A Heuristic for Moment-matching Scenario Generation

Høyland, Kjetil and Kaut, Michal and Wallace, Stein W.
SINTEF Technology and Society, Trondheim, Norway

in: Computational Optimization and Applications. See also BibTeX entry below.

BibTeX:

```
@article{HoylandEA00,
  author = {H{\o}yland, Kjetil and Kaut, Michal and Wallace, Stein W.},
  title = {A Heuristic for Moment-matching Scenario Generation},
  journal = {Computational Optimization and Applications},
  year = {2003},
  volume = {24},
  pages = {169--185},
  number = {2--3},
  doi = {10.1023/A:1021853807313},
}
```


Document created on October 21, 2012.
Cover page generated with CoverPage.sty.

A Heuristic for Moment-matching Scenario Generation

Kjetil Høyland
kjetil.hoyland@gjensidige.no*

Michal Kaut
michal.kaut@iot.ntnu.no†

Stein W. Wallace
stein.w.wallace@himolde.no‡

June 20, 2000; revised May 11, 2001, August 12, 2002

In stochastic programming models we always face the problem of how to represent the random variables. This is particularly difficult with multidimensional distributions. We present an algorithm that produces a discrete joint distribution consistent with specified values of the first four marginal moments and correlations. The joint distribution is constructed by decomposing the multivariate problem into univariate ones, and using an iterative procedure that combines simulation, Cholesky decomposition and various transformations to achieve the correct correlations without changing the marginal moments.

With the algorithm, we can generate 1000 one-period scenarios for 12 random variables in 16 seconds, and for 20 random variables in 48 seconds, on a Pentium III machine.

Keywords: stochastic programming, scenario tree generation, Cholesky decomposition, heuristics

1. Introduction

Gjensidige Nor Asset Management (GNAM) has NOK 65 billion (7 billion US$) under management. During the last few years they have used stochastic-programming-based asset allocation models in their asset allocation processes. The most important step in that process is to establish the market expectations, i.e. to establish what are their beliefs for the major asset categories (bonds, stocks, cash, commodities and currencies) in different major regions of the world. The decision-makers prefer to express their expectations in terms of marginal distributions of the return / interest rates for the different asset

*Gjensidige Nor Asset Management, POBox 276, N-1326 Lysaker, Norway
†Dept. of Industrial Economics and Technology Management, NTNU, N-7491 Trondheim, Norway
‡Molde University College, Servicebox 8, N-6405 Molde, Norway

classes in addition to correlations. The challenge in converting these expectations into an asset allocation mix is twofold: First we need to convert the expectations into a format which a stochastic programming model can handle, second we need an optimization model which gives us the optimal asset mix, given this input.

Practical experience has told us that the first part, the scenario generation, can in fact be the most challenging and (computer-) time consuming one. For the purpose of generating the input data, GNAM has been using the method described in (Høyland and Wallace, 2001), a method developed in 1996. For larger problems, with many asset classes, the scenario generation became the bottleneck in the portfolio optimization process. This paper presents an algorithm that reduces the computing time for the scenario generation substantially.

The most well-known applications of asset allocation models are the Russell-Yasuda-Kasai model in (Cariño and Ziemba, 1998) and the models implemented by Towers Perrin in (Mulvey, 1996). Other applications can be found in (Consigli and Dempster, 1998) and (Dert, 1995).

These models are all focused on long term strategic asset liability planning. Stochastic processes are widely used for scenario generation in such models. The big challenge with such processes is to calibrate the parameters so that the generated scenarios are consistent with the decision-maker's beliefs of the future development of the asset classes. In many applications, the parameters are calibrated so that the future scenarios are consistent with the past. This might be acceptable for long term strategic planning. For tactical short term planning, i.e. for the question of how to construct an asset allocation mix relative to an asset allocation benchmark, such approaches are, however, inappropriate. The user wishes to express views on the future which deviate from the past. It is then important that the decision-maker can express the market expectations in a way that he or she finds convenient and that these expectations are converted to model input in a consistent manner.

