministep, but also on the initial state y or $y(t_{m-1})$ valid before the start of the chain; and on the model specification and model parameters. Their computation is done by procedure *StepProb* described in Section **??**.

The interpretation is that a ministep operates on (i.e., changes) outcome y as implemented by the following function.

1. *ChangeStep*$(y,$ *ms*$)$ transforms state y as follows,
   where *ms* $= (w, i, j, r, d, ...)$;
   This can also be denoted *ChangeStep*$(y, (w, i, j, r, d))$;
   - if w = N and $i \neq j$, change $N_{ij}^{(r)}$ to $1 - N_{ij}^{(r)}$;
   - if w = B, change $B_i^{(r)}$ to $B_i^{(r)} + d$.

The inverse operation is very simple:
in general, *Inverse*$\big($*ChangeStep*$(y, (w, i, j, r, d))\big)$ = *ChangeStep*$(y, (w, i, j, r, -d))$;
in particular, *Inverse*$\big($*ChangeStep*$(y, (N, i, j, r, 0))\big)$ = *ChangeStep*$(y, (N, i, j, r, 0))$.
The definition of *ChangeStep* implies that only those values of d are allowed that do not lead $B_i^{(r)}$ outside of the bounds of this variable. I think this should not always be checked except perhaps for in a test phase, but the creation and transformation of ministeps should contain checks that ensure this condition.

ChangeStep is called a lot, and it will be helpful that it is implemented in a very fast way.

The *Option* of a ministep is defined as (Network, i, j, r) for Network ministeps, and as (Behavior, i, r) for Behavior ministeps. This defines the variable changing by the ministep. Recall that j is meaningless for $w = B$. In general, we can define the options of a ministep as (w, i, j, r). *Option* is called 'kind' in Siena 3.

The *OptionSet* of a ministep is defined as (w, i, r). This defines the choice situation / option set for the ministep.

This definition also means that network ministeps with $i = j$ and behavior ministeps with $d = 0$ have no effect on the outcome. Such ministeps are permitted, and are called *diagonal* ministeps.

A *chain* from observation $y(t_{m-1})$ to observation $y(t_m)$ is a sequence of ministeps $ms_1, ms_2, ..., ms_T$ which, when applied sequentially, transform $y(t_{m-1})$ into $y(t_m)$. We then say that the chain *connects* $y(t_{m-1})$ to $y(t_m)$. For M observations, therefore, we require a sequence of $M - 1$ chains.

For a sequence of ministeps $ms_1, ms_2, ..., ms_T$ we define the following functions. For disregarded values of the ministep (depending on whether it is a N or B ministep) we use the wildcard symbol *.

2. *NetworkNumber*$(i, j, r, S) = \sharp\{s \mid 1 \leq s \leq S, \textit{Option}(ms_s) = (N, i, j, r)\}$.
   In words, this is the number of ministeps, up to and including ministep number S, which imply a change in tie variable (i, j) for Network r.

3. *BehSum*$(i, B, r, S) = \Sigma_{s=1}^{S} (ms_s.d_s)\, I\{\textit{Option}(ms_s) = (B, i, *, r)\}$
   where I is the indicator function defined as $I(A) = 1$ if A is *True* and 0 if A is *False* .
   In words, this is the partial sum, ending at ministep number S, of the d (difference) values of all ministeps by actor i for Behavior r.

If the outcomes $y(t_{m-1})$ and $y(t_m)$ are completely defined (without any missing data) then the requirements on this sequence are as follows.

Networks : (since changes are defined as toggles)
For all i, j, r with $1 \leq r \leq R_N$, $i \neq j$,

$$N_{ij}^{(r)}(t_{m-1}) = N_{ij}^{(r)}(t_m) \;\Leftrightarrow\; \textit{NetworkNumber}(i, j, N, r, T) \text{ is even} \;; \tag{28a}$$

which is equivalent to

$$N_{ij}^{(r)}(t_{m-1}) = 1 - N_{ij}^{(r)}(t_m) \;\Leftrightarrow\; \textit{NetworkNumber}(i, j, N, r, T) \text{ is odd .} \tag{28b}$$

Behavior : (since changes are defined as increments)
For all i, r with $1 \leq r \leq R_B$,

$$B_i^{(r)}(t_{m-1}) + \textit{BehSum}(i, B, r, T) = B_i^{(r)}(t_m) \tag{29}$$

and

$$1 \leq B_i^{(r)}(t_{m-1}) + \textit{BehSum}(i, B, r, S) \leq \textit{maxbeh}_r \text{ for all } 1 \leq S < T. \tag{30}$$

For each option there is a missingness indicator

$$\textit{mis}(w, i, j, r)$$

which is *True* or *False* , depending on whether in at least one of the two end points of the chain, $y(t_{m-1})$ or $y(t_m)$, the corresponding variable $N_{ij}^{(r)}$ or $B_i^{(r)}$ is missing. The use of these indicators is that restrictions (**??**) and (**??**) are not required for the missing data. For missing behavior data, however, condition (**??**) still is required to ensure that the variable remains within range.

4. The number of Network options with missing values is defined as

$$\textit{NumMisNet} = \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} I\{\textit{mis}(N, i, j, r)\} \;,$$

   where $I\{\textit{True}\} = 1$ and $I\{\textit{False}\} = 0$.

5. The number of Behavior options with missing values is defined as

$$\textit{NumMisBeh} = \sum_{i=1}^{n} I\{\textit{mis}(B, i, *, r)\} \;.$$

It must be noted that missing data are not handled in the best possible way in the likelihood-based procedures in Siena 3, and this is done differently here. Therefore, results for likelihood-based procedures in Siena 3 and RSiena will be different.

Classes of functions are required which do the following:

1. Create and transform chains.
2. Calculate probabilities related to chains.
3. Store chains: read from and write to file.

7 Likelihood-based calculations: Create and transform chains

7.1 Data types

1. Ministep. See (**??**).
   The *Option* of a ministep is (Network, i, j, r) for Network ministeps, and (Behavior, i, r) for Behavior ministeps. Note that this defines the variable that is being changed by the ministep.
   Note that in Siena 3 this is called the *rKind* (*restricted Kind*), and the 'Kind' there also includes the value d. A ministep *ms* is *diagonal* if it is of Option (Network, i, j, r) with $i = j$, or of Option (Behavior, i, r) with *ms*.d = 0.

2. Chain. This is a sequence of ministeps connected by the pointers *pred* and *succ* , with a *first* and *last* element. The *first* and *last* elements are dummies, i.e., they are of a special Option and OptionSet (*Extreme*, $0, 0, 0$) which implies no change:
   ChangeStep(y, *first*) = *ChangeStep*(y, *last*) = y.
   Section **??** on structurally fixed values gives an exception to this rule, however, for the *last* element.
   The *first* and *last* elements are used just to have handles for the start and end of the chain. Of course, *first* .*pred* =*last* .*succ* = nil.
   Or perhaps it is more convenient to define *first* .*pred* = *first* and *last* .*succ* = *last* .
   The *first* and *last* ministeps are not *diagonal*.

   The connection implies that if ms_a and ms_b are two ministeps with ms_b.*pred* $= ms_a$, then ms_a.*succ* $= ms_b$. The *first* element has a nil *pred* , and the *last* element has a nil *succ* .

7.2 Functions

In Siena 3, I have defined the ministep type with various other pointers and attributes useful for navigating in the chain. These are functions of the chain, and including them in the ministep type is for the purpose of computational efficiency. These functions are the following. They are defined as functions of the ministep in a given chain. They are not important in themselves, but might be useful for updating the variables relating to CCPs (see below).

1. *nrOption*. The total number of ministeps in the chain of the same Option.

2. *predOption* . Pointer to the last earlier ('preceding') ministep of the same Option, and *first* if such a ministep does not exist.
   Called *predhrkind* in Siena 3.

3. *succOption* . Pointer to the first later ('succeeding') ministep of the same Option and *last* if such a ministep does not exist.
   Called *suchrkind* in Siena 3.

The chain defines an order relation (binary function) of ministeps in an obvious way, representing the time order in which the ministeps take place. When there may be the possibility of confusion, this is called the *chain order*.

1. $ms_a < ms_b$ if there is a sequence $ms_1, ..., ms_K$ ($K \geq 0$) of ministeps such that $ms_a.succ = ms_1, ms_b.pred = ms_K$, and $ms_k.succ = ms_{k+1}$ for all k, $1 \leq k \leq K-1$.

2. For $ms_a < ms_b$, we denote by $length(ms_a, ms_b)$ the number of ministeps from ms_a to ms_b, including these end points, which is the value $K+2$ according to the preceding definition.

3. For $ms_a < ms_b$, we denote by $[ms_a, ms_b]$ the interval of ministeps from ms_a to ms_b, i.e., the sequence $ms_a, ms_1, ..., ms_K, ms_b$ of the definition in (1).
   The interval $[ms_a, ms_a]$ is defined as the ministep ms_a.

4. In the obvious way, we define recursively

$$ms.succ^{\,0} = ms \tag{31}$$

$$ms.succ^{\,k+1} = \big(ms.succ^{\,k}\big).succ \ . \tag{32}$$

Thus, $length(ms, ms.succ^{\,k}) = k + 1$ for $k \geq 0$.

An ordered pair of ministeps (ms_a, ms_b) with $ms_a < ms_b$ is called a CCP (*consecutive canceling pair*) if they are of the same Option, not *diagonal*, cancel each other's effect (see next sentence), have no other ministep of the same Option in between, and there is at least one ministep of a different Option in between (i.e., $length(ms_a, ms_b) \geq 3$), and *neither ministep is missing at start or end of the interval*. Two non-diagonal ministeps ms_a and ms_b cancel each other's effect if the following hold: either they are both of the same Option (Network, i, j, r) (then they cancel because they toggle the same binary variable), or both are of the same Option (Behavior, i, r) and $ms_a.d + ms_b.d = 0$.

For example, if the chain contains a total of three ministeps ms_a, ms_b, ms_c of the Option (Network, 1, 2, 1), with $ms_a < ms_b < ms_c$, and none of which are each others' immediate predecessors/successors, then (ms_a, ms_b) and (ms_b, ms_c) are CCP's.

The reason for this definition is to use it later in defining changes in the chain, such that each change has a unique (i.e., exactly one) inverse operation. Adding a CCP to a chain will not lead to violations of (**??**, **??**), although it may lead to violation of (**??**), which therefore must be separately checked. There is a one-to-one correspondence between the set of all operations of dropping a CCP from a chain, and the set of all operations of adding the two elements of a CCP to the chain as immediate predecessors of two ministeps $ms_a < ms_b$ for which ms_a is not the *first* element, and there is no ministep ms_c with $ms_a < ms_c < ms_b$ of the same *Option* as ms_a and ms_b, and which do not lead to violation of (**??**). All this is elaborated later, and given here only as a motivation for this definition.

Basic functions of the chain are the following.
Since these are frequently used, they should be stored and updated when the chain is changed; this is done in the *Update* function.
The condition *SimpleRates* is defined below.

1. *TotNumber*, the number of ministeps of the chain, excluding the *first* ;
   so an empty chain consisting only of the *first* and *last* ministeps with *first* .*succ* = *last* has *TotNumber* $= 1$.

2. *DiagNumber*, the number of *diagonal* ministeps of the chain.

3. *CCPNumber*, the number of CCP's in the chain.

4. *ChainNumMisNet*, the number of ministeps of some *Option* (N, i, j, r) for which *mis*(N, i, j, r) is true.

5. *ChainNumMisBeh*, the number of ministeps of some *Option* $(B, i, *, r)$ for which *mis*$(B, i, *, r)$ is true.

6. *ChainNumInitMis*, the number of *Options* (w, i, j, r) for which the initial value $N_{ij}^{(r)}(t_{m-1})$ or $B_i^{(r)}(t_{m-1})$ is a missing value.

7. Used only if (not *SimpleRates*):
   mu = $\sum_{s=2}^{T-1}$ *ms*$_s$.*rRate* , where $T =$ *TotNumber*.
   Note that the sum is over all ministeps in the chain except the two extremes (*first* and *last*).

8. Used only if (not *SimpleRates*):
   sigma2 = $\sum_{s=2}^{T-1}$(*ms*$_s$.*rRate*)^2.

The chain can be denoted by

ms$_0$, *ms*$_1$, *ms*$_2$, . . . , *ms*$_{TotNumber}$,

in which *ms*$_0$ and *ms*$_{TotNumber}$ are the extreme elements.

The following functions may be defined on the Options of ministeps: They are not important in themselves, but might be useful for updating the variables relating to CCPs.

1. *NumberOption*(w, i, j, r), the number of ministeps of the chain of Option (w, i, j, r).
   For the network ministeps this will be a sparse matrix, in the sense that for large networks most of the values *NumberOption*(N, i, j, r) will be 0.
   This is called *NumberrKind* in Siena 3.

2. *Multiple*(w,i,j,r) = *True* if *NumberOption*$(w,i,j,r) \geq 2$ and *False* if *NumberOption*$(w,i,j,r) \leq 1$.
   We say that a Option can be multiple or non-multiple.

7.3 Operations

Basic operations on chains are the following. Of course they have to guarantee the consistency of all the derived variables and pointers. The consistency of the log probabilities and reciprocal rates is treated separately (when it is needed), see Section **??**.

1. *Create* an empty chain consisting only of the elements (*first* , *last*).

2. *InsertBefore*(*ms*$,w,i,j,d,r$) :
   for a currently existing ministep *ms* $\neq$ *first* , insert the ministep with values (w,i,j,d,r) between *ms.pred* and *ms*.

3. *Delete* a ministep, and link up its predecessor and successor.

4. *RandomElement* :
   draw a random ministep from the chain, excluding the *first* element; note that the probabilities are 1/*TotNumber*.

5. *RandomElementNotAfter(ms)* :
   draw a random ministep from the chain, among the elements after the *first* element up to and including ministep *ms*;

6. *Connect* : construct randomly a chain that connects two outcomes $y(t_{m-1})$ and $y(t_m)$.
   This is done by repeatedly applying *RandomElement* and *InsertBefore*. If there are no *higher* , *disjoint* , or *atleastone* relations between the various different networks (in particular if $R_N = 1$), the following procedure can be used.

   For all R_N networks:
   For all $(i,j), i \neq j$:
   if $N_{ij}^{(r)}(t_{m-1}) \neq N_{ij}^{(r)}(t_m)$,
   then *InsertBefore* (*RandomElement*, $N,i,j,0,r$);

For all R_B behaviors:

For all i:

Define $D = B_i^{(r)}(t_m) - B_i^{(r)}(t_{m-1})$;
if $D > 0$, then D times *InsertBefore*(*RandomElement*, $B, i, 0, 1, r$);
if $D < 0$, then $-D$ times *InsertBefore*(*RandomElement*, $B, i, 0, -1, r$).

However, if there are any *higher* , *disjoint* , or *atleastone* relations in force, then inserting the changes at randomly chosen places can lead to violations of these requirements. Inserted network changes that might lead to such violations therefore must take account of earlier inserted changes for the same (i, j). In such cases the connections for the networks can be made by the following procedure. With 'incompatible' is referred to the incompatibility because of the *higher* , *disjoint* , or *atleastone* requirements.

For r running from 1 to R_N :

For all $(i, j), i \neq j$:

if $N_{ij}^{(r)}(t_{m-1}) \neq N_{ij}^{(r)}(t_m)$, define by $\mathcal{R}$ the set of networks r', $1 \leq r' < r$, for which $N_{ij}^{(r)}(t_m)$ is incompatible with $N_{ij}^{(r')}(t_{m-1})$.
(For these r' it must hold that $N_{ij}^{(r')}(t_{m-1}) \neq N_{ij}^{(r')}(t_m)$, because $N_{ij}^{(r)}(t_m)$ must be compatible with $N_{ij}^{(r')}(t_m)$;
therefore the following requires the change in option $(N, i, j, 0, r)$ to take place before the changes in options $(N, i, j, 0, r')$ for $r' \in \mathcal{R}$.)
Let *ms* be the first ministep among all inserted ministeps of option (N, i, j, r') for $r' \in \mathcal{R}$;
then *InsertBefore* (*RandomElementNotAfter(ms)*, $N, i, j, 0, r$).

The *Connect* procedure yields a chain connecting the two outcomes $y(t_{m-1})$ and $y(t_m)$ which has minimum length.

The following random draws are not always possible, since the sets from which a random element is drawn, may be empty. (It may be noted, however, that usually the set will be non-empty.) *RandomMultipleOption* and *RandomCCPOption* are dropped! This will be simpler and probably at least as efficient as Siena 3.

7. *RandomDiagonal* :
   draw a random *diagonal* ministep from the chain; note that the probabilities are 1/*DiagNumber*.

8. *RandomCCP* :
   draw a random CCP (ms_a, ms_b) from the chain; note that the probabilities are 1/*CCPNumber*.

9. *RandomMisNet* :
   draw a random ministep ms_a from the chain of which the *Option* (w, i, j, r) satisfies $w = N$ and *mis*(N, i, j, r).
   Note that the probabilities are 1/*ChainNumMisNet*.

10. *RandomMisBeh* :
    draw a random ministep ms_a from the chain of which the *Option* (w, i, j, r) satisfies $w = B$ and *mis*$(B, i, *, r)$.
    Note that the probabilities are 1/*ChainNumMisBeh*.

11. *RandomInitMis* :
    draw a random *Option* (w, i, j, r) for which the initial value $N_{ij}^{(r)}(t_{m-1})$ or $B_i^{(r)}(t_{m-1})$ is a missing value.
    Note that the probabilities are 1/*ChainNumInitMis*.

8 Likelihood-based calculations: Calculate probabilities related to chains

An important special case is the case of state-constant rate functions, i.e., rate functions $\lambda^W(r, i, y)$ depending only on W, r, and i but not on y, and therefore not changing as a consequence of the simulations. This is important also because the majority of users will use state-constant rate functions. In the case of state-constant rate functions, everything related to λ needs to be calculated only when parameters are changed.
Denote this by the Boolean *ConstantRates*.

We define a special Boolean condition *SimpleRates*. The default is to let *SimpleRates* = *ConstantRates*, but this may be changed by the user (not in the gui). In practice it will be changed only for the purposes of algorithm comparison.

Ministeps are interpreted as changes in the chain (procedure *ChangeStep*). These changes are made with certain probabilities, and the rate of change has a certain value when the ministep is going to be made. The probabilities and rates depend on the state immediately before the ministep; this depends in turn on the state at the start of the chain, and the sequence of ministeps before the current ministep. For a ministep *ms* in the chain with a given initial state y (say, $y = y(t_{m-1})$), the state obtaining before *ms* can be defined recursively as follows (where *ChangeStep* is treated as a function with states as outcomes).

1. *StateBefore*(*first*) $= y$.
2. *StateBefore*(*ms.succ*) = *ChangeStep*(*StateBefore*(*ms*), *ms*).

Thus, the state before *ms* is obtained by repeatedly applying *ChangeStep*:

$$\textit{StateBefore}(\textit{ms}) = \textit{ChangeStep}^{(\textit{length}(\textit{first}\ ,\textit{ms})\ -\ 1)}(y_{\text{initial}})$$

(where the superscript indeed means raising the operator to a power, i.e., executing it repeatedly.)

The log-probabilities and rates are defined by the following procedure. For the mathematical symbols $\pi_j, \pi_v, \lambda(...), J_m$, see the notation of Section **??**

where the microstep/ministep is treated for the purpose of simulation of the model, and where the same ingredients are used.

The functions *StepProb1* to *StepProb3* are often used one after each other, and utilizing this will lead to efficiency gains.

1. *StepProb1*(input y, w, i, r; output $rr, lospr$);
   this calculates the aggregate rate for current state y and returns rr as its reciprocal:

   $$rr \leftarrow 1/\lambda^{+}(+, +, y) \ .$$

   it calculates the probability of getting the *OptionSet* (w, i, r) and returns it as $lospr$:

   $$lospr \leftarrow \log\left(\frac{\lambda^{w}(r, i, y)}{\lambda^{+}(+, +, y)}\right)$$

   If *ConstantRates* these are trivial look-up operations (the values then depend only on the parameters included in the functions λ, not on y).

