North American Actuarial Journal

ISSN: 1092-0277 (Print) 2325-0453 (Online) Journal homepage: https://www.tandfonline.com/loi/uaaj20

Simulation of Compound Hierarchical Models in R

Vincent Goulet PhD & Louis-Philippe Pouliot

To cite this article: Vincent Goulet PhD & Louis-Philippe Pouliot (2008) Simulation of Compound Hierarchical Models in R, North American Actuarial Journal, 12:4, 401-412, DOI: 10.1080/10920277.2008.10597532

To link to this article: https://doi.org/10.1080/10920277.2008.10597532

Published online: 28 Dec 2012.

Simulation of Compound Hierarchical Models in R

Vincent Goulet* and Louis-Philippe Pouliot†

Abstract

Hierarchical probability models are widely used in insurance applications for data classified in a tree-like structure and in Bayesian inference. We propose an R function to simulate data from compound models in which both the frequency component and the severity component can have a hierarchical structure. The model description method is based solely on R expressions, and it allows for models with any number of levels and nodes per level, as well as with very general conditional probability structures. The function is part of the R package **actuar.**

1. Introduction

Hierarchical probability models are widely used for data classified in a tree-like structure and in Bayesian inference. The main characteristic of such models is to have the probability law at some level in the classification structure be conditional on the outcome in previous levels. For example, adopting a bottom-to-top description of the model, a simple hierarchical model could be written as

$$\begin{aligned} X_t \mid \Lambda, \Theta &\sim \text{Poisson}(\Lambda), \\ \Lambda \mid \Theta &\sim \text{Gamma}(3, \Theta), \\ \Theta &\sim \text{Gamma}(2, 2), \end{aligned} \tag{1.1}$$

where X_t represents actual data.

In the broader graph theory sense, a hierarchical model is a model that can be represented by a directed acyclic graph (DAG). One special type of DAG is the tree: a graph in which any two nodes are connected by exactly one path. With this terminology, tree models are a subset of hierarchical models. However, in most practical actuarial applications and in the literature the two terms are understood as equivalent. Unless otherwise stated, we will use both without distinction.

Hierarchical models arise naturally in insurance applications. For example, they may be used to describe the probability structure of a portfolio of policies or as a means to incorporate collateral data from other cohorts, lines of business, or even companies in predictions (see Jewell 2004, 1975). In the actuarial literature the random variables Θ and Λ, above, are generally seen as uncertainty or risk parameters; in the sequel we refer to them more generally as mixing parameters.

This paper focuses on simulation of data for hierarchical models using the R statistical system (R Development Core Team 2008). R is a free software version of the award-winning S system; its commercial counterpart is S-PLUS by Insightful Corporation. R is not just another statistical environment, but a full-fledged and self-contained programming language with a strong mathematical orientation.

* Vincent Goulet, PhD, Associate Professor, École d'actuariat, Université Laval. Pavillon Alexandre-Vachon 1045, avenue de la Médecine, Local 1620, Québec (QC) G1V 0A6, Canada. vincent.goulet@act.ulaval.ca.

† Louis-Philippe Pouliot, Actuarial Analyst, Industrielle Alliance, Assurance et services financiers 1080, Grande-Allée Ouest Case Postale 1907, Succ. Terminus Québec (QC) G1K 7M3. louis-philippe.pouliot@inalco.com.

It is based on the notion of vector that the many actuaries who have worked, or are still working, with APL or SAS IML have come to know and love.

The example above is merely a multilevel mixture of models, something that is simple to simulate "by hand" in R. By virtue of vectorization, the following expression will yield n variates of the random variable X_t:

```
> rpois(n, rgamma(n, 3, rgamma(n, 2, 2))).
```

However, for tree-like categorical data common in actuarial applications there will usually be many categories—or *nodes*—at each level. Simulation is then complicated by the need to always use the correct parameters for each variate. In addition, actuaries often need to simulate separately the frequency and the severity of claims for compound models of the form

$$S = C_1 + \cdots + C_N, \tag{1.2}$$

where C_1, C_2, . . . are mutually independent and identically distributed random variables each independent of N.