This is the basic philosophy of the scenario generation method proposed in (Høyland and Wallace, 2001). The user specifies his or her market expectations in terms of marginal distributions for each asset class, in addition to correlations among the different asset classes, and possibly other distributional properties. The stochastic, possibly multi-period, asset allocation model requires discrete outcomes for the uncertain variables. To generate these outcomes a least squares model is applied. The idea is to minimize the distance between some specified properties of the generated outcomes and their target values (either specified directly or derived from the marginal distributions, which may themselves be calculated from data or specified explicitly).

In the general form of the algorithm presented by Høyland and Wallace, outcomes of all the random variables (assets) are generated simultaneously. Such an approach becomes slow when the number of random variables increases. In this paper we generate one marginal distribution at a time and create the joint distribution by putting the marginal distributions together in the following way: All marginal distributions are generated with the same number of realizations, and the probability of the i'th realization is the same for each marginal distribution. The i'th scenario, that is, the i'th realization of the joint distribution, is then created by using the i'th realization from each marginal distribution,

and given the corresponding probability. We then apply various transformations in an iterative loop to reach the target moments and correlations.

The presented algorithm is inspired by the work of (Fleishman, 1978), (Vale, 1983) and (Lurie and Goldberg, 1998). Fleishman presents a cubic transformation that transforms a sample from a univariate normal distribution to a distribution satisfying some specified values for the first four moments. Vale and Maurelli address the multivariate case and analyse how the correlations are changed when the cubic transformation is applied. The algorithm assumes that we start out with multivariate normals. The initial correlations are adjusted so that the correlations after the cubic transformation are the desired ones. The algorithm is only approximate with no guarantee about the level of the error.

Lurie and Goldberg outlined an algorithm that is of similar type as ours. They also generate marginal distributions independently and transform them in an iterative procedure. There are, however, two major differences between the two algorithms. One is in the way they handle the (inevitable) change of distribution during the transition to the multivariate distribution — while they modify the correlation matrix in order to end up with the right distribution, we modify the starting moments. The other major difference is that they start out with parametric marginal distributions whereas we start out with the marginal moments. We believe that specifying marginal moments is a more flexible approach and we certainly could also derive the marginal moments (up to the desired number) from the parametric distribution and apply our approach.

The rest of the paper is organized as follows: In Section 2 we present the algorithm. Numerical results are presented in Section 3, while possible future research areas are discussed in Section 4.

2. The algorithm

In Høyland and Wallace's method, a scenario tree can in principle be constructed to fit all distributional properties that can be formulated as functions of probabilities and outcomes. In this section, we will assume that the properties are the first four marginal moments and the correlations. This assumption is consistent with many other studies, as well as our own empirical analysis in (Høyland and Wallace, 2001) of what were the important properties in the given case study. The presented methodology can, in fact, treat more than four marginal moments and correlations. We could specify even higher moments, but the method is more restrictive than our original approach, which also allowed for such as extreme values.

The general idea of the algorithm is as follows: Generate n discrete univariate random variables, each satisfying a specification for the first four moments. Transform them so that the resulting random vector is consistent with a given correlation matrix. The transformation will distort the marginal moments of higher than second order. Hence, we need to start out with a different set of higher moments, so that we end up with the right ones.

The procedure would lead to the exact desired values for the correlations and the marginal moments if the generated univariate random variables were independent. This

is, however, true only when the number of outcomes goes to infinity and all the scenarios are equally probable.[1] With a limited number of outcomes, and possibly distinct probabilities, the marginal moments and the correlations will therefore not fully match the specifications. To be able to secure that the error is within a pre-specified range, we have developed an iterative algorithm, which is an extension of the core algorithm.

Section 2.1 discusses the assumptions we have on the correlation matrix, Section 2.2 introduces necessary notation, Section 2.3 explains the key transformations used in the algorithm, Section 2.4 describes the core module of the algorithm, while Section 2.5 explains the iterative procedure.

2.1. Assumption on the correlation matrix

There are two assumptions on the specified correlation matrix R. The first is a general assumption that R is a possible correlation matrix, i.e. that it is a symmetric positive semi-definite matrix with 1's on the main diagonal. While implementing the algorithm there is no need to check positive semi-definiteness directly, as we do a Cholesky decomposition of the matrix R at the very start. If R is not positive semi-definite, the Cholesky decomposition will fail.