   Note that in some cases this must be done for all (w, i, r), sometimes only for one value of (w, i, r). When it is done for all cases, this is denoted
   StepProb1$(y, *; rr, lospr*)$,
   and then $lospr*$ is an output array of suitable dimensions.

2. *StepProb2*(input y, w, i, j, r, d; output $rr, lospr, lcpr$);
   After doing the same as in *StepProb1*$(y, w, i, r; rr, lospr)$,
   this calculates for the current state y, conditional on the assumption that a ministep of OptionSet (w, i, r) is made, the log of the conditional probability that this will be the ministep with value (w, i, j, r, d);
   for $w = N$ (network) this is $lcpr \leftarrow \log(\pi_j)$ using π_j defined in (**??**);
   for $w = B$ (behavior) this is $lcpr \leftarrow \log(\pi_d)$ using π_v defined in (**??**).
   Output variables $rr, lospr, lcpr$ are the *rRate* , *lOptionSetProb* , and *lChoiceProb* of this ministep.

   Note that for a given (w, i, r), in some cases this must be done for all j (if $w = N$) or d (if $w = B$), in other cases only for one value of j or d. If it is done for all j or d, the notation is
   StepProb2$(y, w, i, *, r, *; rr, lospr, lcpr*)$

and again $lcpr*$ is an array of suitable dimension.
If the function is denoted *StepProb2*$(y, ms; rr, lospr, lcpr)$, this is the same as *StepProb2*$(y, w, i, j, r, d; rr, lospr, lcpr)$ with the coordinates w, i, j, r, d for the ministep filled in.

3. *StepProb3*(input y, w, i, j, r, d; output rr, sc);
   This calculates rr as in *StepProb1*$(y, w, i, r; rr, lospr)$;
   in addition, it calculates the contribution of this ministep to the score function, which in Section **??** is what is added (as described there for three occasions) to J_m, and stores this in the p-vector sc. Output $lospr$ and $lcpr$ dropped from *StepProb3* because not needed.

While operating on the chain, it is important to keep the log probabilities and rates up to date. This requires the following procedure. It updates only part of the chain, and is applied when it is known that the earlier and later parts do not need to be updated.

4. *Update*(*ms*) for a ministep *ms*:
   Update *TotNumber*, *DiagNumber*, *CCPNumber*, *ChainNumMisNet*, *ChainNumMisBeh*.
   Use *StepProb2* to update the log probabilities and rates for ministep *ms*.
   If not *SimpleRates*: Update the values of *mu* and *sigma2*.

5. *Update*(ms_a, ms_b) for ministeps $ms_a < ms_b$:
   Update *TotNumber*, *DiagNumber*, *CCPNumber*, *ChainNumMisNet*, *ChainNumMisBeh*.
   Use *StepProb2* to update the log probabilities and rates for all ministeps from ms_a to ms_b (i.e., all ministeps between these two in the chain order, including these two ministeps themselves).
   If not *SimpleRates*: Update the values of *mu* and *sigma2*.

   This is called *UpdateRateslprobs* in Siena 3.

In many cases, *StepProb** has been called just before *Update*, so that the log probabilities and rates are known already and the expensive procedure *StepProb** does not have to be called again.

Depending on the implementations, the auxiliary variables for working with CCPs must also be suitably updated.

9 Likelihood-based calculations: Metropolis-Hastings steps

A basic required functionality is to simulate from the distribution of chains that connect $y(t_{m-1})$ to $y(t_m)$, given the model specification and model parameters. This is done by repeated application of Metropolis Hastings steps. These are of the following types, with associated probabilities. The probabilities are constants with default values that can be changed by the very experienced user.

1. *MH_InsertDiag*
   (called *MH_TryInsertDiag* in Siena 3), associated probability *pridg*.
2. *MH_CancelDiag*
   (called *MH_TryCancelDiag* in Siena 3), associated probability *prcdg*.
3. *MH_Permute*, associated probability *prper*.
4. *MH_InsPermute*, associated probability *pripr*.
5. *MH_DelPermute*, associated probability *prdpr*.
6. *MH_InsMis*, associated probability *prims*.
7. *MH_DelMis*, associated probability *prdms*.
8. *MH_RandomMis* dropped version May 30.

Function *MH_DelPermute* also uses internal probabilities *prmin* and *prmib*. If the number of actors always is n, and all networks are one-mode, these could have the default values

$$\mathit{prmin} = \frac{\mathit{NumMisNet}}{\mathit{NumMisNet} + R_N\, n(n-1)} \tag{33}$$

$$\mathit{prmib} = \frac{\mathit{NumMisBeh}}{\mathit{NumMisBeh} + R_B\, n} \tag{34}$$

In the general case, $R_N\, n(n-1)$ would be replaced by the total number of dyadic tie variables, and $R_B\, n$ by the total number of individual behaviour variables. (The reasoning is as follows. In procedure *MH_DelPermute* this

probability serves to balance changes in missings with changes in CCPs, and the total available number of variables/options for which there could be CCPs is $R_N\, n(n-1)$ and $R_B\, n$, respectively).

The definitions of these procedures have three parts:

A. Choose the proposal.

B. Calculate the probability (usually *pra*) for this proposal.

C. With probability *pra* carry it out in practice.

The earlier (version before February 1) description only contained parts B and C. What were the input parameters for those earlier versions now are calculated in part A, and therefore now only have an internal role. The only remaining input parameter is c_0, the maximal order of the permutations.

Functions *MH_InsertDiag* and *MH_CancelDiag* are each other's inverses. Similarly, *MH_InsPermute* and *MH_DelPermute* are each other's inverses. Function *MH_Permute* is the inverse of another *MH_Permute*, for a suitable other permutation. Functions *MH_InsMis* and *MH_DelMis* are each other's inverses.

Function *MH_Permute* basically is part of the two functions *MH_InsPermute* and *MH_DelPermute*. Including it in those functions is done for computational efficiency (most of the calculations have to be done anyway).

The variable below called *KappaFactor* denotes the factor with which the variable called κ in **?** (equation (16) for the *SimpleRates* case, and (21) else) has to be multiplied if the *MH* step is accepted.
The correspondence for the *SimpleRates* case is as follows:
$n\alpha_1(t_2 - t_1)$ in (16) is $\lambda^+(+,+,y) = 1/rr$ here;
R in (16) is *TotNumber* $- 1$ here.

As notation I use the R convention of denoting an assignment statement by $a \leftarrow b$, i.e., the variable a gets the value b.

9.1 Diagonal Insert

The function *MH_InsertDiag*(output *pra, accept*) is roughly described as follows. The interpretation is that the proposal is made to insert a *diagonal* element of *OptionSet* (w, i, r) immediately before a ministep *ms*; according to a random decision with probability *pra*, computed within the function, this proposal is put into effect (yielding *accept* = *True*) or not (yielding *accept* = *False*).

Part *A*:

1. *ms* $\leftarrow$ *RandomElement*
2. $y \leftarrow$ *StateBefore*(*ms*)
3. *StepProb1*$(y, *;$ output $rr, lospr*)$
4. With probabilities defined by $\exp(lospr*)$ choose *OptionSet* (w, i, r). If i is not active, or $w = B$ and $B^{(r)}(i)$ is structurally fixed, then exit.

Note: the proposal probability here is

$$\frac{\exp(lospr)}{\textit{TotNumber}}$$

which is used below in the definition of *pra*.

Part *B*:

9. *StepProb2*$(y, w, i, i, r, 0; rr, lospr, lcpr)$ ($lospr$ was already calculated above)
10. If *SimpleRates*, let

$$\textit{KappaFactor} \leftarrow \frac{1}{rr \times \textit{TotNumber}} \tag{35}$$

else

$$\textit{KappaFactor} \leftarrow \sqrt{\frac{\textit{sigma2}}{\textit{sigma2} + rr^2}} \times \exp\left(\frac{(1-\textit{mu})^2}{2 \times \textit{sigma2}} - \frac{(1-\textit{mu}-rr)^2}{2\,(\textit{sigma2} + rr^2)}\right) . \tag{36}$$

11.

$$\textit{pra} \leftarrow \textit{KappaFactor} \times \exp(lcpr) \times \frac{\textit{TotNumber} \times \textit{prcdg}}{(\textit{DiagNumber} + 1) \times \textit{pridg}} \tag{37}$$

Check the use of $lospr$ and $lcpr$, which may be different from the earlier version.
Note: the proposal probability and the new chain probability both include factors $\exp(lospr)$ which cancel out.
if (*pra* > 1), then *pra* $\leftarrow 1$.

Part *C*:

12. With probability *pra* let *accept* $\leftarrow$ *True* , else *accept* $\leftarrow$ *False* .

13. If *accept*, then

    (a) *InsertBefore*(*ms*, $w, i, i, r, 0$)
    Earlier the order was wrong: it said $w, i, i, 0, r$ instead of $w, i, i, r, 0$.

    (b) *Update*(*ms*)

9.2 Diagonal Delete

The function *MH_CancelDiag*(output *pra, accept*) is roughly described as follows.
The interpretation is that the proposal is made to delete a *diagonal* mini-step *ms*; according to a random decision with probability *pra*, computed within the function, this proposal is put into effect (yielding *accept* = *True*) or not (yielding *accept* = *False*).

Part *A*:

1. *ms* $\leftarrow$ *RandomDiagonal*

Note: the proposal probability here is

$$\frac{1}{\textit{DiagNumber}}$$

which is used below in the definition of *pra*.

Part *B*:

14. $rr \leftarrow$ *ms.rRate*

15. If *SimpleRates*, let

$$\textit{KappaFactor} \leftarrow rr \times (\textit{TotNumber} - 1) \tag{38}$$

else

$$\textit{KappaFactor} \leftarrow \sqrt{\frac{\textit{sigma2}}{\textit{sigma2} - rr^2}} \times \exp\left(\frac{(1-\textit{mu})^2}{2 \times \textit{sigma2}} - \frac{(1-\textit{mu}+rr)^2}{2\,(\textit{sigma2} - rr^2)}\right) . \tag{39}$$

16.

$$\textit{pra} \leftarrow \textit{KappaFactor} \times \exp(-\textit{ms.lChoiceProb}\,) \times \frac{\textit{DiagNumber} \times \textit{pridg}}{(\textit{TotNumber} - 1) \times \textit{prcdg}} . \tag{40}$$

Note: the proposal probability and the new chain probability both include factors $\exp(\textit{ms.lOptionSetProb}\,)$ which cancel out.
if (*pra* > 1), then *pra* $\leftarrow 1$.

Part *C*:

17. With probability *pra* let *accept* $\leftarrow$ *True* , else *accept* $\leftarrow$ *False* .
18. If *accept*, then
    (a) *Delete*(*ms*)
    (b) If (not *SimpleRates*), let
        i. *mu* $\leftarrow$ *mu* $- rr$;
        ii. *sigma2* $\leftarrow$ *sigma2* $- rr^2$.

9.3 Permute

In this section, various formulations are changed but the main content is the same; from the next section to Section **??**, the text is new. This is not indicated by colour any more.

A rough description of the function
MH_Permute(input c_0; output *pra, accept*) is as follows.
The input parameter c_0 is a relatively small integer – in Siena 3 it is determined adaptively with a maximum value of 40.
Within the function, if *ms*$_a$.*succ* $^{(c_0-1)} <$ *last* then $c = c_0$, else c is truncated to *length*$[$*ms*$_a$, *last* $] - 1$; and *ms*$_b$ = *ms*$_a$.*succ* $^{(c-1)}$. Thus, *ms*$_a$ and *ms*$_b$ are two non-extreme ministeps with *ms*$_a <$ *ms*$_b$ and $c =$ *length*$[$*ms*$_a$, *ms*$_b]$ $\leq c_0$.
Further, *perm* is a permutation of the numbers $1, 2, \ldots, c$.
The proposal made is to permute the c ministeps in the interval $[$*ms*$_a$, *ms*$_b]$ by *perm*; according to a random decision with probability *pra*, computed within the function, this proposal is put into effect (yielding *accept* = *True*) or not (yielding *accept* = *False*).

Part *A*:

1. repeat *ms*$_a \leftarrow$ *RandomElement* until *ms*$_a \neq$ *last* .
2. $c \leftarrow \min\{c_0,$ *length*(*ms*$_a$, *last* $) - 1\}$.
   If $c = 1$ then exit.
3. Let *perm* be a random permutation of the numbers 1 to c,
   and denote *ms*$_b$ = *ms*$_a$.*succ* $^{(c-1)}$.

4. $y \leftarrow$ *StateBefore*(ms_a)

5. For all $r, 1 \leq r \leq R_B$, check (**??**) for the permuted chain.
   This condition needs to be checked here only for ministeps from ms_a to ms_b.
   For any variable (w, r) that is involved in a condition of the kind *higher* , *disjoint* , or *atleastone* , similarly check these conditions for the potential new chain.
   If at least one of these conditions are not satisfied, exit.

The inverse of the proposal is a proposal of exactly the same kind. The proposal probability is

$$\frac{1}{(\mathit{TotNumber} - 1) \times c!}$$

but this needs not be used, since the proposal probability is the same as the probability of the inverse proposal.

Part *B*:

6.

$$\mathit{sumlprob} \leftarrow \sum_{s=1}^{c} (ms_a.\mathit{succ}^{\,s-1}).(\mathit{lChoiceProb} + \mathit{lOptionSetProb}) \; ;$$

   If (not *SimpleRates*), then below we use the values *mu* and *sigma2*; these refer to the current chain.

$$\mathit{sumlprob\_new} \leftarrow 0 \; ;$$

$$\mathit{mu\_new} \leftarrow \mathit{mu} - \sum_{s=1}^{c} (ms_a.\mathit{succ}^{\,s-1}).(\mathit{rRate}) \; ;$$

$$\mathit{sigma2\_new} \leftarrow \mathit{sigma2} - \sum_{s=1}^{c} (ms_a.\mathit{succ}^{\,s-1}).(\mathit{rRate})^2 \; ;$$

7. Note that still $y =$ *StateBefore*(ms_a) as was assigned above.
   For $1 \leq s \leq c$ denote by *Coordinates*$_s$ the values (w, i, j, r, d) of $ms_a.\mathit{succ}^{\,s-1}$.
   For s running from 1 to c, do:

(a) *StepProb2*$(y,$ *Coordinates*$_{perm(s)}; rr_s, lospr_s, lcpr_s)$;

(b) *sumlprob_new* $\leftarrow$ *sumlprob_new* $+ lcpr_s + lospr_s$;

(c) If (not *SimpleRates*), then

i. *mu_new* $\leftarrow$ *mu_new* $+ rr_s$;

ii. *sigma2_new* $\leftarrow$ *sigma2_new* $+ (rr_s)^2$;

(d) *ChangeStep*$(y,$ *Coordinates*$_{perm(s)})$;

8. If *SimpleRates*, let

$$\textit{KappaFactor} \leftarrow 1 \tag{41}$$

else

$$\textit{KappaFactor} \leftarrow \sqrt{\frac{\textit{sigma2}}{\textit{sigma2\_new}}} \times \exp\left(\frac{(1-\textit{mu})^2}{2 \times \textit{sigma2}} - \frac{(1-\textit{mu\_new})^2}{2 \times \textit{sigma2\_new}}\right) . \tag{42}$$

9.

$$\textit{pra} \leftarrow \textit{KappaFactor} \times \exp(\textit{sumlprob\_new} - \textit{sumlprob}) \tag{43}$$

if (*pra* > 1), then *pra* $\leftarrow 1$.

Part *C*:

10. With probability *pra* let *accept* $\leftarrow$ *True* , else *accept* $\leftarrow$ *False* .

11. If *accept*, then permute the chain from *ms*$_a$ to *ms*$_b$ by *perm*, and *Update*$($*ms*$_a$, *ms*$_b)$.

9.4 Insert – Permute

The function *MH_InsPermute* (input c_0; output *misdat, pra, accept*) is defined as follows.
The input parameter c_0 is a relatively small integer – in Siena 3 it is determined adaptively with a maximum value of 40.

First a rough description is given.

First a vector of coordinates $(w_0, i_0, j_0, r_0, d_0)$ is selected. The output variable *misdat* indicates whether option (w_0, i_0, j_0, r_0) is missing (see function *mis* defined above).

In the regular case, where *misdat* = *False* , the proposal is made to insert the non-diagonal coordinates $(w_0, i_0, j_0, r_0, d_0)$ before a random ministep ms_a and insert $(w_0, i_0, j_0, r_0, -d_0)$ before some ministep ms_b with $ms_a < ms_b$, such that the two inserted ministeps will be a CCP. This requires the following:
► **(C1)** $\Big((w_0 = N) \Rightarrow i_0 \neq j_0\Big)$ and $\Big((w_0 = B) \Rightarrow d_0 \neq 0\Big)$;
► **(C2)** there are no ministeps of the type (w_0, i_0, j_0, r_0) in the interval $[ms_a, ms_b.pred\,]$.

Since we must be calculating various probabilities anyway, we shall use the opportunity also to propose permuting an interval $[ms_a, ms_e]$ of ministeps; the length of this interval is c. However, we do not wish to risk and create extra CCPs by doing so, as this would require more complicated counting for the calculation of acceptance probabilities. Therefore, first we have a provisional value of c; if the interval of c ministeps starting from ms_a contains two ministeps of the same *Option*, then c is decreased to a value such that it is certain that permuting the interval of this length starting from ms_a will not affect the number of CCPs. This truncation of c uses the auxiliary ministep ms_f.
If $c \geq 2$, *perm* is a permutation of the numbers $1, 2, \ldots, c$, and the proposal includes permuting the ministeps in the interval $[ms_a, ms_a.succ^{\,c-1}]$ by *perm*.

If *misdat* = *True* and *the link is missing at the end of the period*, the proposal is made to insert the non-diagonal ministep $(w_0, i_0, j_0, r_0, d_0)$ before a random ministep ms_a, and permute the c ministeps starting with ms_a, where again c is c_0 truncated to the number of available places.

If *misdat* = *True* and *the link is not missing at the end of the period* do nothing.

According to a probability *pra*, the proposal is put into effect (yielding *accept* = *True*) or not (yielding *accept* = *False*).

The steps taken in the function are as follows.

Part *A*:

1. Repeat $ms_a \leftarrow$ *RandomElement* until $ms_a <$ *last* .

2. $y \leftarrow$ *StateBefore*(ms_a)

3. *StepProb1*$(y, *; rr, lospr*)$
   With probabilities defined by $\exp(lospr*)$ choose *OptionSet* (w_0, i_0, r_0), under the condition that $(w_0, i_0, r_0) \neq$ *OptionSet*(ms_a).
   We just choose one and if same var and actor as ms_a we exit. Not sure how to express it in your notation. I set pr_2 to 1 below because of this
   If i_0 is not active, exit.
   If *uponly* or *downonly* holds for variable (w_0, r_0), then exit.
   $pr_2 \leftarrow 1 - \exp(lospr*($*Option*($ms_a$)$))$.
   By $lospr * ($*Option*(ms_a)$)$ is denoted the element of the array $lospr*$ giving the log-probability of the *OptionSet* of the ministep ms_a.

4. *StepProb2*$(y, w_0, i_0, *, r_0, *; rr, lospr, lcpr*)$
   With probabilities defined by $\exp(lcpr*)$:
   if $w_0 = N$ choose j_0 and let $d_0 = 0$, if $w_0 = B$ choose d_0 and let $j_0 = 0$.
   If $(w_0, i_0, j_0, r_0, d_0)$ is diagonal then exit.
   If $w_0 = N$ and $N^{(r_0)}(i_0, j_0)$ is structurally fixed, or $w_0 = B$ and $B^{(r_0)}(i_0)$ is structurally fixed, then exit.
   If $w_0 = B$ and the chain after inserting $(w_0, i_0, j_0, r_0, d_0)$ before ms_a would not satisfy (**??**) any more at ms_a, then exit.