There exist various software solutions to simulate data from hierarchical models, the most prominent being BUGS (Spiegelhalter, Thomas, and Best 2003); see also Scollnik (2001) for an excellent introduction from an actuarial perspective. At the time of this writing, OpenBUGS (Thomas et al. 2006) and JAGS (Plummer 2008a) appear to be the most current incarnations of the BUGS language. Both have R interfaces (Thomas et al. 2006; Plummer 2008b). However, using BUGS through R is often cumbersome (needing external files to save the description of the model), requires learning a new language, and is overkill if one does not intend to run a full Markov Chain Monte Carlo (MCMC) or Gibbs Sampler analysis.

We propose in this paper a method to describe very general tree models relying solely on R expressions. The method is flexible enough to encompass models with most types of hierarchical interactions of interest to actuaries and any number of nodes at each level.

We also demonstrate our implementation of the method in function `simul` of the R package **actuar** (Dutang, Goulet, and Pigeon 2008). The function is specifically designed to simulate data from compound models like (1.2) where both the frequency and the severity components can have a hierarchical structure. The function also supports weights (or volumes) in the model. We believe that very few, if any, other readily available software packages share these features.

Function `simul` should prove useful whenever random data for a portfolio of insurance contracts are needed. For example, Forgues, Goulet, and Lu (2006) and Belhadj, Goulet, and Ouellet (2008) relied on the function for their simulation studies in credibility theory. Function `aggregateDist` of the **actuar** package also calls `simul` internally to approximate by simulation the aggragate claim amount distribution in the classical collective risk model (Gerber 1979). Note, however, that the function was not designed for full-fledged MCMC analyses.

The paper is structured as follows: Section 2 introduces the model description method, and Section 3 describes the implementation in detail, with some usage examples.

2. DESCRIPTION OF HIERARCHICAL MODELS

Our aim was to develop a method to describe hierarchical models in R that would meet the following criteria:

1. Simple and intuitive to go from the mathematical formulation of the model to the R formulation and back
2. Allows for any number of levels and nodes
3. At every level, allows for any use of parameters higher in the hierarchical structure.

Fulfillment of this last item results in support for very general tree models; see Example 3.

A hierarchical model is completely specified by the number of nodes at each level and by the probability law at each level. One of the main difficulties is to express which of the previous parameters are used in the probability model of a given level and how.

Our proposed method is as follows. First, the number of nodes is specified by means of a named list in which each element is a vector of the number of nodes in each node of a given level. Indirectly, this also provides the number of levels in the model. Vectors are recycled (repeated) when the number of nodes is the same throughout a level.

Second, the probability model is expressed in a semisymbolic fashion using an object of mode "`expression`," a list-like object containing unevaluated R expressions. Each element of the object must be named—with names matching those of the number of nodes list—and should be a complete call to an existing random number generation function, but with the number of variates omitted. Hierarchical models are achieved by replacing one or more parameters of a distribution at a given level by any combination of the names of the levels above. If no mixing is to take place at a level, the model for this level can be `NULL`. All random variables that do not depend on the outcome of other variables should come first in the description. Their order is then unimportant.

Example 1

Consider the following expanded version of model (1.1):

$$\begin{aligned} X_{ijt}|\Lambda_{ij}, \Theta_i &\sim \text{Poisson}(\Lambda_{ij}), & t &= 1, \dots, n_{ij} \\ \Lambda_{ij}|\Theta_i &\sim \text{Gamma}(3, \Theta_i), & j &= 1, \dots, J_i \\ \Theta_i &\sim \text{Gamma}(2, 2), & i &= 1, \dots, I, \end{aligned}$$