Note that having an R that is not positive semi-definite means having some internal inconsistency in the data, so we should re-run our analysis. As an alternative, there exist several algorithms that find a possible correlation matrix that is, in some sense, closest to the specified matrix R. One such algorithm can be found in (Higham, 2000). Another approach is used in (Lurie and Goldberg, 1998), where a QP model is formulated to solve the problem. The latter approach has an advantage of being very flexible and allowing, for example, for specifying weights that express how much we believe in every single correlation. We can also use bounds to restrict the possible values.

The other assumption is that the random variables are not collinear, so that R is a non-singular — hence positive definite — matrix. For checking this property we can again use the Cholesky decomposition, because the resulting lower-triangular matrix L will have zero(s) on its main diagonal in a case of collinearity.

This is not a serious restriction, since having collinearity means that at least one of the variables can be computed from the others after the generation. We can thus decrease the dimension of the problem.

2.2. Notation

To formulate the model we introduce the following notation. Note that vectors are columns by default.

n	number of random variables
s	number of scenarios

[1] With unequal probabilities, we can expect the extreme cases to accumulate in the scenarios with the smallest probabilities, while the most probable scenarios would end up with outcomes close to their respective means. That would result in dependencies.

$\tilde{X}$	general n-dimensional random variable $\rightarrow$ $\tilde{X} = (\tilde{X}_1, \tilde{X}_2, \ldots, \tilde{X}_n)$ $\rightarrow$ every moment of $\tilde{X}$ is a vector of size n $\rightarrow$ the correlation matrix of $\tilde{X}$ is a matrix of size $n \times n$
$\mathbb{X}$	matrix of s scenario outcomes — $\mathbb{X}$ has dimension $n \times s$
$\mathbb{X}_i$	row vector of outcomes of the i'th random variable — dim. s
$\mathbb{P}$	row vector of scenario probabilities — given by the user
$\tilde{\mathcal{X}}$	discrete n-dimensional random variable given by $\mathbb{X}$ and $\mathbb{P}$
$\mathbb{E}\big[\tilde{X}\big]$ or $\mathbb{E}\left[\tilde{\mathcal{X}}\right]$	vector of means of a random variable (general or discrete)
$\mathcal{RV}(mom; corr)$	the set of all random variables with moments $mom = mom_1$, $mom_2, \ldots, mom_4$, and a correlation matrix $corr$, where every mom_i is a vector of size n and $corr$ is a matrix of size $n \times n$
$TARMOM$	matrix of target moments $(4 \times n)$
R	target correlation matrix $(n \times n)$

Note the use of calligraphic letters — $\tilde{\mathcal{X}}$ — to represent a discrete random variable given by $\mathbb{X}$ and $\mathbb{P}$. Hence, we use $\tilde{X}$ when we refer to a general distribution (continuous or discrete), while $\tilde{\mathcal{X}}$ is always discrete, and always part of a construction eventually leading to the scenario tree we are about to make. Since the scenario probabilities $\mathbb{P}$ are given, generating the discrete random variable $\tilde{\mathcal{X}}$ is equivalent to generating a matrix of its outcomes $\mathbb{X}$. The two terms are thus very closely connected, even if $\tilde{\mathcal{X}}$ is a random variable, while $\mathbb{X}$ is a matrix.

Our goal is to generate scenarios with outcomes $\mathbb{Z}$, such that the discrete random variable $\tilde{\mathcal{Z}}$ defined by those outcomes and the scenario probabilities $\mathbb{P}$ has moments equal to $TARMOM$ and a correlation matrix equal to R. In our notation we want to generate a discrete random variable $\tilde{\mathcal{Z}}$ such that $\tilde{\mathcal{Z}} \in \mathcal{RV}(TARMOM; R)$.

2.3. Key transformations

The core module, which will be presented in the next section, has two key transformations. One is a cubic transformation used for generating univariate distributions with specified moments. The other is a matrix transformation used to transform a multivariate distribution to obtain a given correlation matrix.