   Denote the log-probability of the realized choice by $lcpr$;
   note that also $lospr$ now is the log-probability of the option set choice realized in the preceding step, so it was already calculated earlier.

5. *misdat* $\leftarrow$ *mis*(w_0, i_0, j_0, r_0)

6. (a) If (not *misdat*):

      i. Let ms_d be
      the first ministep in the chain after ms_a of *Option* (w_0, i_0, j_0, r_0);
      or the *last* ministep if there is no ministep after ms_a of this *Option*.
      This can be determined using the pointer *succOption* .

ii. *ChoiceLength* $\leftarrow$ *length*$(ms_a, ms_d) - 1$
Condition (iii) omitted

(Note that not going beyond this ministep ms_d, and ms_a having an *OptionSet* different from (w_0, i_0, r_0), implies that **(C2)** will be satisfied.)

(b) If *misdat*:
If ministep is not missing at the end of the period, exit. Otherwise, *ChoiceLength* $\leftarrow 1$

7. (a) If (not *misdat*):
Let ms_b a random ministep in the interval $[ms_a.succ\ , ms_d]$. Note that the number of choices here is *ChoiceLength*.

(b) If *misdat*:
$ms_b \leftarrow$ *last*

8. (a) *ThisLength* $\leftarrow$ *length*$(ms_a, ms_b) - 1$

(b) $c \leftarrow \min\{c_0,$ *ThisLength*$\}$
(Note that $c \geq 1$; if $c = 1$, then the permutation applied below is trivial, and the permutation as well as the checks involved can be skipped, because they have no effect.)

(c) If *ThisLength* $\leq c_0$,
then $ms_g \leftarrow ms_b.pred$,
else $ms_g \leftarrow ms_a.succ^{\,c}$.

(d) If the interval $[ms_a, ms_g]$ contains any pair of two non-diagonal ministeps of the same *Option*:

i. define ms_f as the last ministep in $[ms_a, ms_g]$ such that all *Options* of non-diagonal ministeps in $[ms_a, ms_f]$ are distinct;

ii. $c \leftarrow \min\{c,$ *length*$([ms_a, ms_f]) - 1\}$.
(The permutation below will then affect only ministeps strictly before ms_f; this will ensure that the permuted ministeps, together with ms_f, all have distinct *Options* and therefore the permutation cannot affect the number of CCPs.)

(e) $ms_e \leftarrow ms_a.succ^{\,(c-1)}$

9. Let *perm* be a random permutation of the numbers 1 to c.

10. For all variables (w, r) involved in a condition of the kind *higher* , *disjoint* , or *atleastone* , check these conditions for the chain changed as follows:
$(w_0, i_0, j_0, r_0, d_0)$ inserted before ms_a,
if (not *misdat*): $(w_0, i_0, j_0, r_0, -d_0)$ inserted before ms_b,
and the interval $[ms_a, ms_e]$ permuted according to *perm*.
This needs to be checked here only for ministeps from ms_a to ms_b.
If at least one of these conditions are not satisfied, exit.

The proposal probability is

$$\frac{\exp(lospr + lcpr)}{pr_2 \times (\textit{TotNumber} - 1) \times \textit{ChoiceLength} \times (c!)}$$

This is used in *pra* below.
We also need the probability of the inverse proposal (except for the factor $c!$ which cancels), and therefore calculate the following.

11. This item is used only if not *misdat* :
if *NumberOption*$(w_0, i_0, j_0, r_0) = 0$:

$$\textit{NewCCPNumber} \leftarrow \textit{CCPNumber} + 1 \ ;$$

if *NumberOption*$(w_0, i_0, j_0, r_0) \geq 1$ and $w_0 = N$:

(a) if ms_a.*pred* in the original chain (before the insertion) is of *Option* (w_0, i_0, j_0, r_0) then

$$\textit{NewCCPNumber} \leftarrow \textit{CCPNumber} + 1 \ ,$$

(b) else

$$\textit{NewCCPNumber} \leftarrow \textit{CCPNumber} + 2 \ ;$$

if *NumberOption*$(w_0, i_0, j_0, r_0) \geq 1$ and $w_0 = B$:

if { there is a ministep of option (w_0, i_0, j_0, r_0) before ms_a,
and the last such ministep is not ms_a.*pred* },
define d_- as the d-coordinate of the last such ministep,
and $d_- \leftarrow 0$ otherwise;
if there is a ministep of option (w_0, i_0, j_0, r_0) after ms_b,
define d_+ as the d-coordinate of the first such ministep,
and $d_+ \leftarrow 0$ otherwise; and let

$$\textit{NewCCPNumber} \leftarrow \textit{CCPNumber} + I\{d_- \times d_0 = -1\} + I\{d_+ \times d_0 = +1\}$$

where $I\{A\} = 1$ if A is true and 0 otherwise.

We find NewCCPnumber by insertions, calculate pr1 and then remove the insertions

12.

$$\text{if (not } \textit{misdat}) : \qquad pr_1 \leftarrow \frac{1 - \textit{prmin} - \textit{prmib}}{\textit{NewCCPNumber}} \ ;$$

$$\text{if } \textit{misdat} \text{ and } w_0 = N : \qquad pr_1 \leftarrow \frac{\textit{prmin}}{\textit{ChainNumMisNet} + 1} \ ;$$

$$\text{if } \textit{misdat} \text{ and } w_0 = B : \qquad pr_1 \leftarrow \frac{\textit{prmib}}{\textit{ChainNumMisBeh} + 1} \ .$$

(only the non-*misdat* case was changed).

Part *B*:

13.

$$\textit{sumlprob} \leftarrow \sum_{s=1}^{\textit{ThisLength}} (ms_a.\textit{succ}^{\,s-1}).(\textit{lChoiceProb} + \textit{lOptionSetProb}) \ ;$$

If (not *SimpleRates*), then below we use the values *mu* and *sigma2*; these refer to the current chain.

$$\textit{sumlprob\_new} \leftarrow 0 \ ;$$

$$\textit{mu\_new} \leftarrow \textit{mu} - \sum_{s=1}^{\textit{ThisLength}} (\textit{ms}_a.\textit{succ}^{\,s-1}).(\textit{rRate}\,)\ ;$$

$$\textit{sigma2\_new} \leftarrow \textit{sigma2} - \sum_{s=1}^{\textit{ThisLength}} (\textit{ms}_a.\textit{succ}^{\,s-1}).(\textit{rRate}\,)^2\ ;$$

14. Note that still $y = \textit{StateBefore}(\textit{ms}_a)$ as was assigned above.
StepProb2$(y, w_0, i_0, j_0, r_0, d_0; rr_0, lospr_0, lcpr_0)$;
sumlprob_new $\leftarrow$ *sumlprob_new* $+ lcpr_0 + lospr_0$;
If (not *SimpleRates*), then

    (a) *mu_new* $\leftarrow$ *mu_new* $+ rr_0$;

    (b) *sigma2_new* $\leftarrow$ *sigma2_new* $+ (rr_0)^2$.

    ChangeStep$(y, (w_0, i_0, j_0, r_0, d_0))$

15. For $1 \leq s \leq$ *ThisLength* denote by *Coordinates*$_s$ the values (w, i, j, r, d) of *ms*$_a$.*succ* $^{s-1}$.

16. For s running from 1 to c, do:

    (a) *StepProb2*$(y,$ *Coordinates*$_{\textit{perm}(s)}; rr_s, lospr_s, lcpr_s)$;

    (b) *sumlprob_new* $\leftarrow$ *sumlprob_new* $+ lcpr_s + lospr_s$;

    (c) If (not *SimpleRates*), then

        i. *mu_new* $\leftarrow$ *mu_new* $+ rr_s$;

        ii. *sigma2_new* $\leftarrow$ *sigma2_new* $+ (rr_s)^2$;

    (d) *ChangeStep*$(y,$ *Coordinates*$_{\textit{perm}(s)})$.

17. For s running from $c + 1$ to *ThisLength* do:

    (a) *StepProb2*$(y,$ *Coordinates*$_s; rr_s, lospr_s, lcpr_s)$

    (b) *sumlprob_new* $\leftarrow$ *sumlprob_new* $+ lcpr_s + lospr_s$

    (c) If (not *SimpleRates*), then

        i. *mu_new* $\leftarrow$ *mu_new* $+ rr_s$

        ii. *sigma2_new* $\leftarrow$ *sigma2_new* $+ (rr_s)^2$

    (d) *ChangeStep*$(y,$ *Coordinates*$_s)$

18. if (not *misdat*):

(a) *StepProb2*$(y, w_0, i_0, j_0, r_0, -d_0; rr_0, lospr_0, lcpr_0)$

(b) *sumlprob_new* $\leftarrow$ *sumlprob_new* $+ lcpr_0 + lospr_0$

(c) If (not *SimpleRates*), then

i. *mu_new* $\leftarrow$ *mu_new* $+ rr_0$;

ii. *sigma2_new* $\leftarrow$ *sigma2_new* $+ (rr_0)^2$.

(d) *ChangeStep*$(y, (w_0, i_0, j_0, r_0, -d_0))$
Note that at this point, y has been transformed to *StateBefore*(ms_b) of the current ('old') chain.

19. If *SimpleRates*, then

(a) if (not *misdat*):

$$\textit{KappaFactor} \leftarrow \frac{1}{rr^2 \times \textit{TotNumber} \times (\textit{TotNumber} + 1)}$$

(b) if *misdat*:

$$\textit{KappaFactor} \leftarrow \frac{1}{rr \times \textit{TotNumber}}$$

(This was changed 30-05-10; earlier. the lines were, erroneously, as follows:)

(a) if (not *misdat*):

$$\textit{KappaFactor} \leftarrow \frac{1}{rr^2 \times (\textit{TotNumber} + 1) \times (\textit{TotNumber} + 2)}$$

(b) if *misdat*:

$$\textit{KappaFactor} \leftarrow \frac{1}{rr \times (\textit{TotNumber} + 1)}$$

else (i.e., if not *SimpleRates*)

$$\textit{KappaFactor} \leftarrow \sqrt{\frac{\textit{sigma2}}{\textit{sigma2\_new}}} \times \exp\left(\frac{(1 - \textit{mu})^2}{2 \times \textit{sigma2}} - \frac{(1 - \textit{mu\_new})^2}{2 \times \textit{sigma2\_new}}\right) . \tag{44}$$

20.

$$\textit{pra} \leftarrow \textit{KappaFactor} \times \exp(\textit{sumlprob\_new} - \textit{sumlprob}) \tag{45}$$

$$\times \frac{prdpr \times pr_1 \times pr_2 \times (\textit{TotNumber} - 1) \times \textit{ChoiceLength}}{pripr \times \exp(lospr + lcpr)} \tag{46}$$

if ($\textit{pra} > 1$), then $\textit{pra} \leftarrow 1$.

Part *C*:

21. With probability *pra* let *accept* $\leftarrow$ *True* , else *accept* $\leftarrow$ *False* .
22. If *accept*, then
    (a) insert $(w_0, i_0, j_0, r_0, d_0)$ before $\textit{ms}_a$;
    (b) if (not *misdat*), insert $(w_0, i_0, j_0, r_0, -d_0)$ before $\textit{ms}_b$;
    (c) permute the chain from ms_a to $\textit{ms}_a.\textit{succ}^{\,c-1}$ by *perm*;
    (d) *Update* the chain for the changed part.

9.5 Delete – Permute

The function *MH_DelPermute* (input c_0; output *misdat,* w_0*, pra, accept*) is defined as follows.
The input parameter c_0 is a relatively small integer – in Siena 3 it is determined adaptively with a maximum value of 40.

First a rough description is given.

At the start, there is a choice between deleting a ministep for a missing data variable, reflected by *misdat* = *True* ; or deleting a CCP, reflected by *misdat* = *False* . Although this seems to be quite different, still it has been combined in one procedure because the overlap in the missing and non-missing cases is so large.

If *misdat* = *False* (the regular case), the proposal is made to delete two ministeps $\textit{ms}_a$ and $\textit{ms}_b$ that together are a CCP.
Then the number c is provisionally $\min\{c_0, \textit{length}(\textit{ms}_a, \textit{ms}_b) - 2\}$.

If *misdat* = *True* , the proposal is made to delete one ministep for a missing data variable, ms_a.
Then provisionally $c = \min\{c_0, \textit{length}(ms_a, \textit{last}\ ) - 2\}$.
Like in *MH_InsPermute*, a check is made for the existence of several ministeps of the same *Option* in the interval of c ministeps after ms_a; if this is the case, then c is decreased just like in *MH_InsPermute*.

In both cases, the output parameter w_0 is the aspect (Network or Behavior) of ministep ms_a. The reason for having this output is the possibility to tune the values of the probabilities *prmin* and *prmib*.

In addition, if $c \geq 2$, *perm* is a permutation of the numbers $1, 2, \ldots, c$, and the proposal includes permuting the ministeps in the interval $[ms_a.succ\ , ms_a.succ^{\,c}]$ by *perm*.

According to aprobability *pra*, the proposal is put into effect (yielding *accept* = *True*) or not (yielding *accept* = *False*).

The steps taken in the function are as follows.

Part *A*:

1. With probability *prmin* + *prmib* let *misdat* $\leftarrow$ *True* ,
   else *misdat* $\leftarrow$ *False* .
2. (a) if (not *misdat*):
      i. If *CCPNumber* = 0, then exit.
      ii. $(ms_a, ms_b) \leftarrow$ *RandomCCP*
      iii. *ThisLength* $\leftarrow$ *length*(ms_a, ms_b)
   (b) if *misdat*:
      i. With probability
      $$\frac{\textit{prmin}}{\textit{prmin} + \textit{prmib}}$$
      let $w_0 \leftarrow N$, else $w_0 \leftarrow B$.
      ii. if $w_0 = N$, then
         A. if *ChainNumMisNet* = 0, then exit.
         B. $ms_a \leftarrow$ *RandomMisNet*
      iii. if $w_0 = B$, then

A. if *ChainNumMisBeh* = 0, then exit.

B. $ms_a \leftarrow$ *RandomMisBeh*

iv. *ThisLength* $\leftarrow$ *length*$(ms_a, last\ )$

v. If $ms_a.succ\ = last$ (i.e., *ThisLength* = 2), exit.

3. $c \leftarrow \min\{c_0,$ *ThisLength* $- 2\}$.
   Note that in the case (not *misdat*) the definition of a CCP implies $c \geq 1$.
   If *misdat*, the 'exit' in the preceding step implies that $c \geq 1$.

4. If ((*ThisLength* $- 2 \leq c_0$) and (not *misdat*)),
   then $ms_g \leftarrow ms_b.pred$,
   else $ms_g \leftarrow ms_a.succ^{\,(c+1)}$.
   (ms_g is an upper bound to the interval if ministeps that will be permuted.)

5. If the interval $[ms_a.succ\ , ms_g]$ contains any pair of two non-diagonal ministeps of the same *Option*:

   (a) define ms_f as the last ministep in $[ms_a.succ\ , ms_g]$ such that all *Options* of non-diagonal ministeps in $[ms_a.succ\ , ms_f]$ are distinct;

   (b) $c \leftarrow \min\{c,$ *length*$([ms_a.succ\ , ms_f]) - 1\}$.

   (The permutation below will then affect only ministeps strictly before ms_f; this will ensure that the permuted ministeps, together with ms_f, all have distinct *Options* and therefore the permutation cannot affect the number of CCPs.)

6. Let *perm* be a random permutation of the numbers 1 to c.

7. $y \leftarrow$ *StateBefore*(ms_a) .

8. For all variables (w, r) involved in a condition of the kind *higher* , *disjoint* , or *atleastone* , check these conditions for the chain with ms_a and ms_b deleted, and with $ms_a.succ$ to $ms_a.succ^{\,c}$ permuted by *perm*. This needs to be checked here only for ministeps from $ms_a.succ$ to $ms_b.pred$.
   If at least one of these conditions are not satisfied, exit.

To calculate the proposal probabilities of this proposal and the inverse proposal, used in pra below, we need to calculate the following.

9.

$$\text{if (not } \textit{misdat}) : \ pr_1 \leftarrow \frac{1 - \textit{prmin} - \textit{prmib}}{\textit{CCPNumber}} \ ;$$

$$\text{if } \textit{misdat} \text{ and } w_0 = N : \ pr_1 \leftarrow \frac{\textit{prmin}}{\textit{ChainNumMisNet}} \ ;$$

$$\text{if } \textit{misdat} \text{ and } w_0 = B : \ pr_1 \leftarrow \frac{\textit{prmib}}{\textit{ChainNumMisBeh}} \ .$$

10. If (not *misdat*):

    (a) let $\textit{ms}_d$ be
    the first ministep in the chain after $\textit{ms}_b$ of the same *Option* (w_0, i_0, j_0, r_0);
    or the *last* ministep if there is no ministep after $\textit{ms}_b$ of this *Option*.
    This can be determined using the pointer *succOption* .

    (b) Let *ChoiceLength* $\leftarrow$ *length*$(\textit{ms}_a, \textit{ms}_d) - 3$.
    Note that *ChoiceLength*≥ 1.
    The information we need in the following is only *ChoiceLength*, we can forget about $\textit{ms}_d$ itself.

11. If *misdat*, *ChoiceLength* $\leftarrow 1$.

The probability of this proposal is

$$\frac{pr_1}{c!} \ .$$

Part *B*:

12. If *misdat* then *MaxLength* $\leftarrow c$, else *MaxLength* $\leftarrow$ *ThisLength*.

13.

$$\textit{sumlprob} \leftarrow \sum_{s=1}^{\textit{MaxLength}} (\textit{ms}_a.\textit{succ}^{\,s-1}).(\textit{lChoiceProb} + \textit{lOptionSetProb}) \ ;$$

If (not *SimpleRates*), then below we use the values *mu* and *sigma2*; these refer to the current chain.

$$\textit{sumlprob\_new} \leftarrow 0 \ ;$$

$$\textit{mu\_new} \leftarrow \textit{mu} - \sum_{s=1}^{\textit{MaxLength}} (\textit{ms}_a.\textit{succ}^{\,s-1}).(\textit{rRate}) \ ;$$

$$\textit{sigma2\_new} \leftarrow \textit{sigma2} - \sum_{s=1}^{\textit{MaxLength}} (\textit{ms}_a.\textit{succ}^{\,s-1}).(\textit{rRate})^2 \ ;$$

14.

$$lpr_0 \leftarrow \textit{ms}_a.(\textit{lChoiceProb} + \textit{lOptionSetProb})$$

(This is part of the log probability for the reverse proposal.)

15. Note that still $y = \textit{StateBefore}(\textit{ms}_a)$ as was assigned above.

$$(w_2, i_2, j_2, r_2, d_2) \leftarrow (\textit{ms}_a.\textit{succ}).\textit{Coordinates} \ ;$$

$$\textit{StepProb1}(y, w_2, i_2, r_2; rr_2, lospr_2) \ ;$$

$$pr_2 \leftarrow 1 - \exp(lospr_2)$$

pr_2 *is set to 1 as for insert permute.* (pr_2 also is part of the log probability for the reverse proposal; the others of these numbers with subscript 2 are not used any more.)

16. For $1 \leq s \leq \textit{ThisLength} - 2$ denote by $\textit{Coordinates}_s$ the values (w, i, j, r, d) of $\textit{ms}_a.\textit{succ}^{\,s}$.