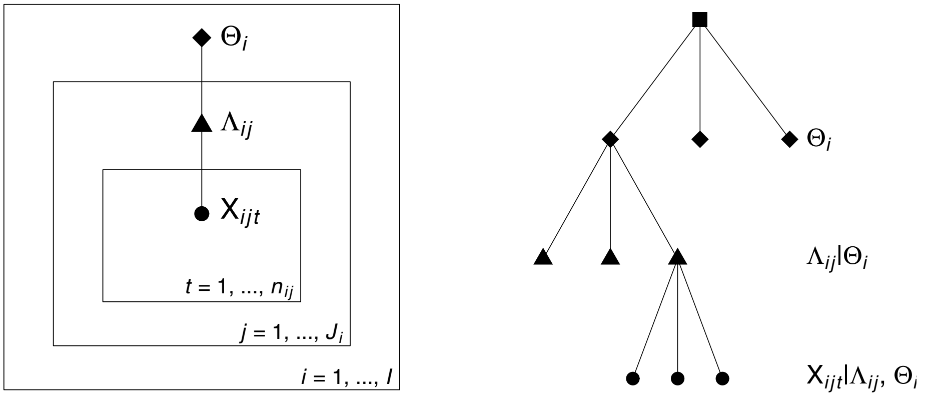
with $I = 3$, $J_1 = 4$, $J_2 = 5$, $J_3 = 6$, and $n_{ij} \equiv n = 10$. Hence, the first level has three nodes, these nodes are split, respectively, into four, five, and six nodes at the second level, and the last level (the data) has 10 nodes in every branch of the tree. Two schematic representations of this model are provided in Figure 1. Figure 1a uses the very convenient plate notation also employed with BUGS. In this notation a rectangle is a metaphor for repetition. Figure 1b is a standard tree representation.

Following the rules explicited above, the number of nodes is specified by

```
> list(Theta = 3, Lambda = c(4, 5, 6), Data = 10),
```

Figure 1

Schematic Representations of the Model of Example 1



(a) Plate notation

(b) Equivalent tree representation

and the probability model is expressed as

```
> expression(Theta = rgamma(2, 2),
+            Lambda = rgamma(3, Theta).
+            Data = rpois(Lambda))
```

□

Storing the probability model requires an expression object to avoid evaluation of the incomplete calls to the random number generation functions. A simulation function based on this model description method will build and execute the calls to the random generation functions from the top of the hierarchical model to the bottom. At each level the function has to (1) infer the number of variates to generate from the number of nodes list and (2) appropriately recycle the mixing parameters simulated previously.

The actual names of the levels are immaterial; they serve only to identify the mixing parameters. Furthermore, any random generation function can be used. The only constraint in our implementation is that the name of the number of variates argument be `n`.

Function `simul` also supports usage of weights in models. These usually modify the frequency parameters to take into account the ‘‘size’’ of an entity. Weights are used in simulation wherever the name `weights` appears in a model.

3. Implementation

We implemented the model description method of the previous section in function `simul` of **actuar** (Dutang, Goulet, and Pigeon 2008), a package providing additional actuarial science functionality to R. The software is distributed through the Comprehensive R Archive Network (CRAN; http://cran.r-project.org).

Function `simul` can simulate data from compound models of the form

$$S = C_1 + \cdots + C_N,$$

where both the frequency and the severity components are hierarchical. The function has four main arguments: (1) `nodes` for the number of nodes list, (2) `model.freq` for the frequency model, (3) `model.sev` for the severity model, and (4) `weights` for the vector of weights in lexicographic order, that is, all weights of entity 1, then all weights of entity 2, and so on. It is the user's responsibility to ensure that the length of `weights` matches the number of nodes when weights are to be used.

Notice that the function was not designed to take care of the simulation of the weights. In practice, the actuary may have real exposures (number of claims, payroll, surface, etc.) at his disposal. Using these to simulate claims data improves the realism of the resulting portfolio. Otherwise, weights are simple to simulate by hand, as we shall see in the examples below.

Function `simul` first simulates and appropriately recycles the frequency mixing parameters at each level and the frequencies per node at the lowest level. Then the same operation is repeated with the severity model, except that the number of nodes at the lowest level is now given by the frequencies just simulated, not by the entries in the `nodes` list. This requires one additional recycling of the mixing parameters.

The most obvious type of storage for the simulated data is a *k*-way array, where *k* is the number of levels in the model. However, array storage rapidly becomes inconvenient and inefficient when the number of nodes at each level is different: the array ends up filled with missing values (`NA`) to patch shorter dimensions.