2.3.1. Cubic transformation

This transformation comes from (Fleishman, 1978), where a method to generate a univariate non-normal random variable $\tilde{Y}_i$ with given first four moments is introduced.[2] It takes a $\mathcal{N}(0,1)$ variable $\tilde{X}_i$ and uses a cubic transformation

$$\tilde{Y}_i = a + b\tilde{X}_i + c\tilde{X}_i^2 + d\tilde{X}_i^3$$

[2] The index i is obsolete in this section, but we use it for consistence with the rest of the paper.

to obtain $\tilde{Y}_i$ with the target moments. Parameters a, b, c and d are found by solving a system of non-linear equations. Those equations utilize normality of the variable $\tilde{X}_i$.

The problem of this approach is that $\tilde{X}_i$ must have the first 12 moments equal to those of $\mathcal{N}(0,1)$ in order to get exactly the target moments of $\tilde{Y}_i$. Since this is hard to achieve, either by sampling or discretization, the results with those formulas are only approximate. We have thus dropped the assumption of normality and derived formulas that work with an arbitrary random variable $\tilde{X}_i$ — see Appendix A. Parameters a, b, c and d are now given by a system of four implicit equations.

We have used a non-linear mathematical-programming (least-squares) model to solve the system. In this model we have a, b, c and d as decision variables, and we express the moments of $\tilde{\mathcal{Y}}_i$ as functions of these variables and the first 12 moments of $\tilde{\mathcal{X}}$.[3] We then minimize the distance between those moments and their target values. We do not need to assume that the system has a solution — if the solution does not exist, the model gives us the $\tilde{\mathcal{Y}}_i$ with the closest possible moments.

Our method for generating a discrete approximation $\tilde{\mathcal{Y}}_i$ of $\tilde{Y}_i$ is thus as follows:

- take some discrete r.v. $\tilde{\mathcal{X}}_i$ with the same number of outcomes as $\tilde{\mathcal{Y}}_i$
- calculate the first 12 moments from $\tilde{\mathcal{X}}_i$
- compute the parameters a, b, c, d
- compute the outcomes $\mathbb{Y}_i$ of $\tilde{\mathcal{Y}}_i$ as $\mathbb{Y}_i = a + b\mathbb{X}_i + c\mathbb{X}_i^2 + d\mathbb{X}_i^3$

2.3.2. Matrix transformation

Our other main tool in the algorithm is a matrix transformation of a random variable $\tilde{X}$

$$\tilde{Y} = L\,\tilde{X}$$

where L is a lower-triangular matrix. The matrix L always comes from a Cholesky decomposition of the correlation matrix R, so we have $L\,L^T = R$.[4]

From theory we know that if $\tilde{X}$ is an n-dimensional $\mathcal{N}(0,1)$ random variable with correlation matrix I (and therefore with $\tilde{X}_i$ mutually independent), then the $\tilde{Y} = L\,\tilde{X}$ is an n-dimensional $\mathcal{N}(0,1)$ random variable with correlation matrix $R = L\,L^T$.

Since we do not have normal variables, we need a more general result. To make the formulas as simple as possible, we restrict ourselves to the case of zero means and variances equal to 1. In the beginning of Section 2.4 we show how to deal with this restriction. Note that $E[\tilde{X}] = 0$ leads to $mom_i = E[\tilde{X}^i]$. We will thus use the two interchangeably for the rest of the section.

In Appendix B, we show that, in that case, $\tilde{Y}(= L\,\tilde{X})$ is an n-dimensional random variable with zero means, variances equal to 1, and correlation matrix $R = L\,L^T$. In addition, the higher moments of $\tilde{Y}$ are as follows:

[3] Since we have switched from general random variables to a discrete scenario model, we have to switch the notation from $\tilde{Y}$ to $\tilde{\mathcal{Y}}$.