17. Note that still $y = \textit{StateBefore}(\textit{ms}_a)$ as was assigned above.
For s running from 1 to c, do:

    (a) $\textit{StepProb2}(y, \textit{Coordinates}_{\textit{perm}(s)}; rr_s, lospr_s, lcpr_s)$

(b) *sumlprob_new* $\leftarrow$ *sumlprob_new* $+ lcpr_s + lospr_s$

(c) If (not *SimpleRates*), then

i. *mu_new* $\leftarrow$ *mu_new* $+ rr_s$

ii. *sigma2_new* $\leftarrow$ *sigma2_new* $+ (rr_s)^2$

(d) *ChangeStep*$(y,$ *Coordinates*$_{perm(s)})$

18. For s running from $c+1$ to *ThisLength* -2 do:

(a) *StepProb2*$(y,$ *Coordinates*$_s; rr_s, lospr_s, lcpr_s)$

(b) *sumlprob_new* $\leftarrow$ *sumlprob_new* $+ lcpr_s + lospr_s$

(c) If (not *SimpleRates*), then

i. *mu_new* $\leftarrow$ *mu_new* $+ rr_s$

ii. *sigma2_new* $\leftarrow$ *sigma2_new* $+ (rr_s)^2$

(d) *ChangeStep*$(y,$ *Coordinates*$_s)$;
note that at this point, if (not *misdat*), y has been transformed to *StateBefore*(ms_b) of the current ('old') chain.

19. If *SimpleRates*, then

(a) if (not *misdat*):

$$\textit{KappaFactor} \leftarrow rr^2 \times (\textit{TotNumber} - 1) \times (\textit{TotNumber} - 2)$$

(b) if *misdat*:

$$\textit{KappaFactor} \leftarrow rr \times (\textit{TotNumber} - 1)$$

(This was changed 30-05-10; earlier. the lines were, erroneously, as follows:)

(a) if (not *misdat*):

$$\textit{KappaFactor} \leftarrow rr^2 \times \textit{TotNumber} \times (\textit{TotNumber} - 1)$$

(b) if *misdat*:

$$\textit{KappaFactor} \leftarrow rr \times \textit{TotNumber}$$

else (i.e., if not *SimpleRates*)

$$\textit{KappaFactor} \leftarrow \sqrt{\frac{\textit{sigma2}}{\textit{sigma2\_new}}} \times \exp\left(\frac{(1-\textit{mu})^2}{2 \times \textit{sigma2}} - \frac{(1-\textit{mu\_new})^2}{2 \times \textit{sigma2\_new}}\right) . \tag{47}$$

Part *C*:

20. With probability *pra* let *accept* $\leftarrow$ *True* , else *accept* $\leftarrow$ *False* .
21. If *accept*, then
    (a) permute the chain from ms_a to ms_a.*succ* $^{c-1}$ by *perm*;
    (b) delete ms_a;
    (c) if (not *misdat*), delete ms_b;
    (d) *Update* the chain for the changed part.

9.6 Randomize Initial Missings: Insert

This section is new in the version of May 30, 2010.

Two functions are used to randomize values of variables for which the value at the start of the period, element of $y(t_{m-1})$, is missing. These are *MH_InsMis* and *MH_DelMis*. They are called only when there is at least one missing value in $y(t_{m-1})$.

Introducing a natural Bayesian element in this otherwise frequentist procedure, these procedures utilize *prior probability distributions* for the unobserved initial variables. For the very first observation $y(t_1)$, a simple specification of these prior distributions is as follows (using $m = 1$).

- All unobserved variables are prior independent.

N. For the network variables, $N_{ij}^{(r)}(t_{m-1})$ has a prior distribution with probability
$P\{N_{ij}^{(r)}(t_{m-1}) = 1\} =$ *net_prior*(r, i, j).

B. For the behavior variables, $B_i^{(r)}(t_{m-1})$ has a prior distribution with probabilities
$P\{B_i^{(r)}(t_{m-1}) = v\} =$ ***beh_prior***(r, i, v) for v in the permitted range of his variable.

The simplest and presumably very adequate way is to let this depend only on the dependent variable, $N^{(r)}$ or $B^{(r)}$:

net_prior$(r, i, j) =$ observed density of network r, observation $m - 1$,
beh_prior$(r, i, .) =$ observed distribution of behavior r, observation $m - 1$,

For $m \geq 3$ the best solution would be to do the simulations of the $m - 1$ chains in sequence, and utilize the simulated last version of the preceding period as the first version of the following period. This does not go well, I presume, with the current architecture of RSiena which further needs no communication between simulation of different periods. A simple way out is to use the priors above here also (i.e., for general m). I think this will be very reasonable, unless there are a lot of missing data or the interest of the analysis is precisely in the missings.

The function ***MH_InsMis*** (output $w_0, i_0, j_0, r_0,$ ***pra***, ***accept***) is defined as follows. First a rough description is given.

The initial state of the chain is denoted y_{init}; this must be equal to $y(t_{m-1})$ except for the coordinates for which there is an initial missing value, and these coordinates are being changed in this procedure.

A random selection is made among the coordinates (options) for which the initial value is missing. This yields the values (w_0, i_0, j_0, r_0).
For this variable, the proposal is made to insert a ministep $(w_0, i_0, j_0, r_0, d_0)$ before some existing ministep ms_a; this ms_a is randomly chosen, under the constraint that there are no earlier ministeps of option (w_0, i_0, j_0, r_0); the proposal also includes transforming the initial value y_{init} by the opposite change $(w_0, i_0, j_0, r_0, -d_0)$ (which amounts to changing the current value of $N_{i_0 j_0}^{(r_0)}(t_{m-1})$ if $w_0 = N$ or $B_{i_0}^{(r_0)}(t_{m-1})$ if $w_0 = B$) so that from ms_a onward, the chain is the same as it used to be.

The steps taken in the function are as follows.

Part *A*:

1. If *ChainNumInitMis* $= 0$, then exit.

2. $(w_0, i_0, j_0, r_0) \leftarrow$ *RandomInitMis* ;
   if $w_0 = B$, then

   (a) $pr_1 \leftarrow 0.5$,

   (b) $d_0 \leftarrow$ random choice in $\{-1, +1\}$;

   else

   (a) $pr_1 \leftarrow 1$,

   (b) $d_0 \leftarrow 0$.

3. If $w_0 = B$, and *subtracting (rather than adding), as this is what is actually done.* d_0 to the current value of $B_{i_0}^{(r)}(t_{m-1})$ (which is one of the coordinates in y_{init}) would lead to a value of this variable outside of its permitted range, then

   (a) $pr_1 \leftarrow 1$,

   (b) $d_0 \leftarrow -d_0$,

   (c) *reversed* $\leftarrow$ true ,

   else
   reversed $\leftarrow$ false .

4. Let ms_b be
   the first ministep in the chain of *Option* (w_0, i_0, j_0, r_0);
   or the *last* ministep if there is no ministep of this *Option*.
   (Above there has been mention somewhere of the pointer *succOption* ; I do not know if something like that has been implemented, but clearly some of the same machinery might be used here.)

5. *ChoiceLength* $\leftarrow$ *length*(*first* , ms_b) $- 1$

6. Let ms_a a random ministep in the interval $[$*first* .*succ* , $ms_b]$.
   (Thus, ms_b also is permitted.)
   Note that the number of choices here is *ChoiceLength* ≥ 1.

7. If variable (w_0, r_0) is involved in any of the conditions *uponly* , *downonly* , *higher* , *disjoint* , or *atleastone* , then :

   If changing y_{init} by $(w_0, i_0, j_0, r_0, -d_0)$ while inserting a ministep of option $(w_0, i_0, j_0, r_0, d_0)$ immediately before ms_a would lead to a violation of at least one of these conditions, then:

   (a) if *reversed* or $(w_0 = N)$ then exit, else

   $$d_0 \leftarrow -d_0$$
   $$pr_1 \leftarrow 1 \ ;$$

   (b) If for this opposite value of d_0, changing y_{init} by $(w_0, i_0, j_0, r_0, -d_0)$ while inserting a ministep of option $(w_0, i_0, j_0, r_0, d_0)$ immediately before ms_a would also violate one or more of these conditions, then exit.

The proposal probability here is

$$\frac{pr_1}{\textit{NumInitMis} \times \textit{ChoiceLength}} \ .$$

This is used in *pra* below.

Part *B*:

8.

$$\textit{sumlprob} \leftarrow \sum_{\textit{ms}=\textit{first.succ}}^{\textit{ms}_a.\textit{pred}} (\textit{ms}.(\textit{lChoiceProb} + \textit{lOptionSetProb}) ;$$

$$\textit{sumlprob\_new} \leftarrow 0 ;$$

$$\textit{mu\_new} \leftarrow \textit{mu} - \sum_{\textit{ms}=\textit{first.succ}}^{\textit{ms}_a.\textit{pred}} (\textit{ms.rRate}) ;$$

$$\textit{sigma2\_new} \leftarrow \textit{sigma2} - \sum_{\textit{ms}=\textit{first.succ}}^{\textit{ms}_a.\textit{pred}} (\textit{ms.rRate})^2 .$$

9. $y \leftarrow y_{\text{init}}$;

   Let pr_2 be the prior probability (see *net_prior* and *beh_prior* discussed above) of value $y(w_0, i_0, j_0, r_0)$ and pr_3 the prior probability of the value when this variable is changed according to ministep $(w_0, i_0, j_0, r_0, -d_0)$;

   ChangeStep$(y, (w_0, i_0, j_0, r_0, -d_0))$ (now we have the new initial value);

   StepProb1$(y, w_0, i_0, j_0, r_0; rr_0, lospr_0)$;

   If (not *SimpleRates*), then

   (a) *mu_new* $\leftarrow$ *mu_new* $+ rr_0$,

   (b) *sigma2_new* $\leftarrow$ *sigma2_new* $+ (rr_0)^2$;

10. For *ms* running from *first .succ* to $\textit{ms}_a.\textit{pred}$ do:

    (a) *StepProb2*$(y,$ *ms.Coordinates*$; rr, lospr, lcpr)$, where *ms.Coordinates* are the coordinates of ministep *ms*;

    (b) *sumlprob_new* $\leftarrow$ *sumlprob_new* $+ lcpr + lospr$

    (c) If (not *SimpleRates*), then

        i. *mu_new* $\leftarrow$ *mu_new* $+ rr$;

ii. *sigma2_new* $\leftarrow$ *sigma2_new* $+ (rr)^2$.

(d) *ChangeStep*$(y,$ *ms*$)$

11. (a) *StepProb2*$(y, w_0, i_0, j_0, r_0, d_0; rr_0, lospr_0, lcpr_0)$

(b) *sumlprob_new* $\leftarrow$ *sumlprob_new* $+ lcpr_0 + lospr_0$

12. If *SimpleRates* (which implies $rr = rr_0$), then

(a)

$$\textit{KappaFactor} \leftarrow \frac{1}{rr_0 \times \textit{TotNumber}}$$

else (i.e., if not *SimpleRates*)

$$\textit{KappaFactor} \leftarrow \sqrt{\frac{\textit{sigma2}}{\textit{sigma2\_new}}} \times \exp\left(\frac{(1-\textit{mu})^2}{2 \times \textit{sigma2}} - \frac{(1-\textit{mu\_new})^2}{2 \times \textit{sigma2\_new}}\right) . \tag{48}$$

13.

$$\textit{pra} \leftarrow \textit{KappaFactor} \times \exp(\textit{sumlprob\_new} - \textit{sumlprob}) \tag{49}$$

$$\times \frac{\textit{prdms} \times \textit{ChoiceLength} \times pr_3}{\textit{prims} \times pr_1 \times pr_2} ; \tag{50}$$

if (*pra* > 1), then *pra* $\leftarrow 1$.

Part *C*:

14. With probability *pra* let *accept* $\leftarrow$ *True* , else *accept* $\leftarrow$ *False* .

15. If *accept*, then

(a) *ChangeStep*$(y_{\text{init}}, (w_0, i_0, j_0, r_0, -d_0))$

(b) insert $(w_0, i_0, j_0, r_0, d_0)$ before ms_a;

(c) *Update* the chain for the changed part.

9.7 Randomize Initial Missings: Delete

This section is new in the version of May 30, 2010.

Two functions are used to randomize values of variables for which the value at the start of the period, element of $y(t_{m-1})$, is missing. These are *MH_InsMis* and *MH_DelMis*. They are called only when there is at least one missing value in $y(t_{m-1})$. This section specifies *MH_DelMis*. The same prior distributions *net_prior*(r, i, j) and *beh_prior*(r, i, v) are used as in the preceding section.

The function *MH_DelMis* (output $w_0, i_0, j_0, r_0,$ *pra*, *accept*) is defined as follows. First a rough description is given.

The initial state of the chain is denoted y_{init}; this must be equal to $y(t_{m-1})$ except for the coordinates for which there is an initial missing value, and one of these coordinates may be changed in this procedure.

A random selection is made among the coordinates (options) for which the initial value is missing. This yields the output values (w_0, i_0, j_0, r_0).
If no ministep of this option exist, then nothing happens. If ministeps of this option do exist, the proposal is made to delete the first ministep of this option; given that the ministep has the coordinates $(w_0, i_0, j_0, r_0, d_0)$, the proposal is combined with changing transforming the initial value y_{init} by this change $(w_0, i_0, j_0, r_0, d_0)$ (which amounts to changing $N_{i_0 j_0}^{(r_0)}(t_{m-1})$ if $w_0 = N$ or $B_{i_0}^{(r_0)}(t_{m-1})$ if $w_0 = B$) so that after the to-be-deleted ministep, the chain will be the same as it used to be.

The quantities *ChoiceLength* and pr_1 are computed because they play a role in the proposal probability for the reverse proposal, and hence in the acceptance probability of this proposal.

The steps taken in the function are as follows.

Part *A*:

1. If *ChainNumInitMis* $= 0$, then exit.
2. $(w_0, i_0, j_0, r_0) \leftarrow$ *RandomInitMis* .
3. Let *ms*$_a$ be the first ministep in the chain of *Option* (w_0, i_0, j_0, r_0), and $d_0 \leftarrow$ *ms*$_a$.d .
   (Above there has been mention somewhere of the pointer *succOption* ; I do not know if something like that has been implemented, but clearly some of the same machinery might be used here.)
4. Let *ms*$_b$ be the second ministep in the chain of *Option* (w_0, i_0, j_0, r_0); and the *last* ministep if there is no second of this option;
   ChoiceLength $\leftarrow$ *length*(*first* , *ms*$_b$) $- 2$.
   (*ms*$_b$ is not used further.)
5. If $w_0 = B$, and adding $2d_0$ to $B_{i_0}^{(r)}(t_{m-1})$ would not lead to a value of this variable outside of its permitted range, then

   $$pr_1 \leftarrow 0.5,$$

   else

   $$pr_1 \leftarrow 1 \ .$$

6. If variable (w_0, r_0) is involved in any of the conditions *uponly* , *downonly* , *higher* , *disjoint* , or *atleastone* :
   (a) If changing y_{init} by $(w_0, i_0, j_0, r_0, d_0)$ while deleting ministep *ms*$_a$ would lead to a violation of this condition, then exit.
   (b) If $pr_1 = 0.5$ and changing y_{init} by $(w_0, i_0, j_0, r_0, 2d_0)$ while simultaneously replacing ministep *ms*$_a$ by a ministep with coordinates $(w_0, i_0, j_0, r_0, -d_0)$ would lead to a violation of this condition, then:

   $$pr_1 \leftarrow 1 \ .$$

The proposal probability here is

$$\frac{1}{\textit{NumInitMis}} \ .$$

This is used in *pra* below.

Part *B*:

7.

$$\textit{sumlprob} \leftarrow \sum_{\textit{ms}=\textit{first.succ}}^{\textit{ms}_a} (\textit{ms.}(\textit{lChoiceProb} + \textit{lOptionSetProb}) ;$$

$$\textit{sumlprob\_new} \leftarrow 0 ;$$

$$\textit{mu\_new} \leftarrow \textit{mu} - \sum_{\textit{ms}=\textit{first.succ}}^{\textit{ms}_a} (\textit{ms.rRate}) ;$$

$$\textit{sigma2\_new} \leftarrow \textit{sigma2} - \sum_{\textit{ms}=\textit{first.succ}}^{\textit{ms}_a} (\textit{ms.rRate})^2 .$$

8. $y \leftarrow y_{\text{init}}$;

   Let pr_2 be the prior probability (see *net_prior* and *beh_prior* discussed above) of value $y(w_0, i_0, j_0, r_0)$ and pr_3 the prior probability of the value when this variable is changed according to ministep $(w_0, i_0, j_0, r_0, d_0)$;

   ChangeStep$(y, (w_0, i_0, j_0, r_0, d_0))$;

   StepProb1$(y, w_0, i_0, j_0, r_0; rr_0, lospr_0)$;
   Krists had added lospr0 and lcpr0 to sumlprobnew. I removed this.

   If (not *SimpleRates*), then

   (a) *mu_new* $\leftarrow$ *mu_new* $+ rr_0$,

   (b) *sigma2_new* $\leftarrow$ *sigma2_new* $+ (rr_0)^2$.

9. For *ms* running from *first .succ* to $\textit{ms}_a$*.pred* do:

   (a) *StepProb2*$(y,$ *ms.Coordinates*$; rr, lospr, lcpr)$,
   where *ms.Coordinates* are the coordinates of ministep *ms* ;

   (b) *sumlprob_new* $\leftarrow$ *sumlprob_new* $+ lcpr + lospr$

(c) If (not *SimpleRates*) and ($ms < ms_a.pred$), then

i. $mu\_new \leftarrow mu\_new + rr$,

ii. $sigma2\_new \leftarrow sigma2\_new + (rr)^2$;

comment : the reciprocal rate calculated for $ms = ms_a.pred$ is, in the proposed new chain, *rRate* for $ms_a.succ$, which is unchanged and has not been subtracted in step **??**.

(d) *ChangeStep*(y, ms) .

10. If *SimpleRates* (which implies $rr = rr_0$), then

$$KappaFactor \leftarrow rr_0 \times (TotNumber - 1)$$

else (i.e., if not *SimpleRates*)

$$KappaFactor \leftarrow \sqrt{\frac{sigma2}{sigma2\_new}} \times \exp\left(\frac{(1-mu)^2}{2 \times sigma2} - \frac{(1-mu\_new)^2}{2 \times sigma2\_new}\right) . \tag{51}$$

11.

$$pra \leftarrow KappaFactor \times \exp(sumlprob\_new - sumlprob) \tag{52}$$

$$\times \frac{prims \times pr_1 \times pr_3}{\times prdms \times ChoiceLength \times pr_2} ; \tag{53}$$

if ($pra > 1$), then $pra \leftarrow 1$.

Part *C*:

12. With probability *pra* let *accept* $\leftarrow$ *True* , else *accept* $\leftarrow$ *False* .

13. If *accept*, then

(a) *ChangeStep*$(y_{\text{init}}, (w_0, i_0, j_0, r_0, d_0))$

(b) delete ministep ms_a;

(c) *Update* the chain for the changed part.

10 Likelihood-based calculations: Simulation complete-data log-likelihoods

The simulation of the complete-data log-likelihood for a given parameter θ is done as follows (cf. Section 3.4 in **?**). Separate chains are made for all periods $m = 1, \ldots, M-1$. This chapter describes what is done for each period.

1. The chain is initialized by the procedure *Connect* and by setting

$$y_{\text{init}} \leftarrow y(t_{m-1}) \ .$$

2. After this initialization, a burn-in is necessary (only once, at the start of the likelihood-based MCMC process, i.e., only for the very first value of θ employed; search for "previous value" in **?**).

3. A large number of Metropolis Hastings steps is used to transform the current chain to a new chain which can be regarded as a random draw from the conditional distribution of chains given the observed data, for the current parameter value θ.

4. For this chain, the complete-data log-likelihood is calculated.

10.1 Burn-in

The burn-in procedure here is different from the one in Siena 3 (the difference is not important).

As a pre-burn-in it is good to insert, immediately after *Connect*, some *diagonal* steps and some mutually canceling pairs of steps as long as this increases the likelihood of the chain. This is done as follows, using functions *MH_InsertDiag* and *MH_InsPermute* explained below.