Hence, in the long run it seems more appropriate to store the data in a two-dimensional object, much like a database or a spreadsheet. In the present context, the rows of the object will contain the "entities" of the classification structure, and the columns will contain the nodes of the last level, usually the periods of observation. However, for the sake of generality, we wanted function `simul` to return the individual claim amounts C_j for each entity. Thus, we opted to store the results in a list of class "portfolio" with a dim attribute of length two. Since every element can be a vector, the object can be seen as a three-dimensional array with a third dimension of potentially varying length. The function also returns a matrix of integers giving the classification indexes of each entity in the portfolio.

The **actuar** package also defines four summary methods to easily access key quantities for each entity of the simulated portfolio:

1. A method of aggregate to compute the aggregate claim amount S
2. A method of frequency to compute the number of claims N
3. A method of severity (a generic function introduced by the package) to return the individual claim amounts C_j
4. A method of weights to extract the weights matrix.

In addition, all methods have a classification and a prefix argument. When the first is FALSE, the classification index columns are omitted from the result. The second argument overrides the default column name prefix; see the simul.summaries help page in the package for details.

The following example illustrates these concepts in detail.

EXAMPLE 2

Consider an insurance portfolio where contracts are classified into ''cohorts'' for rate-making purposes. This is common practice in workers compensation or automobile insurance, for example, although usually with more levels. Now, say we need to simulate claims data for this portfolio according to the following probability model:

$$S_{ijt} = C_{ijt1} + \cdots + C_{ijtN_{ijt}}$$

for $i = 1, \ldots, I$, $j = 1, \ldots, J_i$, $t = 1, \ldots, n_{ij}$, with

$$\begin{aligned} N_{ijt}|\Lambda_{ij}, \Phi_i &\sim \text{Poisson}(w_{ijt}\Lambda_{ij}) & C_{ijtu}|\Theta_{ij}, \Psi_i &\sim \text{Lognormal}(\Theta_{ij}, 1) \\ \Lambda_{ij}|\Phi_i &\sim \text{Gamma}(\Phi_i, 1) & \Theta_{ij}|\Psi_i &\sim N(\Psi_i, 1) \\ \Phi_i &\sim \text{Exponential}(2) & \Psi_i &\sim N(2, 0.1), \end{aligned}$$

and the w_{ijt}s are known, fixed weights. Here the random variables Φ and Ψ represent incertainty due to the cohort, whereas Λ and Θ represent incertainty due to the contract. Using as per convention to number the data level 0, the above is a two-level compound hierarchical model.

Assuming that $I = 2$, $J_1 = 4$, $J_2 = 3$, $n_{11} = \cdots = n_{14} = 4$ and $n_{21} = n_{22} = n_{23} = 5$ and that weights are simply simulated from a uniform distribution on (0.5, 2.5), then simulation of a data set with simul is achieved with

```
> nodes <- list(cohort = 2,
+               contract = c(4, 3),
+               year = c(4, 4, 4, 4, 5, 5, 5))
> mf <- expression(cohort = rexp(2),
+                  contract = rgamma(cohort, 1),
+                  year = rpois(weights * contract))
> ms <- expression(cohort = rnorm(2, sqrt(0.1)),
+                  contract = rnorm(cohort, 1),
+                  year = rlnorm(contract , 1))
> wijt <- runif(31, 0.5, 2.5)
> pf <- simul(nodes = nodes, model.freq = mf,
+             model.sev = ms, weights = wijt)
```

Object pf is a list of class "portfolio" containing, among other things, the aforementioned two-dimensional list as element data and the classification matrix (subscripts i and j) as element classification:

```
> class(pf)

[1] “portfolio”

> pf$data

     year.1    year.2    year.3    year.4    year.5
[1,] Numeric,2 Numeric,2 11.38     Numeric,0 NA
[2,] Numeric,0 Numeric,0 Numeric,0 Numeric,0 NA
[3,] Numeric,0 Numeric,3 Numeric,0 Numeric,2 NA
[4,] Numeric,0 98.13     50.62     55.7      NA
[5,] Numeric,0 11.79     2.253     2.397     Numeric,2
[6,] Numeric,0 Numeric,0 Numeric,0 Numeric,0 Numeric,0
[7,] Numeric,3 Numeric,4 Numeric,2 Numeric,2 Numeric,0

> pf$classification

     cohort contract
[1,]      1        1
[2,]      1        2
[3,]      1        3
[4,]      1        4
[5,]      2        1
[6,]      2        2
[7,]      2        3
```