[4] Note that L always exists, since we assume R to be positive semi-definite.

$$\begin{aligned}
\mathbb{E}\left[\tilde{Y}_i^3\right] &= \sum_{j=1}^{i} L_{ij}^3\, \mathbb{E}\left[\tilde{X}_j^3\right] \\
\mathbb{E}\left[\tilde{Y}_i^4\right] - 3 &= \sum_{j=1}^{i} L_{ij}^4 \left(\mathbb{E}\left[\tilde{X}_j^4\right] - 3\right)
\end{aligned}$$

We will need also the opposite direction of the transformation:

$$\tilde{X} = L^{-1}\,\tilde{Y}.$$

Since L^{-1} is a triangular matrix, it is easy to invert the formulas:

$$\begin{aligned}
\mathbb{E}\left[\tilde{X}_i^3\right] &= \frac{1}{L_{ii}^3}\left(\mathbb{E}\left[\tilde{Y}_i^3\right] - \sum_{j=1}^{i-1} L_{ij}^3\, \mathbb{E}\left[\tilde{X}_j^3\right]\right) \\
\mathbb{E}\left[\tilde{X}_i^4\right] - 3 &= \frac{1}{L_{ii}^4}\left[\mathbb{E}\left[\tilde{Y}_i^4\right] - 3 - \sum_{j=1}^{i-1} L_{ij}^4 \left(\mathbb{E}\left[\tilde{X}_j^4\right] - 3\right)\right]
\end{aligned}$$

We divide only by the diagonal elements L_{ii} in these formulas. We can do it since L_{ii} are positive due to regularity (positive definiteness) of R.

2.4. The core algorithm

This section presents the core algorithm. It runs as follows: Find the target marginal moments from stochastic processes, from statistics or by specifying them directly. Generate n discrete random variables with these moments. Create the multivariate random variable by combining the univariate variables, as explained in the Introduction. Transform this variable so that it has the desired correlations and marginal moments. If the random variables $\tilde{\mathcal{X}}_i$ were independent, we would end up with $\tilde{\mathcal{Y}}$ having exactly the desired properties.

To facilitate the reading, we have divided the algorithm in two parts. In the *input phase* we read the target properties specified by the user and transform them to a form needed by the algorithm. In the *output phase* we generate the distributions and transform them to the original properties.

2.4.1. The input phase

In this phase we work only with the target moments and correlations, we do not yet have any outcomes. This means that all operations are fast and independent of the number of scenarios s.

Our goal is to generate a discrete approximation $\tilde{\mathcal{Z}}$ of an n-dimensional random variable $\tilde{Z}$ with moments $TARMOM$ and correlation matrix R. Since the matrix transformation needs zero means and variances equal to 1, we have to change the targets to

match this requirement. Instead of $\tilde{\mathcal{Z}}$ we will thus generate random variables $\tilde{\mathcal{Y}}$ with moments MOM (and correlation matrix R), such that $MOM_1 = 0$, and $MOM_2 = 1$. $\tilde{\mathcal{Z}}$ is then computed at the very end of the algorithm as

$$\tilde{\mathcal{Z}} = \alpha\tilde{\mathcal{Y}} + \beta.$$

It is easily shown that the values leading to the correct $\tilde{\mathcal{Z}}$ are:

$$\alpha = TARMOM_2^{1/2} \qquad MOM_3 = \frac{TARMOM_3}{\alpha^3}$$
$$\beta = TARMOM_1 \qquad MOM_4 = \frac{TARMOM_4}{\alpha^4}$$

The final step in the input phase is to derive moments of independent univariate random variables $\tilde{X}_i$ such that $\tilde{Y} = L\,\tilde{X}$ will have the target moments and correlations. To do this we need to find the Cholesky-decomposition matrix L, i.e. a lower-triangular matrix L so that $R = L\,L^T$.

The input phase then contains the following steps:

1. Specify the target moments $TARMOM$ and target correlation matrix R of $\tilde{Z}$ (and $\tilde{\mathcal{Z}}$)
2. Find the normalized moments MOM for $\tilde{\mathcal{Y}}$
3. Compute L and find the transformed moments $TRSFMOM$ for $\tilde{\mathcal{X}}$ — see Section 2.3.2

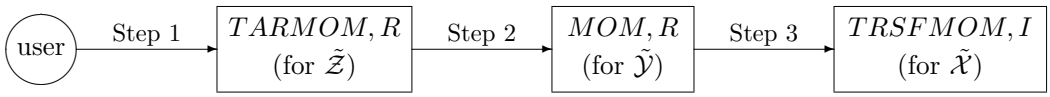

Figure 1: Input Phase

2.4.2. The output phase

In this phase we start by generating the outcomes for the independent random variables. Next, we transform them to get the intermediate-target moments and target correlations, and finally obtain the moments specified by the user. Since the last transformation is a linear one, it will not change the correlations. All the transformations in this phase are with the outcomes, so the computing time needed for this phase is longer and increases with the number of scenarios.