5. Repeat *MH_InsertDiag*(*pra, accept*) until 5 times '*not accept*'.

6. Repeat *MH_InsPermute*(*1, misdat, pra, accept*) until 5 times '*not accept*'.

The number 5 is arbitrary but reasonable.

After the pre-burn-in *nummax* Metropolis Hastings steps are made as described in the following section, where *nummax* is a suitable number. I suggest *nummax* = 500 for the moment; we may experiment with other values.

To keep users at ease it will be good to have the gui (if it is being used) display the message 'Burn-in iterations' during the burn-in phase.

10.2 Metropolis Hastings steps

The Metropolis Hastings step is a probabilistic choice between the following procedures.

1. With probability *pridg*, make step *MH_InsertDiag*;
2. with probability *prcdg*, make step *MH_CancelDiag*;
3. with probability *prper*, make step *MH_Permute*;
4. with probability *pripr*, make step *MH_InsPermute*;
5. with probability *prdpr*, make step *MH_DelPermute*;
6. with probability *prims*, make step *MH_InsMis*;
7. with probability *prdms*, make step *MH_DelMis*;

Probabilities *prims* and *prdms* will be 0 if *ChainNumInitMis* = 0.
Evidently, *pridg* + *prcdg* + *prper* + *pripr* + *prdpr* + *prims* + *prdms* = 1, and I suppose that it makes sense to try working with *pridg* = *prcdg*, *pripr* = *prdpr*, and *prims* = *prdms*.

The probabilities of the various Metropolis Hastings steps are parameters of the algorithm. I believe that in Siena 3, the following values were used:
pridg = *prcdg* = 0.05;
prper = 0.3;
pripr = *prdpr* = 0.3;
prims = *prdms* = 0.

If for all dependent variables *uponly* or *downonly* holds, then *pripr* = *prdpr* = 0.

If *ChainNumInitMis* > 0, the probabilities *prims* and *prdms* associated with the steps *MH_InsMis* and *MH_DelMis* could have the default value

$$\textit{prims} = \textit{prdms} = \frac{\textit{ChainNumInitMis}}{2(R_N\, n(n-1) \,+\, R_B\, n)} \,,$$

truncated to some range between, say, .001 and .05. (These values will still require further experimentation!)

The integer number $c_0 \geq 0$, used in procedures *MH_Permute*, *MH_InsPermute*, and *MH_DelPermute* to define the length of the sequence of ministeps to be permuted, is a parameter of the algorithm and is determined adaptively. It depends on the period. (In Siena 3 this is stored in the variable *numm*.) The heuristic idea is that if proposals for permuting longer sequences (which make larger steps in the outcome space but imply more work) have a lower probability to be accepted, then we aim at an acceptance rate of about 0.5; if proposals with longer sequences have a higher acceptance probability, then we can propose the permutation of long sequences. In all cases c_0 is bounded between $c_{\text{min}} = 2$ and $c_{\text{max}} = 40$. (The constants 2 and 40 are parameters meant to be touched only for algorithmic fine-tuning by very experienced users.) c_0 can be initialised at 20 for all periods. The adaptive procedure is as follows.

After each of the procedures *MH_Permute*, *MH_InsPermute*, and *MH_DelPermute*:

'up' If *accept*, then $c_0 \leftarrow c_0 + 0.5$,
but c_0 is not allowed to exceed $\min\{c_{\text{max}}, \textit{TotNumber-1}\}$.

'down' If *not accept*, then $c_0 \leftarrow c_0 - 0.5$,
but c_0 is not allowed to become less than c_{min}.

This adaptive choice is suggested by the Robbins-Monro algorithm as a way to aim at an acceptance rate of 0.5 under the conditions and constraints mentioned above. It is felt that the complications for the convergence of the Metropolis Hastings algorithm that are caused by this adaptation are of such a minor nature that they can safely be ignored.

However, it is important to check the algorithm and tune the adaptation; in particular if c_0 would hover close to c_{min} this might be a sign that something is not well.

The number of Metropolis Hastings steps until the next recorded complete-data log-likelihood (with a Robbins-Monro step, or for a Bayesian random update of the parameters) for period $m-1$ is

$$\begin{aligned} \textit{NumStep} = &\textit{MultiplicationFactor} \times \\ &\big(\textit{TotalDistance(Network)} + \textit{TotalDistance(Behavior)}\big) \end{aligned} \tag{54}$$

where

$$\textit{TotalDistance(Network)} = \sum_{r=1}^{R_N} \sum_{i,j} \mid x_{ij}^{(r)\,\text{obs}}(t_m) - x_{ij}^{(r)\,\text{obs}}(t_{m-1}) \mid ,$$

where the sum is over all tie variables that are not structurally fixed at t_{m-1} or t_m (note that it is possible that tie variables are structurally fixed but have different subsequent values) and

$$\textit{TotalDistance(Behavior)} = \sum_{r=1}^{R_B} \sum_{i} \mid z_{i}^{(r)\,\text{obs}}(t_m) - z_{i}^{(r)\,\text{obs}}(t_{m-1}) \mid ,$$

where the sum is over all actors that are not structurally inactive at t_{m-1} or t_m.

The *MultiplicationFactor* must be tuned by the user, based mainly on information about the autocorrelations of generated function values (scores) in Phase 3. A reasonable default value is 5.0, but often (mostly?) higher values will be necessary. For the Robbins-Monro methods, autocorrelations should preferably be less than 0.4, and a warning may be issued in the output file if this is not the case.

For diagnostic output, it will be good to have information about the proportion of accepted Metropolis Hastings proposals of all the different step types for all the $R_N + R_B$ different outcome variables (networks and behaviors); about the distribution of c_0; and about the autocorrelations of the generated scores in Phase 3. This information should be on the sienaFit object, but only the autocorrelations should be reported by default; the rest should be available on request.

10.3 Score functions

After the *NumStep* Metropolis-Hastings steps of the preceding subsection, the score function must be calculated for the resulting chain. This can be done separately for each of the periods, yielding J_m for $m = 2, \ldots, M$ as defined in Section **??**. This is elaborated in this section.

The score function for the parameters of the rate function here is different from what is specified in simstats0c. The reason is that for the likelihood-based calculations, the 'complete data' is the embedded chain (without the time increments), whereas in simstats0c the complete chain, including the time increments ('tau'), is used. In simstats0c the contributions to the score function are calculated immediately after each ministep. Working with only the embedded chain is more efficient, because part of the variability is integrated out; the advantage for the likelihood-based calculations is that there are no complications springing from changing dimensionality of the continuous part of the outcome space. For rate functions depending on the state y, working with the embedded chain entails the complications associated with the constant denoted κ and its consequence *KappaFactor*.

First consider the case where the rate function does not depend on the state $y = (x, z)$, although it may depend on the actor i, and it will depend (if $R_N + R_B > 1$) on the dependent variable labeled by (w, r). The rate function then is expressed by

$$\lambda^w(\theta, r, i, y) = \rho_r^w \, \exp(\alpha_r^w s_{ri}^w) \tag{55}$$

(where $\alpha_r^w s_{ri}^w$ is the inner product of these two vectors). Thus, there is a basic rate parameter ρ_r^w for each dependent variable labeled (w, r); in addition there may be actor-dependent variables s_{ri}^w that affect the rate for dependent variable (w, r), but the parameters α_r^w are distinct for distinct dependent variables.

All sums over actors i in the following are over the active actors.

Denote the number of ministeps of *OptionSet* (w, i, r) by T_{ri}^w and $T_r^w = \sum_i T_{ri}^w$. Note that $\sum_{w,r} T_r^w =$ *TotNumber* $- 1$. The score functions are

given for the basic rate parameters by

$$\frac{\partial \,\text{complete data log-likelihood}}{\partial \rho_r^w} = \frac{T_r^w}{\rho_r^w} - \sum_{\text{active } i} \exp(\alpha_r^w s_{ri}^w) \;, \tag{56}$$

which for cases with *only* a basic rate parameter $\lambda^w(\theta, r, i, y) = \rho_r^w$, and constant number of active actors n_{act}, reduces to

$$\frac{\partial \,\text{complete data log-likelihood}}{\partial \rho_r^w} = \frac{T_r^w}{\rho_r^w} - n_{\text{act}} \;; \tag{57}$$

and, denoting the index number of the covariate for the rate function by h, the score functions for the other rate parameters are

$$\begin{aligned} \frac{\partial \,\text{complete data log-likelihood}}{\partial \alpha_{rh}^w} &= \sum_{\text{active } i} s_{rih}^w \big(T_{ri}^w - \rho_r^w \, \exp(\alpha_r^w s_{ri}^w)\big) \\ &= \sum_{\text{active } i} s_{rih}^w \big(T_{ri}^w - \lambda^w(\theta, r, i, y)\big) \;. \end{aligned} \tag{58}$$

More background is given in Appendix **??**.

In the general case, where rates can depend on the state y, we use a heuristic approximation which is the direct generalization of the above. Suppose the rate function is given by

$$\lambda^w(\theta, r, i, y) = \rho_r^w \, \exp(\alpha_r^w s_{ri}^w(y)) \tag{59}$$

which is just like (**??**) except that now the variables $s_{ri}^w(y)$ are functions of y and not only of (w, r, i). Then we must sum over the ministeps, denoted here by $m = 1, \ldots,$ *TotNumber*-1, instead of over the actors. By $\Delta_{ri}^w(m)$ we denote the indicator function of the event that in ministep m, actor i makes a ministep in variable (w, r); in terms of the earlier sections, the event that the *OptionSet* of ministep m is (w, i, r); we define $\Delta_{ri}^w(m) = 1$ if this event is true, and 0 if it is false. Thus, $\sum_m \Delta_{ri}^w(m) = T_{ri}^w$ and $\sum_{w,r,i} \Delta_{ri}^w(m) = 1$. The state before ministep m is denoted y_m.

The score functions now are given for the basic rate parameters by

$$\begin{aligned} &\frac{\partial \,\text{complete data log-likelihood}}{\partial \rho_r^w} = \\ &\qquad \frac{T_r^w}{\rho_r^w} - \frac{1}{\textit{TotNumber} - 1} \sum_{m=1}^{\textit{TotNumber}-1} \sum_{\text{active } i} \exp(\alpha_r^w s_{ri}^w(y_m)) \;, \end{aligned} \tag{60}$$

and for the other rate parameters they are

$$\frac{\partial \,\text{complete data log-likelihood}}{\partial \alpha^w_{rh}} =$$
$$\sum_{m=1}^{\mathit{TotNumber}-1} \sum_{\text{active } i} s^w_{rih}(y_m) \left(\Delta^w_{ri}(m) - \frac{\rho^w_r \, \exp(\alpha^w_r s^w_{ri}(y_m))}{\mathit{TotNumber} - 1} \right)$$
$$= \sum_{m=1}^{\mathit{TotNumber}-1} \sum_{\text{active } i} s^w_{rih}(y_m) \left(\Delta^w_{ri}(m) - \frac{\lambda^w(\theta, r, i, y_m)}{\mathit{TotNumber} - 1} \right) . \quad (61)$$

This reduces to the equations above in case that the functions $s^w_{ri}(y)$ are independent of y, so there is nothing against using these last two equations in all cases.

In Phase 3 of the Robbins-Monro algorithm, we need something extra, viz., the complete-data observed information matrix, which is the $p \times p$ matrix of minus the second partial derivatives of the log-likelihood function. Denote this by H_m.

11 Likelihood-based calculations: Robbins-Monro algorithm

The Robbins-Monro algorithm as above is applied, but now to the statistic defined as the score function $S = \sum_{m=2}^{M} J_m$. Note that equation to be solved here is

$$\mathsf{E}_\theta S = 0 \, ,$$

so the role of the observed value s in the MoM is now taken by the number 0.

The estimated covariance matrix of the ML estimator is

$$\left(\hat{\Sigma}_{\text{complete}} - \hat{\Sigma}_{\text{missing}} \right)^{-1}$$

where $\hat{\Sigma}_{\text{complete}}$ is the estimated complete data information matrix while $\hat{\Sigma}_{\text{missing}}$ is the estimated information matrix for the missing data (cf. (25) in

?). $\hat{\Sigma}_{\text{complete}}$ is obtained as the average of the generated $\sum_{m=2}^{M} H_m$ over all Phase-3 iterations, and $\hat{\Sigma}_{\text{missing}}$ is obtained as the covariance matrix of the generated score functions in Phase 3, i.e., Σ in (a) of Phase 3 of Section **??**.

My experience with this estimated covariance matrix is mixed, as the difference between two positive definite matrices sometimes turns out to be not positive semi-definite itself. In such cases having a long Phase 3 will improve things. It will be worthwhile to think about a better way of estimating Cov$(\hat{\theta})$.

In the Robbins-Monro algorithm we might later experiment with an adaptation, in which after the Robbins-Monro update step we solve for ρ using the score function for ρ – consider (**??**), (**??**), and (**??**). (I am not sure that this works well, as this might conflict with the stochasticity of the Robbins-Monro algorithm.)

12 Likelihood-based calculations: Store chains

For communication with users and with other programs, it is necessary to have a way of reading chains from files and writing them to files. Chains also have to be communicated to R.

For writing, I propose to have two ways of writing them. Always, one line for each ministep, in their natural order. Numbers within lines separated by separator that can be space, tab, (comma & space).

1. brief: the line gives w, r, i, j, d in this order (note r comes second).
2. long: the line gives w, r, i, j, d, *rRate* , *lOptionSetProb* , *lChoiceProb* in this order.

13 Likelihood-based calculations: Structurally fixed values

If $N^r(i,j)$ is structurally fixed and $N^r(i,j)(t_{m-1}) = N^r(i,j)(t_m)$, then the chain for period m must not contain any ministeps of Option (Network, i,j,r).
If $B^r(i)$ is structurally fixed and $B^r(i)(t_{m-1}) = B^r(i)(t_m)$, then the chain for period m must not contain any ministeps of Option (Behavior, $i,*,r$).

For the variables that are structurally fixed but have values at t_m different from their values at t_{m-1}, the principle is that these changes are enforced either directly before the *last* ministep, or as part of the *last* ministep (whichever is the simpler or more elegant; I think the latter). These changes do not contribute anything to probabilities or rates; this can be implemented formally by omitting them from sums or by defining *lChoiceProb* = 0 and *rRate* = 0. If there are several such variables, the order in which these changes are enforced does not matter (and is inconsequential).

14 Meta-analysis

Results from several independent network data sets can be combined in a meta-analysis according to the method of **?**, who applied the method of **?** (also described by **?**) to this type of analysis. This section also elaborates some further methods.

Suppose we have N independent network data sets, in which the same sets of covariates are used, and that were analyzed by the same model specification. The meta-analysis is done for each parameter separately. Thus, for this explanation we focus on any coordinate of the parameter vector, and denote this coordinate by θ. From the j'th data set we obtain estimate $\hat{\theta}_j$ with standard error s_j. The model postulates that

$$\hat{\theta}_j = \theta_j + E_j, \tag{62}$$

where θ_j for $j = 1, \ldots, N$ is an i.i.d. sample from a distribution with mean μ_θ and variance σ_θ^2; and E_j is independent of θ_j and has mean 0 and

standard deviation s_j. Thus, we ignore in this analysis the error in the estimation of the estimation error var(E_j). The purpose of the meta-analysis is estimating and testing μ_θ and σ^2_θ.

What we observe from data set j is not θ_j but the estimate $\hat{\theta}_j$. This is a random variable with mean μ_θ and variance $\sigma^2_\theta + s^2_j$.

14.1 Preliminary and two-step estimator

Here we give the unbiased estimator for σ^2_θ and a two-stage estimator for the mean μ_θ that were presented in **?**, following **?**.

A preliminary unbiased estimator for μ_θ is given by

$$\hat{\mu}^{\text{OLS}}_\theta = \frac{1}{N} \sum_j \hat{\theta}_j \ . \tag{63}$$

This estimator does not take into account the fact that the standard errors s^2_j may be different. This implies that, although it is unbiased, the estimator may be inefficient. Its standard error is

$$\text{s.e.}\,(\hat{\mu}^{\text{OLS}}_\theta) = \sqrt{\frac{1}{N}\,(\sigma^2_\theta + \bar{s}^2)} \tag{64}$$

where

$$\bar{s}^2 = \frac{1}{N} \sum_j s^2_j \tag{65}$$

is the *average error variance*. An unbiased estimator for the variance σ^2_θ is

$$\hat{\sigma}^{2,\text{OLS}}_\theta = \frac{1}{N-1} \sum_j \left(\hat{\theta}_j - \hat{\mu}^{\text{OLS}}_\theta\right)^2 \ - \ \bar{s}^2 \ . \tag{66}$$

In words, this is the the *observed variance* of the estimates minus the *average error variance*. If this difference yields a negative value, it will be good to truncate it to 0.

Given that the latter estimator has been calculated, it can be used for an improved estimation of μ_θ, viz., by the weighted least squares (WLS) estimator

$$\hat{\mu}^{\text{WLS}}_\theta = \frac{\sum_j \left(\hat{\theta}_j / (\hat{\sigma}^{2,\text{OLS}}_\theta + s^2_j)\right)}{\sum_j \left(1/(\hat{\sigma}^{2,\text{OLS}}_\theta + s^2_j)\right)} \ . \tag{67}$$

This is the 'semi-weighted mean' of **?**, treated also in **?**, Section 9.F. Its standard error can be calculated as

$$\text{s.e.}\big(\hat{\mu}_\theta^{\text{WLS}}\big) = \frac{1}{\sqrt{\sum_j 1/(\hat{\sigma}_\theta^{2,\text{OLS}} + s_j^2)}} \,. \tag{68}$$

14.2 Maximum likelihood estimator

The maximum likelihood estimator (MLE) under the assumption that the $\hat{\theta}_j$ are independent and normally distributed (note that this is an assumption about their marginal distributions, not their distributions conditional on the true values θ_j) is defined by two equations. The first is the equation for $\hat{\mu}$ given σ^2:

$$\hat{\mu} \;=\; \frac{\sum_j \big(\hat{\theta}_j/(\sigma^2 + s_j^2)\big)}{\sum_j \big(1/(\sigma^2 + s_j^2)\big)} \,. \tag{69}$$

The second is the requirement that the profile log-likelihood for σ^2 is maximized. This profile log-likelihood is given by

$$p(\sigma^2) \;=\; -\frac{1}{2}\sum_j \log\big(\sigma^2 + s_j^2\big) \;-\; \frac{1}{2}\sum_j \frac{\big(\hat{\theta}_j - \hat{\mu}\big)^2}{\sigma^2 + s_j^2} \,. \tag{70}$$

As a first step to maximize this, the derivative and second derivative can be computed; here it should be kept in mind that $\hat{\mu} = \hat{\mu}(\hat{\sigma}^2)$ is given as a function of $\hat{\sigma}^2$ in (**??**) – however, that part cancels out in the derivative so forgetting this might still yield the correct answer. Further it is convenient to work with the function $c_j(\sigma^2) = 1/(\sigma^2 + s_j^2)$ and note that $dc_j/d\sigma^2 = -c_j^2$. The result is

$$\frac{d\,p(\sigma^2)}{d\,\sigma^2} \;=\; -\frac{1}{2}\sum_j \frac{1}{\sigma^2 + s_j^2} \;+\; \frac{1}{2}\sum_j \frac{(\hat{\theta}_j - \hat{\mu})^2}{(\sigma^2 + s_j^2)^2} \tag{71}$$

$$\frac{d^2\,p(\sigma^2)}{d\,(\sigma^2)^2} \;=\; -\frac{1}{2}\sum_j \frac{1}{\big(\sigma^2 + s_j^2\big)^2} \;-\; \sum_j \frac{(\hat{\theta}_j - \hat{\mu})^2}{(\sigma^2 + s_j^2)^3} \,. \tag{72}$$

Thus, one way to compute the MLE is to iterate the two steps:

1. Compute $\hat{\mu}$ by (**??**)

2. Solve $d\,p(\sigma^2)/d\,\sigma^2 = 0$ using definition (**??**).

Another way is to iterate the two steps:

1. Compute $\hat{\mu}$ by (**??**)
2. One Newton-Raphson step (or two):

$$\sigma^2_{\text{new}} \;=\; \sigma^2 \;+\; \frac{\sum_j c_j(c_j d_j^2 - 1)}{\sum_j c_j^2(2c_j d_j^2 + 1)} \tag{73}$$

where

$$c_j \;=\; \frac{1}{\sigma^2 + s_j^2}\;,\; d_j \;=\; \hat{\theta}_j - \hat{\mu}\;.$$

The results of this iteration scheme will be denoted by $\hat{\mu}_\theta^{\text{IWLS}}$ and $\hat{\sigma}_\theta^{2,\text{IWLS}}$ (IWLS for *iteratively reweighted least squares*), but the name ML could equally well be used.