The output of `pf$data` is not very readable. Printing the results of `simul` like this would bring many users to wonder what `Numeric,`*n* means. It is actually R’s way to specify that a given element in the list is a numeric vector of length *n*—the third dimension mentioned above. To ease reading, the print method for objects of class `"portfolio"` prints only the simulation model and the number of claims in each node:

```
> pf

Portfolio of claim amounts

 Frequency model
  cohort   ~ rexp(2)
  contract ~ rgamma(cohort, 1)
  year     ~ rpois(weights * contract)
 Severity model
  cohort   ~ rnorm(2, sqrt(0.1))
  contract ~ rnorm(cohort, 1)
  year     ~ rlnorm(contract, 1)

Number of claims per node:

     cohort contract year.1 year.2 year.3 year.4 year.5
[1,]      1        1      2      2      1      0     NA
[2,]      1        2      0      0      0      0     NA
[3,]      1        3      0      3      0      2     NA
[4,]      1        4      0      1      1      1     NA
[5,]      2        1      0      1      1      1      2
[6,]      2        2      0      0      0      0      0
[7,]      2        3      3      4      2      2      0
```

Depending on the application, one may need either one or all of the aggregate claim amounts S_{ijt}, the numbers of claims N_{ijt}, and the individual claim amounts C_{ijtu}. The extractor functions `aggregate`, `frequency`, and `severity` allow easy access to these quantities.

By default, the method of `aggregate` returns the values of S_{ijt} in a regular matrix (subscripts i and j in the rows, subscript t in the columns):

```
> aggregate(pf)

     cohort contract year.1 year.2 year.3 year.4 year.5
[1,]      1        1  31.37  7.521 11.383  0.000     NA
[2,]      1        2   0.00  0.000  0.000  0.000     NA
[3,]      1        3   0.00 72.706  0.000 23.981     NA
[4,]      1        4   0.00 98.130 50.622 55.705     NA
[5,]      2        1   0.00 11.793  2.253  2.397  10.48
[6,]      2        2   0.00  0.000  0.000  0.000   0.00
[7,]      2        3  44.81 88.737 57.593 14.589   0.00
```

The method has a `by` argument to get statistics for other groupings and a `FUN` argument to get statistics other than the sum. For example, one can quickly obtain the average claim amount per cohort and per year with the following:

```
> aggregate(pf, by = c(“cohort”, “year”), FUN = mean)

     cohort year.1 year.2 year.3 year.4 year.5
[1,]      1  15.69  29.73  31.00 26.562     NA
[2,]      2  14.94  20.11  19.95  5.662  5.238
```

The method of `frequency` returns the values of N_{ijt}. It is mostly a wrapper for the aggregate method with the default `sum` statistic replaced by `length`. Hence, arguments `by` and `FUN` remain available:

```
> frequency(pf)

     cohort contract year.1 year.2 year.3 year.4 year.5
[1,]      1        1      2      2      1      0     NA
[2,]      1        2      0      0      0      0     NA
[3,]      1        3      0      3      0      2     NA
[4,]      1        4      0      1      1      1     NA
[5,]      2        1      0      1      1      1      2
[6,]      2        2      0      0      0      0      0
[7,]      2        3      3      4      2      2      0

> frequency(pf, by = “cohort”)

     cohort freq
[1,]      1   13
[2,]      2   16
```

The method of `severity` returns the individual variates C_{ijtu} in a matrix similar to those above, but with a number of columns equal to the maximum number of observations per entity,

$$\max_{ij} \sum_{t=1}^{n_{ij}} N_{ijt}.$$

Thus, the original period of observation (subscript t) and the identifier of the severity within the period (subscript u) are lost and each variate now constitute a “period” of observation:

```
> severity(pf)

$main
      cohort contract claim.1 claim.2  claim.3  claim.4 claim.5
[1,]       1        1   7.974  23.401    3.153    4.368  11.383
[2,]       1        2      NA      NA       NA       NA      NA
[3,]       1        3   3.817  41.979   26.910    4.903  19.078
[4,]       1        4  98.130  50.622   55.705       NA      NA
[5,]       2        1  11.793   2.253    2.397    9.472   1.004
[6,]       2        2      NA      NA       NA       NA      NA
[7,]       2        3  14.322  11.522   18.966   33.108  15.532
     claim.6  claim.7 claim.8 claim.9 claim.10 claim.11
[1,]      NA       NA      NA      NA       NA       NA
[2,]      NA       NA      NA      NA       NA       NA
[3,]      NA       NA      NA      NA       NA       NA
[4,]      NA       NA      NA      NA       NA       NA
[5,]      NA       NA      NA      NA       NA       NA
[6,]      NA       NA      NA      NA       NA       NA
[7,]   14.99    25.11   40.15   17.44    4.426    10.16

$split
NULL
```

One may need to extract separately the individual claim amounts of one or more periods. For this purpose the `severity` method provides an argument `splitcol`. For example, to extract the claim amounts of the first year:

```
> severity(pf, splitcol = 1)

$main
      cohort contract claim.1 claim.2 claim.3 claim.4 claim.5
[1,]       1        1   3.153   4.368  11.383      NA      NA
[2,]       1        2      NA      NA      NA      NA      NA
[3,]       1        3   3.817  41.979  26.910   4.903  19.078
[4,]       1        4  98.130  50.622  55.705      NA      NA
[5,]       2        1  11.793   2.253   2.397   9.472   1.004
[6,]       2        2      NA      NA      NA      NA      NA
[7,]       2        3  33.108  15.532  14.990  25.107  40.150
     claim.6  claim.7 claim.8
[1,]      NA       NA      NA
[2,]      NA       NA      NA
[3,]      NA       NA      NA
[4,]      NA       NA      NA
[5,]      NA       NA      NA
[6,]      NA       NA      NA
[7,]   17.44    4.426   10.16

$split
     cohort contract claim.1 claim.2 claim.3
[1,]      1        1   7.974   23.40      NA
[2,]      1        2      NA      NA      NA
[3,]      1        3      NA      NA      NA
```

```
[4,]      1       4      NA      NA      NA
[5,]      2       1      NA      NA      NA
[6,]      2       2      NA      NA      NA
[7,]      2       3  14.322   11.52   18.97
```

Finally, the weights matrix corresponding to the data in object `pf` is

```
> weights(pf)

     cohort contract year.1 year.2 year.3 year.4 year.5
[1,]      1        1 0.8361  2.115 1.2699 1.1555     NA
[2,]      1        2 1.7042  1.709 0.7493 1.0892     NA
[3,]      1        3 1.6552  1.762 1.5240 1.5100     NA
[4,]      1        4 1.5681  1.614 2.2358 2.1594     NA
[5,]      2        1 0.7229  1.907 2.2950 1.0595 0.9564
[6,]      2        2 0.5307  0.758 0.6868 0.9738 2.0823
[7,]      2        3 1.6995  2.320 1.6208 2.0114 1.2583
```

Combined with the argument `classification = FALSE`, the above methods can be used to easily compute loss ratios S_{ijt}/w_{ijt}:

```
> aggregate(pf, classif = FALSE)/weights(pf, classif = FALSE)

     year.1 year.2  year.3 year.4 year.5
[1,]  37.53  3.556  8.9638  0.000     NA
[2,]   0.00  0.000  0.0000  0.000     NA
[3,]   0.00 41.264  0.0000 15.881     NA
[4,]   0.00 60.781 22.6412 25.796     NA
[5,]   0.00  6.183  0.9818  2.263  10.95
[6,]   0.00  0.000  0.0000  0.000   0.00
[7,]  26.37 38.244 35.5328  7.253   0.00
```

□

The following example shows that function `simul` supports more than multilevel models.