The standard error of $\hat{\mu}_\theta^{\text{IWLS}}$ can be calculated as

$$\text{s.e.}\big(\hat{\mu}_\theta^{\text{IWLS}}\big) = \frac{1}{\sqrt{\sum_j 1/(\hat{\sigma}_\theta^{2,\text{IWLS}} + s_j^2)}}\;. \tag{74}$$

14.3 Testing

(This section again follows **?**.)

For testing μ_θ and σ_θ^2, it is assumed that the parameter estimates $\hat{\theta}_j$ conditional on θ_j are approximately normally distributed with mean θ_j and variance s_j^2. The first null hypothesis to be tested is that the effects are 0 in all groups. This can be tested by the test statistic

$$T^2 = \sum_j \left(\frac{\hat{\theta}_j}{s_j}\right)^2 \tag{75}$$

which has an approximate χ^2 distribution with N degrees of freedom under the null hypothesis. The test that the mean effect μ_θ is zero can be tested on the basis of the t-ratio

$$t_{\mu_\theta} = \frac{\hat{\mu}_\theta}{\text{s.e.}\big(\hat{\mu}_\theta\big)} \tag{76}$$

which has approximately a standard normal distribution under the null hypothesis. Finally, the test that the variance of the effects σ_θ^2 is zero can be tested using the test statistic

$$Q = T^2 - \tilde{t}^2 \tag{77}$$

where

$$\tilde{t} = \frac{\sum_j \hat{\theta}_j / s_j^2}{\sqrt{\sum_j 1/s_j^2}} \tag{78}$$

which has under the null hypothesis approximately a chi-squared distribution with $N - 1$ degrees of freedom.

14.4 Fisher combination of p-values

Fisher's (1932) procedure for combination of independent p-values is applied both to left-sided and right-sided p-values. In this way, we are able to report tests for both the following testing problems:

$$H_0^{(R)} : \ \theta_j \leq 0 \quad \text{for all } j;$$
$$H_1^{(R)} : \ \theta_j > 0 \quad \text{for at least one } j.$$

Significance is interpreted here, that there is evidence that in *some* (at least one) data set, parameter θ_j is positive.

$$H_0^{(L)} : \ \theta_j \geq 0 \quad \text{for all } j;$$
$$H_1^{(L)} : \ \theta_j < 0 \quad \text{for at least one } j.$$

Significance is interpreted here, that there is evidence that in *some* (at least one) data set, parameter θ_j is negative.

Note that it is very well possible that both one-sided combination tests are significant: then there is evidence for some positive and some negative effects.

The procedure operates as follows. Calculate p_j^+ and p_j^-, being the right and left one-sided p-values:

$$p_j^+ = 1 - \Phi\left(\frac{\hat{\theta}_j}{s_j}\right)$$
$$p_j^- = \Phi\left(\frac{\hat{\theta}_j}{s_j}\right) \ ,$$

where Φ is the c.d.f. of the standard normal distribution. The Fisher combination statistic is defined as

$$C_j^+ = -2 \sum_{j=1}^{N} \ln\left(p_j^+\right)$$
$$C_j^- = -2 \sum_{j=1}^{N} \ln\left(p_j^-\right) \ .$$

Both of these must be tested in a χ^2 distribution with $2\,N$ degrees of freedom.

14.5 Combinations of score-type tests

It is possible that for a parameter, score-type tests are given instead of estimates. Then these score-type tests can be combined also in a Fisher procedure. This is done just as above; but now for p-values obtained from the standard normal variates obtained as a result from the score-type test. Of course this makes sense only if the tested null values are all the same (usually 0).

14.6 Further regression analyses

The data frame of values $(\hat{\theta}_j, s_j)$, $j = 1, \ldots, N$ is made available for further analysis, possibly extended by other variables x, for analysis according to

the model

$$\hat{\theta}_j \sim \mathcal{N}\big(x_j'\beta,\, \sigma^2 + s_j^2\big), \qquad \text{independent for } j = 1, \ldots, N. \tag{79}$$

Note that the IWLS estimates of Section **??** are the estimates under such a model if $x_j'\beta$ is comprised of just a constant term.

IWLS/ML regression analysis here can be carried out by iteration of the two steps mentioned above, but now the step (**??**) is replaced by a weighted least squares analysis with weights being normalised versions of

$$w_j \;=\; \frac{1}{\sigma^2 + s_j^2}\;.$$

14.7 Differences in model specification

In practice, it can happen that a set of data sets is being offered for a meta-analysis in which the model specifications are not identical. An example is the case where one of the independent variables has variance 0 in some data sets (e.g.: an analysis of networks in schools, with pupils' sex as an independent variable; there may be some all-girls or all-boys schools).

This then must be noted in the output; and the data sets combined as if this parameter here has an estimate 0 but with an infinite standard error – in other words, this parameter should be ignored for this data set; and this data set should not add to the degrees of freedom for this particular parameter.

15 Models for Dynamics of Non-directed Networks

For notational simplicity only, this section assumes that there is only one network. This is not a restriction, it merely means that we can denote x_{ij} instead of $x_{ij}^{(r)}$.

In this section it is assumed that the network is non-directed, i.e., ties have no directionality: $X_{ij} = X_{ji}$ holds by necessity, and the tie variables X_{ij} and X_{ji} are treated as being one and the same variable. This is the case in many types of tie, such as mutual collaboration or agreement. Ties now are indicated by $i \leftrightarrow j$.

15.1 Two-sided Choices

For modeling non-directed networks, it is necessary to make assumptions about the negotiation or coordination between the two actors involved in the creation or termination of a tie. We present several models, all based on a two-step process of opportunity and choice, and making different assumptions concerning the combination of choices between the two actors involved in a tie.

For the opportunity, or timing, process, two options are presented.

1. *One-sided initiative*: One actor i is selected and gets the opportunity to make a change, based on rate function $\lambda_i(x; \alpha, \rho_m)$.

2. *Two-sided opportunity*: An ordered pair of actors (i, j) (with $i \neq j$) is selected and gets the opportunity to make a new decision about the existence of a tie between them.
   This is based on pairwise rate functions denoted $\lambda_{ij}(x; \alpha, \rho_m)$. The waiting time until the next opportunity for change by any pair of actors has the exponential distribution with parameter

$$\lambda_{\text{tot}} = \sum_{i \neq j} \lambda_{ij}(x; \alpha, \rho_m) \ .$$

The probability that the next opportunity for change is for pair (i, j) is given by

$$\text{P}\{\text{Next opportunity for change is for pair } (i,j)\} = \frac{\lambda_{ij}(x;\alpha,\rho_m)}{\lambda_{\text{tot}}(x;\alpha,\rho_m)} \ . \tag{80}$$

For the moment, we have only implemented the case where λ_{ij} is a product

$$\lambda_{ij}(x;\alpha,\rho_m) = \lambda_i(x;\alpha,\rho_m)\lambda_j(x;\alpha,\rho_m) \ ,$$

so that

$$\lambda_{\text{tot}} = \Big(\sum_i \lambda_i(x;\alpha,\rho_m)\Big)^2 - \sum_i \big(\lambda_i(x;\alpha,\rho_m)\big)^2 \ . \tag{81}$$

The choice process is modeled as one of three options D(ictatorial), M(utual) and C(ompensatory). We now define, for graphs x and $i \neq j$, by $x^{(+ij)}$ the graph which is identical to x in all tie variables except possibly for the tie between i and j, and to which the tie $i \leftrightarrow j$ is added if it was not already there: $x_{ij}^{(+ij)} = 1$. For the non-directed case, $x^{(\pm ij)}$ is defined by analogy to the definition above: it is the graph identical to x except that the indicator for the non-directed tie $i \leftrightarrow j$ has been toggled: $x_{ij}^{(\pm ij)} = x_{ji}^{(\pm ij)} = 1 - x_{ij} = 1 - x_{ji}$. Thus if $x_{ij} = 0$ then $x^{(+ij)} = x^{(\pm ij)}$; if $x_{ij} = 1$ then $x^{(+ij)} = x$.

In all cases assumption (7.) as defined for the directed case is retained, and assumption (8.) is replaced as indicated below.

D. *Dictatorial*: One actor can impose a decision about a tie on the other. Like in the directed case, actor i selects the (myopically) best toggle of a single tie variable X_{ij} given the objective function $f_i(x;\beta)$ plus a random disturbance, and actor j just has to accept. Combined with the two opportunity options, this yields the following cases.

8.D.1. (alias A-1 alias AFORCE)
The probability that the tie variable changed is X_{ij}, so that the network x changes into $x^{(\pm ij)}$, is given by

$$p_{ij}(x,\beta) = \frac{\exp\big(f_i(x^{(\pm ij)};\beta)\big)}{\sum_{h=1}^n \exp\big(f_i(x^{(\pm ih)};\beta)\big)} \ . \tag{82}$$

8.D.2. (alias B-1 alias BFORCE)
The probability that network x changes into $x^{(\pm ij)}$, is given by

$$p_{ij}(x,\beta) = \frac{\exp\left(f_i(x^{(\pm ij)};\beta)\right)}{\exp\left(f_i(x;\beta)\right) + \exp\left(f_i(x^{(\pm ij)};\beta)\right)} \,. \tag{83}$$

M. *Mutual*:
Both actors must agree for a tie between them to exist, in line with Jackson and Wolinsky (1996).

8.M.1. (alias A-2 alias AAGREE)
In the case of one-sided initiative, actor i selects the best possible choice, with probabilities (**??**). If currently $x_{ij} = 0$ so that this means creation of a new tie $i \leftrightarrow j$, this is proposed to actor j, who then accepts according to a binary choice based on objective function $f_j(x;\beta)$, with acceptance probability

$$\mathsf{P}\{j \text{ accepts tie proposal}\} = \frac{\exp\left(f_j(x^{(+ij)};\beta) + \beta^{\mathsf{b}}\right)}{\exp\left(f_j(x;\beta)\right) + \exp\left(f_j(x^{(+ij)};\beta) + \beta^{\mathsf{b}}\right)} \,,$$

where β^{b} is an offset; this is a fixed parameter, given as `UniversalOffset` in siena07algorithmCreate, and is not estimated.
If the choice by i means termination of an existing tie, the proposal is always put into effect. Jointly these rules lead to the following probability that the current network x changes into $x^{(\pm ij)}$:

$$\begin{aligned} p_{ij}(x,\beta) = {} & \frac{\exp\left(f_i(x^{(\pm ij)};\beta)\right)}{\sum_{h=1}^{n} \exp\left(f_i(x^{(\pm ih)};\beta)\right)} \\ & \times \left(\frac{\exp\left(f_j(x^{(+ij)};\beta) + \beta^{\mathsf{b}}\right)}{\exp\left(f_j(x;\beta)\right) + \exp\left(f_j(x^{(+ij)};\beta) + \beta^{\mathsf{b}}\right)} \right)^{1-x_{ij}} \,. \end{aligned} \tag{84}$$

(Note that the second factor comes into play only if $x_{ij} = 0$, which implies $x^{(+ij)} = x^{(\pm ij)}$.)

8.M.2. (alias B-2 alias BAGREE)
In the case of two-sided opportunity, actors i and j both reconsider the value of the tie variable X_{ij}. Actor i proposes a change (toggle) with probability (**??**) and actor j similarly. If currently

there is no tie, $x_{ij} = 0$, then the tie is created if this is proposed by both actors, which has probability

$$p_{ij}(x, \beta) = \tag{85a}$$
$$\frac{\exp\big(f_i(x^{(+ij)}; \beta)\big)}{\Big(\exp\big(f_i(x; \beta)\big) + \exp\big(f_i(x^{(+ij)}; \beta)\big)\Big)} \times$$
$$\frac{\exp\big(f_j(x^{(+ij)}; \beta)\big)}{\Big(\exp\big(f_j(x; \beta)\big) + \exp\big(f_j(x^{(+ij)}; \beta)\big)\Big)} .$$

If currently there is a tie, $x_{ij} = 1$, then the tie is terminated if one or both actors wish to do this, which has probability

$$p_{ij}(x, \beta) = 1 - \tag{85b}$$
$$\left\{ \frac{\exp\big(f_i(x; \beta)\big)}{\Big(\exp\big(f_i(x; \beta)\big) + \exp\big(f_i(x^{(\pm ij)}; \beta)\big)\Big)} \times \right. \tag{85c}$$
$$\left. \frac{\exp\big(f_j(x; \beta)\big)}{\Big(\exp\big(f_j(x; \beta)\big) + \exp\big(f_j(x^{(\pm ij)}; \beta)\big)\Big)} \right\} .$$

C. *Compensatory*: (alias B-3 alias BJOINT)
The two actors decide on the basis of their combined interests.
The combination with one-sided initiative is rather artificial here, and we only elaborate this option for the two-sided initiative.

8.C.2. The binary decision about the existence of the tie $i \leftrightarrow j$ is based on the objective function $f_i(x; \beta) + f_j(x; \beta)$. The probability that network x changes into $x^{(\pm ij)}$, now is given by

$$p_{ij}(x, \beta) = \frac{\exp\big(f_i(x^{(\pm ij)}; \beta) + f_j(x^{(\pm ij)}; \beta)\big)}{\exp\big(f_i(x; \beta) + f_j(x; \beta)\big) + \exp\big(f_i(x^{(\pm ij)}; \beta) + f_j(x^{(\pm ij)}; \beta)\big)} . \tag{86}$$

The two model components, rate function and objective function, can be put together by considering the transition rates. Given that the only permitted transitions between networks are toggles of a single tie variable, the transition rates can be defined as

$$q_{ij}(x) = \lim_{\Delta t \downarrow 0} \frac{\mathsf{P}\{X(t + \Delta t) = x^{(\pm ij)} \mid X(t) = x\}}{\Delta t} \tag{87}$$

for $i \neq j$. Note that this definition implies that the probabilities of toggling a particular tie variable X_{ij} in a short time interval are approximated by

$$\mathsf{P}\{X(t+\Delta t) = x^{(\pm ij)} \mid X(t) = x\} \approx q_{ij}(x)\,\Delta t\,.$$

In the derivation of the score functions account must be taken of the fact that toggling variable X_{ij} is the same as toggling X_{ji}, and the rules described above give different roles for the first and the second actor in the pair (i, j). For the models with one-sided initiative, the transition rate is

$$q_{ij}(x) = \lambda_i(x; \alpha, \rho_m)\, p_{ij}(x, \beta) + \lambda_j(x; \alpha, \rho_m)\, p_{ji}(x, \beta)\ , \tag{88}$$

and for the models with two-sided opportunity

$$q_{ij}(x) = \lambda_{ij}(x; \alpha, \rho_m)\, p_{ij}(x, \beta) + \lambda_{ji}(x; \alpha, \rho_m)\, p_{ji}(x, \beta)\ . \tag{89}$$

15.2 Score function for two-sided ties: objective function

The score function for the complete data likelihood with respect to the β parameters is as follows. (Explained here in shorthand for readers knowing theory and terminology from other papers.) It can be calculated as the sum of the contributions from all the ministeps, where ministeps leading to no change are also included. Therefore, we only need to give the expression for the score function contribution for a single ministep leading from x to $x^{(\pm ij)}$; and the contribution for a ministep leading to no change, but where the actor or pair of actors concerned in the ministep is known.

We must realize that the phrase “score function for the complete data likelihood” is ambiguous, because we have liberty to define what we take as complete data; perhaps ‘augmented data’ would be a better term. We can use any data augmentation that allows an easy calculation of the score functions. More extensive augmentation introduces extra variability and thereby extra noise in the score function. For example, in the models with one-sided initiative, if the data augmentation includes the knowledge of who took the initiative, then effectively we are working with the transition rates

$$q_{ij}(x) = \lambda_i(x; \alpha, \rho_m)\, p_{ij}(x, \beta) \tag{90}$$

instead of (**??**). We shall always do this. This leads to simpler but somewhat noisier score functions.

In the following, the complete/augmented data always includes the knowledge of the actor i who takes the initiative, and in the two-sided opportunity case the ordered pair (i, j) who make the ministep.

For the case M.1 we shall give two different expressions which differ in this way.

By $\mathcal{A}_-$ is denoted the set of potential alters in a given ministep and we denote $\mathcal{A} = \mathcal{A}_- \cup \{i\}$ where i is the actor making the ministep, but signifying here 'no change' so alternatively this could be denoted by 0. In the default case, $\mathcal{A} = \{1, \ldots, n\}$; if not all actors are active then $\mathcal{A}_-$ will contain only active actors; but it will be potentially a smaller set if one of the conditions *uponly* , *downonly* , *higher* , *disjoint* , or *atleastone* holds.

Denote the change in objective function when toggling x_{ij} by

$$(\Delta f)_{ij}(x, \beta) = f_i\big(x^{(\pm ij)}; \beta\big) - f_i(x; \beta) \ ,$$

which implies

$$(\Delta f)_{ii}(x, \beta) = 0.$$

Further denote

$$p_{1ij}(x, \beta) = \frac{\exp\big((\Delta f)_{ij}(x, \beta)\big)}{\sum_{h \in \mathcal{A}} \exp\big((\Delta f)_{ih})(x, \beta)\big)} \tag{91}$$

and

$$p_{0ij}(x, \beta) = \frac{\exp\big((\Delta f)_{ij}(x, \beta)\big)}{1 + \exp\big((\Delta f)_{ij})(x, \beta)\big)} \ . \tag{92}$$

In the following, we often use expressions such as

$$\frac{\partial (\Delta f)_{ij}(x, \beta)}{\partial \beta_k} \ .$$

Since the changes $\Delta f_{ij}(x, \beta)$ are linear combinations (**??**), these partial derivatives are the change contributions of the effects,

$$\frac{\partial (\Delta f)_{ij}(x, \beta)}{\partial \beta_k} = s_{ki}(x^{(\pm ij)}) - s_{ki}(x) \ . \tag{93}$$

This is denoted in shorthand by

$$s_{ijk} = s_{ki}(x^{(\pm ij)}) - s_{ki}(x) \ , \tag{94}$$

where the dependence on x is omitted. Note that $s_{iik} = 0$.