EXAMPLE 3

Scollnik (2001) considers the following model for the simulation of claims frequency data in a MCMC context:

$$\begin{aligned} S_{it}|\Lambda_i, \alpha, \beta &\sim \text{Poisson}(w_{it}\Lambda_i), \\ \Lambda_i|\alpha, \beta &\sim \text{Gamma}(\alpha, \beta), \\ \alpha &\sim \text{Gamma}(5, 5), \\ \beta &\sim \text{Gamma}(25, 1) \end{aligned}$$

for $i = 1, 2, 3$, $t = 1, \ldots, 5$, and with weights w_{it} simulated from

$$\begin{aligned} w_{it}|a_i, b_i &\sim \text{Gamma}(a_i, b_i), \\ a_i &\sim U(0, 100), \\ b_i &\sim U(0, 100). \end{aligned}$$

For illustration purposes, we will take the simulation of weights as an integral part of the model. Figure 2a shows the plate notation of the simulation model. To use `simul`, one simply has to find the equivalent tree representation shown in Figure 2b. The call is then

```
> x <- simul(nodes = list(alpha = 1, beta = 1, Lambda = 3,
+               a   = 1, b = 1, w = 5, year = 1),
+            model.freq = expression(alpha = rgamma(5, 5),
+               beta = rgamma(25, 1),
+               Lambda = rgamma(alpha, beta),
+               a = runif(0, 100),
+               b = runif(0, 100),
+               w = rgamma(a, b),
+               year = rpois(w * Lambda)))
```

A call to `frequency` yields the variates, in this case a 15 × 1 matrix. It may be more convenient for further use to reorganize the data as follows:

```
> matrix(frequency(x, classification = FALSE),
+     nrow = 3, ncol = 5, byrow = TRUE)

     [,1] [,2] [,3] [,4] [,5]
[1,]    0    0    0    0    1
[2,]    1    0    0    0    1
[3,]    0    2    0    0    1
```

Of course, one can also easily simulate the weights in a separate step with a single R expression:

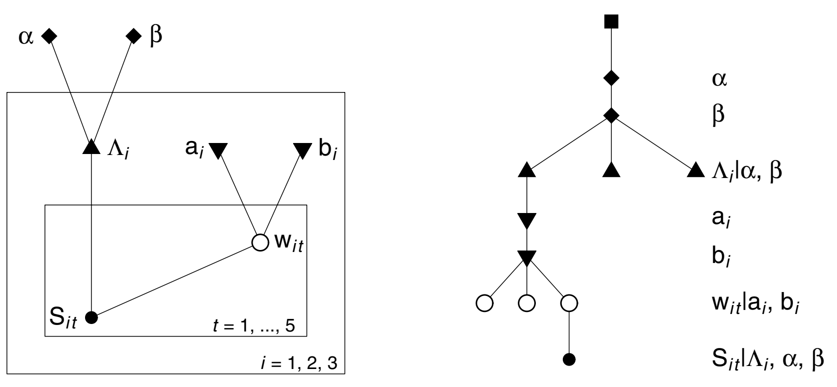
```
> wit <- rgamma(15, rep(runif(3, 0, 100), each = 5),
+                   rep(runif(3, 0, 100), each = 5))
```

Feeding these weights to `simul` results in a simpler call and straightforward extraction of the variates:

```
> x <- simul(list(a = 1, b = 1, Lambda = 3, year = 5),
+            expression(a = rgamma(5, 5),
+                       b = rgamma(25, 1),
+                       Lambda = rgamma(a, b),
```

Figure 2

Schematic Representations of the Model of Example 3



(a) Plate notation (b) Equivalent tree representation

```
+                          year = rpois(weights * Lambda)),
+            weights = wit)
```

```
> frequency(x)
```

```
     a b Lambda year.1 year.2 year.3 year.4 year.5
[1,] 1 1      1      0      0      0      0      0
[2,] 1 1      2      0      0      0      0      0
[3,] 1 1      3      0      1      0      1      1
```

□

One will find more examples of `simul` usage in the `simulation` demo file of the **actuar** package. Type

```
> demo(simulation, package = “actuar”)
```

at the R prompt to run the demo.