15.3 Dictatorial D.1 (alias A-1 alias AFORCE)

Probability of change, see (**??**)

$$p_{ij}(x,\beta) = \frac{\exp\big(f_i(x^{(\pm ij)};\beta)\big)}{\sum_{h\in\mathcal{A}}\exp\big(f_i(x^{(\pm ih)};\beta)\big)} = p_{1ij}(x,\beta)\ . \tag{95}$$

This is just the same as the directed case. Well-known calculus leads to the score function

$$\begin{aligned}\frac{\partial \log\, p_{ij}(x,\beta)}{\partial\beta_k} &= \frac{\partial \log\, p_{1ij}(x,\beta)}{\partial\beta_k}\\ &= \frac{\partial(\Delta f)_{ij}(x,\beta)}{\partial\beta_k} - \sum_{h\in\mathcal{A}} p_{1ih}(x,\beta)\frac{\partial(\Delta f)_{ih}(x,\beta)}{\partial\beta_k}\\ &= s_{ijk} - \sum_{h\in\mathcal{A}} p_{1ih}(x,\beta)\, s_{ihk}\ .\end{aligned} \tag{96a}$$

This formula also applies to the case of no change $j = i$, where it yields

$$\frac{\partial \log\, p_{ii}(x,\beta)}{\partial\beta_k} = -\sum_{h\in\mathcal{A}} p_{1ih}(x,\beta)\, s_{ijh}\ . \tag{96b}$$

15.4 Dictatorial D.2 (alias B-1 alias BFORCE)

Note that we condition here on the choice, in the opportunity process, of i and j, in this order: i is allowed to make a decision which then is imposed on j. The probability of change is, see (**??**),

$$p_{ij}(x,\beta) = \frac{\exp\big(f_i(x^{(\pm ij)};\beta)\big)}{\exp\big(f_i(x;\beta)\big) + \exp\big(f_i(x^{(\pm ij)};\beta)\big)} = p_{0ij}(x,\beta)\ . \tag{97}$$

Well-known calculus leads to the score function for change:

$$\begin{aligned}\frac{\partial \log\, p_{ij}(x,\beta)}{\partial\beta_k} &= \frac{\partial \log\, p_{0ij}(x,\beta)}{\partial\beta_k}\\ &= (1 - p_{0ij}(x,\beta))\frac{\partial(\Delta f)_{ij}(x,\beta)}{\partial\beta_k}\\ &= (1 - p_{0ij}(x,\beta))\, s_{ijk}\ ;\end{aligned} \tag{98a}$$

for no change:

$$\frac{\partial \log\, \big(1 - p_{ij}(x,\beta)\big)}{\partial\beta_k} = -p_{0ij}(x,\beta)\frac{\partial(\Delta f)_{ij}(x,\beta)}{\partial\beta_k} = -p_{0ij}(x,\beta)\, s_{ijk}\ . \tag{98b}$$

15.5 Mutual M.1 (alias A-2 alias AAGREE)

The probability of change is

$$p_{ij}(x,\beta) = \frac{\exp\left(f_i(x^{(\pm ij)};\beta)\right)}{\sum_{h\in\mathcal{A}}\exp\left(f_i(x^{(\pm ih)};\beta)\right)} \times \left(\frac{\exp\left(f_j(x^{(+ij)};\beta)+\beta^{\mathsf{b}}\right)}{\exp\left(f_j(x;\beta)\right)+\exp\left(f_j(x^{(+ij)};\beta)+\beta^{\mathsf{b}}\right)}\right)^{1-x_{ij}}$$

$$= p_{1ij}(x,\beta)\left(p_{0ji}(x,\beta)\right)^{(1-x_{ij})} \tag{99}$$

$$= p_{1ij}(x,\beta)\left(x_{ij}+(1-x_{ij})p_{0ji}(x,\beta)\right)\,, \tag{100}$$

where it may be noted that x_{ij} is the current state (before the possible change).

The outcome 'no change' may be further refined into the following list of possibilities:

[$i-$]: i decides not to propose any change.

[$ij-$]: (i proposes to j to add a tie, but this is rejected by j), for $j \in \mathcal{A}_-$.

The probability that there is no change is as follows (somewhat arbitrarily we denote this here by p_{i0})

$$p_{i0}(x,\beta) = 1 - \sum_{j\in\mathcal{A}_-} p_{1ij}(x,\beta)\left(x_{ij}+(1-x_{ij})p_{0ji}(x,\beta)\right)\,. \tag{101}$$

This can be decomposed into the sub-events as listed above, with probabilities

$$p_{[i-]}(x,\beta) = p_{1ii}(x,\beta)\,, \tag{102}$$

$$p_{[ij-]}(x,\beta) = (1-x_{ij})\big(1-p_{0ji}(x,\beta)\big)\,. \tag{103}$$

For the score functions, we combine (**??**) and (**??**). The score function for

a change is

$$\frac{\partial \log\, p_{ij}(x,\beta)}{\partial \beta_k} = \frac{\partial (\Delta f)_{ij}(x,\beta)}{\partial \beta_k} - \sum_{h\in\mathcal{A}} p_{1ih}(x,\beta)\frac{\partial (\Delta f)_{ih}(x,\beta)}{\partial \beta_k} + (1-x_{ij})(1-p_{0ji})\frac{\partial (\Delta f)_{ji}(x,\beta)}{\partial \beta_k}$$
$$= s_{ijk} - \left(\sum_{h\in\mathcal{A}} p_{1ih}(x,\beta) s_{ihk}\right) + (1-x_{ij})(1-p_{0ji}) s_{jik}\ . \tag{104a}$$

The score function for no change is a bit tedious,

$$\frac{\partial \log\, p_{i0}(x,\beta)}{\partial \beta_k} = \frac{-1}{p_{i0}(x,\beta)} \sum_{j\in\mathcal{A}_-} \Big\{ x_{ij}\frac{\partial p_{1ij}}{\partial \beta_k} + (1-x_{ij})\big(p_{0ji}\frac{\partial p_{1ij}}{\partial \beta_k} + p_{1ij}\frac{\partial p_{0ji}}{\partial \beta_k}\big)\Big\} \tag{104b}$$

where $\partial p_{0ij}/\partial\beta_k$ and $\partial p_{1ij}/\partial\beta_k$ must be substituted from (**??**) and (**??**). I do not think this can be simplified to an important extent.

If instead of 'no change' we work with the sub-events $[i-]$ = 'i wishes no change' and $[ij-]$ = 'i proposed to add a tie which is rejected by j', the score functions are

$$\frac{\partial \log\, p_{[i-]}(x,\beta)}{\partial \beta_k} = -\sum_{h\in\mathcal{A}} p_{1ih}(x,\beta)\, s_{ijh} \tag{104c}$$

as in (**??**), and

$$\frac{\partial \log\, p_{[ij-]}(x,\beta)}{\partial \beta_k} = (1-x_{ij})\left(s_{ijk} - \sum_{h\in\mathcal{A}} p_{1ih}(x,\beta)\, s_{ihk} - p_{0ji}(x,\beta)\, s_{jik}\right) \tag{104d}$$

which is much simpler indeed.

15.6 Mutual M.2 (alias B-2 alias BAGREE)

The probability of change is:
if $x_{ij} = 0$,

$$p_{ij}(x,\beta) = p_{0ij}(x,\beta)\, p_{0ji}(x,\beta) \tag{105a}$$

and if $x_{ij} = 1$,

$$p_{ij}(x,\beta) = p_{0ij}(x,\beta) + p_{0ji}(x,\beta) - p_{0ij}(x,\beta)\, p_{0ji}(x,\beta)\ . \tag{105b}$$

For the case $x_{ij} = 1$, it is easier to start working with the probability of no change,

$$1 - p_{ij}(x,\beta) = \big(1 - p_{0ij}(x,\beta)\big)\big(1 - p_{0ji}(x,\beta)\big)\ . \tag{105c}$$

This gives the following four cases:

1. $x_{ij} = 0$, change:

$$\frac{\partial \log\, p_{ij}(x,\beta)}{\partial \beta_k} = \big(1 - p_{0ij}(x,\beta)\big) s_{ijk} + \big(1 - p_{0ji}(x,\beta)\big) s_{jik}\ ; \tag{106a}$$

2. $x_{ij} = 0$, no change:

$$\begin{aligned}\frac{\partial \log\,(1 - p_{ij}(x,\beta))}{\partial \beta_k} &= \\ -\ \frac{p_{ij}(x,\beta)}{1 - p_{ij}(x,\beta)} &\left\{\big(1 - p_{0ij}(x,\beta)\big) s_{ijk} + \big(1 - p_{0ji}(x,\beta)\big) s_{jik}\right\}\ ;\end{aligned} \tag{106b}$$

3. $x_{ij} = 1$, change:

$$\frac{\partial \log\, p_{ij}(x,\beta)}{\partial \beta_k} = \frac{1 - p_{ij}(x,\beta)}{p_{ij}(x,\beta)} \left(p_{0ij}(x,\beta) s_{ijk} + p_{0ji}(x,\beta) s_{jik}\right)\ ; \tag{106c}$$

4. $x_{ij} = 1$, no change:

$$\frac{\partial \log\,(1 - p_{ij}(x,\beta))}{\partial \beta_k} = \ -p_{0ij}(x,\beta)\, s_{ijk} - p_{0ji}(x,\beta)\, s_{jik}\ . \tag{106d}$$

Here also a further decomposition would be possible, following the two consecutive choices by i and j, However, in this case this leads to more variability but not much more simplicity, so this is not elaborated now.

15.7 Compensatory C (alias B-3 alias BJOINT)

The probability of change is

$$p_{ij}(x,\beta) = \frac{\exp\big((\Delta f_i)(x^{(\pm ij)};\beta) + (\Delta f_j)(x^{(\pm ij)};\beta)\big)}{1+\exp\big((\Delta f_i)(x^{(\pm ij)};\beta) + (\Delta f_j)(x^{(\pm ij)};\beta)\big)} \ . \tag{107}$$

(Note that $x^{(\pm ij)} = x^{(\pm ji)}$, so indeed this is symmetric in i and j.)
Here it is convenient to define

$$p_{2ij}(x,\beta) = \frac{\exp\big((\Delta f)_{ij}(x,\beta) + (\Delta f)_{ji}(x,\beta)\big)}{1+\exp\big((\Delta f)_{ij})(x,\beta) + (\Delta f)_{ji}(x,\beta)\big)} \ . \tag{108}$$

Well-known calculus leads to the score function:
for change:

$$\begin{aligned}\frac{\partial \log\, p_{ij}(x,\beta)}{\partial \beta_k} &= \frac{\partial \log\, p_{2ij}(x,\beta)}{\partial \beta_k} \\ &= \big(1-p_{2ij}(x,\beta)\big)\,\frac{\partial (\Delta f)_{ij}(x,\beta) + (\Delta f)_{ji}(x,\beta)}{\partial \beta_k} \\ &= \big(1-p_{2ij}(x,\beta)\big)(s_{ijk}+s_{jik}) \ ; \end{aligned} \tag{109a}$$

for no change:

$$\frac{\partial \log\, \big(1-p_{ij}(x,\beta)\big)}{\partial \beta_k} = -p_{2ij}(x,\beta)\,(s_{ijk}+s_{jik}) \ . \tag{109b}$$

15.8 Coding the score calculations

The easiest way is to code the calculation of the scores in one routine together with the probabilistic choices; and decompose all choices into their components (sub-choices). For all five procedures this can be done using the following three basic procedures. Here $\mathcal{A}_-$ is the set of actors who are candidate alters for a given ministep (varies between ministeps), $\mathcal{A}$ is $\mathcal{A}_-$ to which an option 'no change' has been added (represented by i = the actor making the ministep, or 0), while $\mathcal{K}$ is the set indexing the parameters (always the same).

MultipleChoice (input: π, s; output: j; in/output: c).
Interpretation:

π = probability vector for outcomes $h \in \mathcal{A}$;
s = array of change contributions s_{hk} for $h \in \mathcal{A}, k \in \mathcal{K}$;
j = choice made;
c = vector of scores to which something will be added.
Body:
Define j as outcome of probabilistic choice according to π.
For all $k \in \mathcal{K}$ do $c_k \leftarrow c_k + s_{jk} - \sum_{h\in\mathcal{A}} \pi_h s_{hk}$.
EndBody.

BinaryChoice (input: π_0, s_0; output: r; in/output: c).
Interpretation:
π_0 = probability (number between 0 and 1);
s_0 = vector of change contributions s_k for $k \in \mathcal{K}$;
r = result $\in \{T, F\}$, where T denotes True = Accept and F denotes False = Reject;
c = vector of scores to which something will be added.
Body:
Define r as outcome of probabilistic choice according to π_0.
For all $k \in \mathcal{K}$ do: $\big($if r then $c_k \leftarrow c_k + (1-\pi)s_k$, else $c_k \leftarrow c_k - \pi\, s_k\big)$.
EndBody.

Change (input: i, j).
Interpretation:
Toggle tie $i \leftrightarrow j$.
Body:
If $j \in \mathcal{A}_-$ then $\big($set $x_{ij} \leftarrow 1 - x_{ij}$, $x_{ji} \leftarrow 1 - x_{ji}\big)$.
EndBody.

The models can can be coded as follows. We use p_{0ij}, p_{1ij}, s_{ijk} as defined in (**??**, **??**, **??**).

D.1 = A-1 Given i, use $\pi = p_{1i.}$, $s = s_{i..}$.
Apply **MultipleChoice** (π, s, j, c).
If $j \neq i$, apply **Change**(i, j).

D.2 = B-1 Given i and j in this order, use $\pi_0 = p_{0ij}$, $s_0 = s_{ijk}$.
Apply **BinaryChoice** (π_0, s_0, r, c).
If r, apply **Change**(i, j).

M.1 = A-2 Given i, use $\pi = p_{1i.}$, $s = s_{i..}$.
Apply **MultipleChoice** (π, s, j, c).
If $x_{ij} = 1$, apply **Change**(i, j);
else

> Use $\pi_0 = p_{0ji}$, $s_0 = s_{jik}$;
> Apply **BinaryChoice** (π_0, s_0, r, c);
> If r, apply **Change**(i, j).

M2 = B-2 One implementation is as follows:
Given i and j in arbitrary order:
Use $\pi_0 = p_{0ij}$, $s_0 = s_{ijk}$.
Apply **BinaryChoice** (π_0, s_0, r_A, c).
Use $\pi_0 = p_{0ji}$, $s_0 = s_{jik}$.
Apply **BinaryChoice** (π_0, s_0, r_B, c).
If $(x_{ij} = 0$ and r_A and $r_B)$ or $(x_{ij} = 1$ and $\big(r_A$ or $r_B\big))$, apply **Change**(i, j).

Another implementation, which integrates out over the double choice and therefore has a less variable score function and is preferable:
If $x_{ij} = 0$:

> Use $\pi_0 = p_{0ij}\, p_{0ji}$,
>
> $$s_0 = \frac{\big(1 - p_{0ij}(x, \beta)\big)\, s_{ijk} + \big(1 - p_{0ji}(x, \beta)\big)\, s_{jik}}{1 - p_{0ij}(x, \beta)\, p_{0ji}(x, \beta)};$$
>
> Apply **BinaryChoice** (π_0, s_0, r, c);
> If r, apply **Change**(i, j).

If $x_{ij} = 1$:

> Use $\pi_0 = 1 - \big(1 - p_{0ij}\big)\big(1 - p_{0ji}\big) = p_{0ij} + p_{0ji} - p_{0ij}p_{0ji}$,
>
> $$s_0 = \frac{p_{0ij}(x, \beta)\, s_{ijk} + p_{0ji}(x, \beta)\, s_{jik}}{p_{0ij}(x, \beta) + p_{0ji}(x, \beta) - p_{0ij}(x, \beta)p_{0ji}(x, \beta)};$$
>
> Apply **BinaryChoice** (π_0, s_0, r, c);
> If r, apply **Change**(i, j).

C = B-3 Given i and j in arbitrary order, use $\pi_0 = p_{2ij}$ as in (**??**), $s_0 = s_{ijk} + s_{jik}$.

Apply **BinaryChoice** (π_0, s_0, r, c).
If r, apply **Change**(i, j).

16 Score function: rate function

We consider here two types of rate function, for the one-sided and two-sided opportunity.

16.1 One-sided opportunity

For the one-sided opportunity (D.1 and M.1) the rate of change is

$$\lambda_i = \rho \, \exp\Big(\sum_k \alpha_k z_{ik} \Big) \tag{110}$$

where z_{ik} is a vector which is allowed to depend on the network x. A mini-step by actor i, when associated with an elapsed waiting time t, contributes to the probability density of the entire process a factor

$$\lambda_i e^{-t\,\lambda_+}$$

where $\lambda_+ = \sum_i \lambda_i$ is the total rate of change.

Calculus shows that the score function with respect to any parameter θ_k (α_k as well as ρ) is given by

$$\frac{\partial}{\partial \theta_k} \log\Big(\lambda_i e^{-t\,\lambda_+} \Big) = \frac{1}{\lambda_i} \frac{\partial \lambda_i}{\partial \theta_k} \; - \; t \frac{\partial \lambda_+}{\partial \theta_k} \; .$$

This yields the following contributions to the overall score function.

Since $\partial \lambda_i / \partial \rho = \lambda_i / \rho$, we get

$$\frac{\partial}{\partial \rho} \log\Big(\lambda_i e^{-t\,\lambda_+} \Big) = \frac{1 - t\lambda_+}{\rho} \; . \tag{111}$$

Since $\partial \lambda_i / \partial \alpha_k = z_{ik}\, \lambda_i$, we get

$$\frac{\partial}{\partial \alpha_k} \log\Big(\lambda_i e^{-t\,\lambda_+} \Big) = z_{ik} \; - \; t \sum_j z_{jk} \lambda_j \; , \tag{112}$$

where the $\sum_j$ is taken over the set of all active actors.

For variables z_{jk} that are not changing during the simulations, i.e., actor covariates that are not dependent behavior variables, the term $\sum_j z_{jk}\lambda_j$ does not change during the simulations, unless there is composition change and the set of active actors changes. For such variables, it is efficient to calculate $\sum_j z_{jk}\lambda_j$ at the start of each period, and at each composition change, but not at each ministep.

16.2 Two-sided opportunity

For the two-sided opportunity (D.2, M.1 and C) the pairwise rate of change is the product of two actor-dependent rates,

$$\lambda_{ij} = \lambda_{0i}\lambda_{0j} = \rho^2 \exp\Big(\sum_k \alpha_k(z_{ik} + z_{jk})\Big) \,. \tag{113}$$

(This may be extended later – this is how it was implemented in Siena 3.)

To avoid double summations over actors in the case where some variables X_{ij} are not allowed to change because of structurally missing values or having reached an absorbing state, we assume that the pairwise meetings take place according to (**??**) between all pairs (i, j) of actors subject to the conditions that i and j are active, and $i \neq j$. If a pair (i, j) meets which is not allowed to change anything, then time will advance, so there will be a contribution to the score function for the parameters of the rate function, but further nothing happens.

Ministep are made by all pairs of active actors i and j ($i \neq j$). When the elapsed waiting time for a ministep is t, the meeting event contributes to the probability density of the entire process a factor

$$\lambda_{ij} e^{-t\,\lambda_{++}}$$

where

$$\lambda_{++} = \sum_{i,j} \lambda_{ij} = \sum_{\text{active } i} \; \sum_{\text{active } j, j\neq i} \lambda_{0i}\lambda_{0j} \tag{114}$$

is the total rate of change, the summation being over all pairs (i, j) of active actors, $i \neq j$.

The pairwise meeting process accordingly proceeds as follows.

1. Define

$$\lambda_{0+} = \sum_{\text{active } i} \lambda_{0i} \ . \tag{115}$$

2. Repeat

   (a) Choose i among the active i with probabilities

   $$\frac{\lambda_{0i}}{\lambda_{0+}} \ ,$$

   (b) choose j among the active j with probabilities

   $$\frac{\lambda_{0j}}{\lambda_{0+}} \ ,$$

   until $i \neq j$.

3. Increase time by a value drawn from an exponential distribution with parameter (**??**).

4. If X_{ij} is allowed to change, then determine the new value according to its distribution; if X_{ij} is not allowed to change, continue (one could say that i and j meet but do not change how they are related).

The variable X_{ij} is not allowed to change if:

(X_{ij} is structurally fixed)
or (X_{ij} has reached an absorbing state because of a $*$*only* condition).

Here $*$*only* stands for any of *uponly* , *downonly* , *higher* , *disjoint* , or *atleast-one* .
(The last three conditions depend on other networks, and for those conditions the word 'absorbing' is perhaps not appropriate because when the other network changes the variable X_{ij} also might be allowed to change again.)
(Variable X_{ij} also cannot change if i or j is inactive, but this already is excluded in the choice of i and j.)

See below for a remark about the calculation of λ_{++}.

The rates λ_{ij} may depend on variables changing during the simulations, like in- or outdegrees or behavioral dependent variables. Then also λ_{++} will change as a consequence. In addition, λ_{++} can change during the simulations even if the λ_{ij} remain constant, namely, by a composition change.