4. CONCLUSION

This paper introduced an R function to simulate data from compound hierarchical (tree) models. The function is based on a model description method relying on two objects: a list giving the number of nodes in each level of the model, and an expression object containing the probability model at each level. The probability models are expressed in a semisymbolic fashion as calls to random number generation functions with the number of variates omitted. The function is quite flexible, allowing for any kind of interaction between mixing parameters at every level of the hierarchy.

To make the function even more useful for actuarial applications, we would like to extend support to a wider range of random effects models; crossed classification models (Dannenburg, Kaas, and Goovaerts 1996) come to mind here. It is not clear if we may be able to go as far as the full range of models supported by BUGS. This will need further investigation.

The **actuar** package is released under the GNU General Public License (www.fsf.org/licensing/licenses/gpl.html). In a nutshell this means that anyone is free to use and modify our code, provided that any published derivative work is also released under the GPL. Therefore, the model description method or the `simul` function can be adapted to special needs or contexts other than insurance.

Finally, if R or **actuar** is used for actuarial analysis, the software should be cited in publications. Use

```
> citation()
```

and

```
> citation(“actuar”)
```

at the R command prompt for information on how to cite the software.

5. ACKNOWLEDGMENTS

This research benefited from financial support from the Natural Sciences and Engineering Research Council of Canada and from the Chaire d’actuariat (Actuarial Science Chair) of Université Laval. The authors also thank an anonymous referee and a co-editor for improvements to the paper.

REFERENCES

BELHADJ, H., V. GOULET, AND T. OUELLET. 2008. On parameter estimation in Hierarchical Credibility. *ASTIN Bulletin,* submitted.

DANNENBURG, D. R., R. KAAS, AND M. J. GOOVAERTS. 1996. *Practical Actuarial Credibility Models*. Leuven: Ceuterick.

DUTANG, C., V. GOULET, AND M. PIGEON. 2008. **actuar:** An R package for Actuarial Science. *Journal of Statistical Software* 25(7). www.jstatsoft.org/v25/i07.

FORGUES, A., V. GOULET, AND J. LU. 2006. Credibility for Severity Revisited. *North American Actuarial Journal* 10(1): 49–62.

GERBER, H. U. 1979. *An Introduction to Mathematical Risk Theory*. Philadelphia: Huebner Foundation.

JEWELL, W. 1975. The Use of Collateral Data in Credibility Theory: A Hierarchical Model. *Giornale dell'Istituto Italiano degli Attuari* 38: 1–16.

———. 2004. Bayesian Statistics. In J. L. Teugels and B. Sundt, (eds.), *Encyclopedia of Actuarial Science,* vol. 1, pp. 153–166. New York: John Wiley.

PLUMMER, M. 2008a. *JAGS Version 1.0.2 Manual*. Lyon: International Agency for Research on Cancer. www-ice.iarc.fr/~martyn/software/jags.

———. 2008b. ***rjags:*** *Bayesian Graphical Models Using MCMC.* R package version 1.0.2-3. http://mcmc-jags.sourceforge.net.

R DEVELOPMENT CORE TEAM. 2008. *R: A Language and Environment for Statistical Computing*. Vienna: R Foundation for Statistical Computing. www.r-project.org.

SCOLLNIK, D. P. M. 2001. Actuarial Modeling with MCMC and BUGS. *North American Actuarial Journal* 5(2): 96–124.

SPIEGELHALTER, D. J., A. THOMAS, AND N. G. BEST. 2003. *WinBUGS Version 1.4 User Manual*. Cambridge: MRC Biostatistics Unit.

THOMAS, A., B. O'HARA, U. LIGGES, AND S. STURTZ. 2006. Making BUGS Open. *R News* 6(1): 12–17. http://cran.r-project.org/doc/Rnews.

Discussions on this paper can be submitted until April 1, 2009. The authors reserve the right to reply to any discussion. Please see the Submission Guidelines for Authors on the inside back cover for instructions on the submission of discussions.