The development further is analogous to the case of one-sided opportunity. The score function with respect to any parameter θ_k (α_k as well as ρ) is given by

$$\frac{\partial}{\partial \theta_k} \log\left(\lambda_{ij} e^{-t\,\lambda_{++}}\right) = \frac{1}{\lambda_{ij}} \frac{\partial \lambda_{ij}}{\partial \theta_k} \; - \; t \frac{\partial \lambda_{++}}{\partial \theta_k} \; .$$

This yields the following contributions to the overall score function.

Since $\partial \lambda_{ij} / \partial \rho = 2 \lambda_{ij} / \rho$, we get

$$\frac{\partial}{\partial \rho} \log\left(\lambda_{ij} e^{-t\,\lambda_{++}}\right) = 2\,\frac{1 - t\lambda_{++}}{\rho} \; . \tag{116}$$

Since $\partial \lambda_{ij} / \partial \alpha_k = (z_{ik} + z_{jk})\,\lambda_{ij}$, we get

$$\frac{\partial}{\partial \alpha_k} \log\left(\lambda_{ij} e^{-t\,\lambda_{++}}\right) = (z_{ik} + z_{jk}) \; - \; t \sum_{g,h} (z_{gk} + z_{hk}) \lambda_{gh} \; , \tag{117}$$

where the $\sum_{g,h}$ is taken over all pairs (g, h) where g as well as h are active and $g \neq h$. This can change because of a composition change, but also because of a change in $z_{gk} + z_{hk}$ or in λ_{gh}, which is possible if and only if λ_{gh} depends on endogenously changing variables.

16.2.1 Remark about calculation of the double summations

Some attention is needed for the terms λ_{++} and $\sum_{g,h} (z_{gk} + z_{hk}) \lambda_{gh}$, as these are defined as summations over $n(n-1)$ terms, where n is the number of active actors. We have

$$\begin{aligned}
\lambda_{++} &= \sum_{\text{active } i} \; \sum_{\text{active } j, j \neq i} \lambda_{0i} \lambda_{0j} \\
&= \sum_{\text{active } i} \lambda_{0i} \sum_{\text{active } j, j \neq i} \lambda_{0j} \; = \; \sum_{\text{active } i} \lambda_{0i} \left(\Big(\sum_{\text{active } j} \lambda_{0j} \Big) \; - \lambda_{0i} \right) \\
&= \Big(\sum_{\text{active } i} \lambda_{0i} \Big)^2 \; - \; \Big(\sum_{\text{active } i} \lambda_{0i}^2 \Big)
\end{aligned}$$

and

$$\begin{aligned}\sum_{g,h}(z_{gk}+z_{hk})\lambda_{gh} &= \sum_{\text{active } i}\ \sum_{\text{active } j, j\neq i}(z_{ik}+z_{jk})\lambda_{0i}\lambda_{0j}\\ &= \Big(\sum_{\text{active } i,j}(z_{ik}+z_{jk})\lambda_{0i}\lambda_{0j}\Big) - \Big(\sum_{\text{active } i} 2\,z_{ik}\lambda_{0i}^2\Big)\\ &= 2\Big(\sum_{\text{active } i}\lambda_{0i}\Big)\Big(\sum_{\text{active } i} z_{ik}\lambda_{0i}\Big) - 2\Big(\sum_{\text{active } i} z_{ik}\lambda_{0i}^2\Big)\,.\end{aligned}$$

Thus, everything can be expressed as a combination of single summations.

The remark at the end of Section **??** applies here, too: if the functions λ_{0i} are not changing during the simulations, i.e., either they do not depend on i or only through actor covariates that are not dependent behavior variables, these sums do not change during the simulations, unless there is composition change and the set of active actors changes. For such model specifications, it is efficient to calculate the sums at the start of each period, and at each composition change, but not again at each ministep because they do not change.

For model specifications where the rate function depends on endogenously changing variables, the sums will need to be calculated repeatedly.

17 Modeltype for behavior

This section describes the option `BehaviorModelType`, introduced in RSiena version 1.1-306.

The enumerated types `NetworkModelType` and `BehaviorModelType` are defined in `DependentVariable.h` as

```
enum NetworkModelType { NOTUSED, NORMAL, AFORCE, AAGREE,
BFORCE, BAGREE, BJOINT };
enum BehaviorModelType { OUTOFUSE, RESTRICT, ABSORB };
```

Functions `modelType()` and `modelType(type)` are defined in `NetworkLongitudinalData.h`, which is a class covering one dependent network.

Functions `behaviorModelType()` and `behaviorModelType(type)` are defined in `BehaviorLongitudinalData.h`, which is a class covering one dependent behavioral variable.

The model types are transferred from these classes to `NetworkVariable.h` and `BehaviorVariable.h` where they have the accessor functions `networkModelType` and `behaviorModelType`. This makes it possible to specify different model types for each dependent network or behavioral variable. The network model type then is used further in `NetworkVariable.cpp`. In `BehaviorVariable.cpp`, the behavioral model type then is used further through the function `behaviorModelTypeABSORB`.

17.1 Behavior micro-step

Whenever actor i may make a change in variable Z, she changes z_i to the new value v (changes can be –1, 0, +1).
Denote the new vector by $z(i \rightsquigarrow v)$. Change probabilities are given by

$$p_i(v; \beta, z, x) = \frac{\exp(f(i, v))}{\sum_{u \in \mathcal{C}} \exp(f(i, u))}$$

where

$$f(i, v) = f_i^Z(\beta, z(i \rightsquigarrow v), x) \ ,$$

f_i^Z is the objective function of actor i for behavior Z, and $\mathcal{C}$ is the set of allowed changes:
$\{-1, 0, 1\}$, $\{0, 1\}$, $\{-1, 0\}$,
depending on whether z_i currently is at a boundary of its range.

Thus, the range is restricted to the permissible values, and the objective function is evaluated accordingly. Therefore, this model option is called `RESTRICT`.

The new model option is called behavioral model type `ABSORB`.
It calculates, when z_i currently is at the boundary of the range, hypothetically, the objective function for changing to the next value *outside* the range; but if this value is chosen, then it is *absorbed* into the range of Z.

The choice between these options is specified by parameter `behModelType` in `sienaAlgorithmCreate`. This parameter can be given as an integer (1 or 2); or, for several dependent networks requiring different model types, as a named integer vector.

In the earlier only available option, which is the case `RESTRICT`, if $z_i = z^- = \min\{\text{range}(Z)\}$, the probabilities are

$$p_i(z^-; \beta, z^-, x) = \frac{\exp\big(f(i, z^-)\big)}{\exp\big(f(i, z^-)\big) + \exp\big(f(i, z^- + 1)\big)}$$

and

$$p(z^- + 1; \beta, z^-, x) = \frac{\exp\big(f(i, z^- + 1)\big)}{\exp\big(f(i, z^-)\big) + \exp\big(f(i, z^- + 1)\big)} \ .$$

Similarly, if $z_i = z^+ = \max\{\text{range}(Z)\}$, the probabilities are

$$p_i(z^+; \beta, z^+, x) = \frac{\exp\big(f(i, z^+)\big)}{\exp\big(f(i, z^+ - 1)\big) + \exp\big(f(i, z^+)\big)}$$

and

$$p(z^+ - 1; \beta, z^+, x) = \frac{\exp\big(f(i, z^+ - 1)\big)}{\exp\big(f(i, z^+ - 1)\big) + \exp\big(f(i, z^+ + 1)\big)} \ .$$

For the new option, the case `ABSORB`, if $z_i = z^- = \min\{\text{range}(Z)\}$, the probabilities are

$$p_i(z^-; \beta, z^-, x) = \frac{2\,\exp\big(f(i, z^-)\big)}{2\,\exp\big(f(i, z^-)\big) + \exp\big(f(i, z^- + 1)\big)}$$

and

$$p(z^- + 1; \beta, z^-, x) = \frac{\exp\big(f(i, z^- + 1)\big)}{2\,\exp\big(f(i, z^-)\big) + \exp\big(f(i, z^- + 1)\big)} \ .$$

Similarly, if $z_i = z^+ = \max\{\text{range}(Z)\}$, the probabilities are

$$p_i(z^+; \beta, z^+, x) = \frac{2\,\exp\big(f(i, z^+)\big)}{\exp\big(f(i, z^+ - 1)\big) + 2\,\exp\big(f(i, z^+)\big)}$$

and

$$p(z^+ - 1; \beta, z^+, x) = \frac{\exp\big(f(i, z^+ - 1)\big)}{\exp\big(f(i, z^+ - 1)\big) + 2\,\exp\big(f(i, z^+)\big)} \ .$$

17.2 Score function

To implement this requires more than changing the probabilities; also the score function J_θ needs to be changed. This is used for calculating

$$\frac{\partial \mathsf{E}_\theta Z}{\partial \theta} \;=\; \mathsf{E}_\theta\big\{ J_\theta \, Z \big\} \;.$$

For a change in the behavior variable, define by $\Delta_{ik}(d, z)$ the change statistic for effect k, actor i, current state z, difference d.

Define the change in the objective function by

$$\Delta f_i(d) \;=\; \sum_k \beta_k^Z \, \Delta_{ik}(d, z) \;.$$

Note that $\Delta f_i(0) = 0$, which will be used repeatedly in the sequel.

If there are no boundary effects, change probabilities are defined by

$$p_i(d) \;=\; \frac{\exp\big(\Delta f_i(d)\big)}{\sum_{d=-1}^{1} \exp\big(\Delta f_i(d)\big)} \;. \tag{118}$$

The scores for changes in behavioral variables are

$$J_k(d) \;=\; \frac{\partial}{\partial \beta_k} \log p_i^Z(d; z, \beta) \;=\; \Delta_{ik}(d, z) \;-\; \overline{\Delta_{ik}(., z)} \tag{119a}$$

where

$$\overline{\Delta_{ik}(., z)} \;=\; \sum_{d=-1}^{1} p_i(d) \, \Delta_{ik}(d, z) \;. \tag{119b}$$

In the standard model ('`RESTRICT`'), for the boundary cases:
if the current state is at the minimum, we have

$$\pi_i(-1) \;=\; 0 \;, \quad \pi_i(0) \;=\; \frac{1}{1 \,+\, \exp\big(\Delta f_i(1)\big)} \;, \tag{120a}$$

$$\pi_i(1) \;=\; \frac{\exp\big(\Delta f_i(1)\big)}{1 \,+\, \exp\big(\Delta f_i(1)\big)} \;; \tag{120b}$$

if the current state is at the maximum, we have

$$\pi_i(-1) \;=\; \frac{\exp\big(\Delta f_i(-1)\big)}{1 \,+\, \exp\big(\Delta f_i(-1)\big)}\;, \tag{120c}$$

$$\pi_i(0) \;=\; \frac{1}{1 \,+\, \exp\big(\Delta f_i(-1)\big)}\;, \quad \pi_i(1) \;=\; 0\;. \tag{120d}$$

The scores in the boundary cases still are given by (**??**) but with $p_i(d) = \pi_i(d)$ given in (**??**).

In the new model ('ABSORB'), the probabilities are:

if the current state is at the minimum,

$$\pi_i(-1) \;=\; 0\;, \quad \pi_i(0) \;=\; p_i(-1) \,+\, p_i(0)\;, \quad \pi_i(1) \;=\; p_i(1)\;; \tag{121a}$$

and if the current state is at the maximum,

$$\pi_i(-1) \;=\; p_i(-1)\;, \quad \pi_i(0) \;=\; p_i(0) \,+\, p_i(1)\;, \quad \pi_i(1) \;=\; 0\;. \tag{121b}$$

To calculate the scores in the second model type ('ABSORB'), for the boundary cases, we may note that this is based on a multinomial regression model with three options $\{-1, 0, 1\}$, of which the first two outcomes are combined. Consider the case for the right boundary. The first two outcomes have the same value $\Delta_{ik}(-1, z) \;=\; \Delta_{ik}(0, z) \;=\; 0$; the value is 0 because this option means no change. A score function is a function of the sufficient statistic, and for this 3-option statistical model the sufficient statistic corresponds to the partition of the outcome space into $\big\{\{-1, 0\}, \{1\}\big\}$. Therefore the score function that we need is again (**??**), for the original probabilities p_i. However, since $\Delta_{ik}(-1, z) = \Delta_{ik}(0, z) = 0$ and $\pi_i(1) = p_i(1)$, it does not matter whether we calculate (**??**) for p_i or for π_i.

18 sienaBayes

This is the start of work on documenting *sienaBayes* .
Incomplete, to be expanded.

18.1 Initialization

If a *prevBayes* object is supplied, this initialization phase is skipped.

1. Unless *prevAns=NULL* :
   Estimate using MoM with multigroup option, i.e., under assumption of common parameter values, using only 2 subphases. This yields the *sienaFit* object *startupGlobal* .
   Defaults can be changed by parameters *initgainGlobal* , *initgainGroupwise* , and *initML* .
   Stop if some of the estimated non-rate parameters are larger than 40.

2. If *priorPrecFactor* > 0, use a weighted mean of the parameter estimate in *startupGlobal* and the prior mean. This weighted mean is called the Kelley estimator[1].
   The weights are the inverses of the covariance matrices of the estimate in *startupGlobal* and of $\{$the prior for the global parameters, multiplied by *priorPrecFactor* $\}$. Names of variables related to this include the string *prec* .

3. For all groups seperately, if *initgainGroupwise* > 0, one subphase of the Robbins-Monro algorithm for MoM is executed, starting from the overall estimate, with step size *initgain* , to estimate the group-varying parameters.
   This provides get initial values *initialEstimates* per group, and covariance matrices *proposalCov* for proposal distributions per group.
   Stop if some of the estimated non-rate parameters are larger than 40.

[1] After the psychometrician Truman Lee Kelley

4. The scale of basic rate parameters can be easily modified, e.g., to the sqrt scale, by changing functions *trafo* , *antitrafo* , *devtrafo* .

19 Main algorithm

What follows is very incomplete.
It is an older text by Ruth Ripley.

1. Set up data in C as usual
2. Create minimal chain and do burnin
3. *improveMH* : Get scalefactors such that about 25 out of 100 Bayesian proposals after single MH steps are accepted. See below for details of generation and probabilities for Bayesian proposals. Keep theta unchanged throughout this step.
4. Do a warming phase of *nwarm* iterations of some number of MH steps.
5. Repeat step **??**.
6. Do requested number of Bayesian iterations. The length of the ML ones are determined by the multiplication factor and the observed distance.

Bayesian proposals

for all groups **do**

Create a mask to exclude basic rate effects for other periods than this group.

Get a multivariate normal with mean 0 and *proposalCov* $\times$ scale factor for this group

Calculate the proposal probability:

prior Multivariate normal density for the parameters with mean 0 and covariance as supplied in the input argument.

chain Add

1. sum of log probabilities of choice of variable/actor
2. sum of log choice probabilities
3. minus the sum of basic rate parameters times the relevant number of actors
4. sum of log(basic rate) parameter times the number of real steps in the chain for the corresponding variable.

(If not constant rates, use mu and sigma from the normal approximation instead.) Since chain does not change size, ignore the log factorial of chain length.

The log probability of acceptance is then new - old of log prior + log chain

end for

A The contribution of the number of ministeps to the likelihood

In **?**, a special role is played by the quantity

$$\kappa\big(\theta, x^{(0)}, (i_1, j_1), \ldots, (i_T, j_T)\big) \\ = P_\theta\{\text{time}_T \leq t_2 < \text{time}_{T+1} \mid x^{(0)}, (i_1, j_1), \ldots, (i_T, j_T)\} \,. \quad (122)$$

defined there with equation number (15), and given here with some notational changes to make it correspond better with the current paper. $T =$ *TotNumber* $- 1$ is the total number of 'real' ministeps;
and (i_t, j_t) indicates the option of the t'th network ministep. This was already used above in the acceptance probabilities for the Metropolis Hastings steps. We now elaborate its role for the score function.

The case is rather simple if the aggregate rate function

$$\sum_{\text{active } i} \left(\sum_{r=1}^{R_N} \lambda^N(\theta, r, i, y) + \sum_{r=1}^{R_B} \lambda^B(\theta, r, i, y) \right) \quad (123)$$

is constant: i.e., independent of the state y and of time; the time must be mentioned because even if this aggregate rate does not depend on y, a changing number of actors (changing composition) could make the sum time-dependent.

Let us denote the number of active actors, when assumed constant, by n_{act}, and

$$\lambda_{\text{ave}} = \frac{1}{n_{\text{act}}} \sum_{\text{active } i} \left(\sum_{r=1}^{R_N} \lambda^N(\theta, r, i, y) + \sum_{r=1}^{R_B} \lambda^B(\theta, r, i, y) \right) . \quad (124)$$

Note that formally we assume that all time durations are unity, $t_{m+1} - t_m = 1$. Then the total number of 'real' ministeps T has a Poisson distribution with parameter $n_{\text{act}}\lambda_{\text{ave}}$.

The latter also holds with changing composition, provided that

$$\lambda^+(\theta, +, i, y) = \lambda_{\text{ave}}$$

is independent of i as well as y, and that we let n_{act} denote the *average* number of active actors over the time period from t_m to t_{m+1}.

In this case (constant aggregate rate function), κ is equal to

$$\kappa\big(\theta, x^{(0)}, (i_1, j_1), \ldots, (i_T, j_T)\big) = \exp(-n_{\text{act}}\lambda_{\text{ave}})\frac{(n_{\text{act}}\lambda_{\text{ave}})^T}{T!} \ , \tag{125}$$

cf. (16) in **?**.

A.1 Score functions for rate parameters

If there is only a single dependent variable ($R_N + R_B = 1$) and ρ is the basic and only rate parameter, then $\lambda_{\text{ave}} = \rho$ and the score function for ρ is

$$\frac{\partial \log(\kappa)}{\partial \rho} = -n_{\text{act}} + \frac{T}{\rho} \ . \tag{126}$$

More generally, now suppose that R_N and R_B are arbitrary and the rate function is given by

$$\lambda^w(\theta, r, i, y) = \rho_r^w \ \exp(\alpha_r^w s_{ri}) \tag{127}$$

(where $\alpha_r^w s_{ri}$ is the inner product of these two vectors). Thus, there is a basic rate parameter ρ_r^w for each given dependent variable labeled (w, r); in addition there may be actor-dependent variables s_{ri} that affect the rate for dependent variable (w, r), but the parameters α_r^w are distinct for distinct dependent variables.

Then for the 'complete data' situation, the information is equivalent to the information in the variables T_{ri}^w, indicating the number of ministeps made of *OptionSet* (w, i, r). Denote $T_r^w = \sum_i T_{ri}^w$. The variables T_{ri}^w have independent Poisson distributions with parameter (**??**). Therefore the score functions can be derived from the Poisson distribution. The score functions are given for the basic rate parameters by

$$\frac{\partial \,\text{complete data log-likelihood}}{\partial \rho_r^w} = \frac{T_r^w}{\rho_r^w} - \sum_i \exp(\alpha_r^w s_{ri}) \ , \tag{128}$$

which for cases with *only* a basic rate parameter $\lambda^w(\theta, r, i, y) = \rho_r^w$ reduces to

$$\frac{\partial \,\text{complete data log-likelihood}}{\partial \rho_r^w} = \frac{T_r^w}{\rho_r^w} - n_{\text{act}} \ ; \tag{129}$$

and the score functions for the other rate parameters are

$$\frac{\partial \text{complete data log-likelihood}}{\partial \alpha_{rh}^{w}} = \sum_{i} s_{rih}\big(T_{ri}^{w} - \rho_{r}^{w}\ \exp(\alpha_{r}^{w} s_{ri})\big) \quad (130)$$

$$= \sum_{i} s_{rih}\big(T_{ri}^{w} - \lambda^{w}(\theta, r, i, y))\big)\ . \quad (131)$$