We start by generating n discrete univariate random variables $\tilde{\mathcal{X}}_i$. This is a well-known problem and there are several possible ways to do it. We have used a method

from (Fleishman, 1978), in a way described in Section 2.3.1. The method starts by sampling from $\mathcal{N}(0,1)$ and afterwards uses the cubic transformation to get the desired moments. For the starting $\mathcal{N}(0,1)$ sample we use a random-number generator. An alternative would be to use a discretization of the distribution, or some other method for discretizing $\mathcal{N}(0,1)$, for example the method described in (Høyland and Wallace, 2001).

Once we have generated the outcomes $\mathbb{X}_i$ for the random variables $\tilde{\mathcal{X}}_i$, we can proceed with the transformations. First $\mathbb{Y} = L\,\mathbb{X}$ to get the target correlations and then $\mathbb{Z} = \alpha\mathbb{Y} + \beta$ to get the user-specified moments.[5]

The output phase of the core algorithm consists of the following steps:

4. Generate outcomes $\mathbb{X}_i$ of 1-dimensional variables $\tilde{\mathcal{X}}_i$ (independently for $i = 1 \ldots n$)

5. Transform $\tilde{\mathcal{X}}$ to the target correlations: $\mathbb{Y} = L\,\mathbb{X} \;\rightarrow \tilde{\mathcal{Y}} \in \mathcal{RV}(MOM; R)$

6. Transform $\tilde{\mathcal{Z}}$ to the original moments: $\mathbb{Z} = \alpha\mathbb{Y} + \beta \;\rightarrow \tilde{\mathcal{Z}} \in \mathcal{RV}(TARMOM; R)$

Figure 2: Output Phase

2.5. The modified algorithm

We know that the core algorithm gives us the exact results only if the random variables $\tilde{\mathcal{X}}_i$ are independent. Since we generate each set of outcomes $\mathbb{X}_i$ separately, and have a limited number of outcomes, the resulting co-moments will most certainly differ from zero and the results of the core algorithm will be only approximate.

The modified algorithm is iterative, so all results are still approximate, but with a pre-specified maximal error. Hence, we can control the quality of the results. We will again use the matrix transformation $\mathbb{Y} = L\,\mathbb{X}$, this time both forward and backward, as mentioned in 2.3.2. Recall that this transform allows us to obtain a desired correlation matrix, but it changes the marginal moments while doing so.

In Section 2.3.1 we showed that the cubic transformation allows us to transform a general univariate random variable $\tilde{X}_i$ to a variable with some target moments. This transformation changes the correlations, but if the starting $\tilde{\mathcal{X}}_i$'s have moments close to their targets, the changes in the correlations are expected to be small.

The idea of the new algorithm is thus to introduce iterative loops in the core algorithm, namely in Steps 4 and 5, in the following way:

[5] Note that as we have stopped to speak about distributions and started to speak about outcomes, we have to change the notation from $\tilde{\mathcal{X}}$ to $\mathbb{X}$.

The purpose of Step 4 is to generate the independent random variables $\tilde{\mathcal{X}}_i$. Since independence is very hard to achieve, we change our target to generating uncorrelated r.v. $\tilde{\mathcal{X}}_i$, i.e. we seek to get $\tilde{\mathcal{X}} \in \mathcal{RV}(TRSFMOM; I)$. We use an iterative approach to achieve those properties.

Since we do not control the higher co-moments, they will most likely not be zero, and the $\tilde{\mathcal{Y}}$ obtained in Step 5 will not have the target properties. We thus need another loop there to end up with the desired $\tilde{\mathcal{Y}}$.

The iterative versions of Steps 4 and 5 are:[6]

Step 4

4.*i.* Generate n univariate random variables with moments $TRSFMOM$ (independently)
→ we get $\tilde{\mathcal{X}}$ with correlation matrix R_1 close to I due to the independent generation

4.*ii.* `let` $p = 1$ and $\tilde{\mathcal{X}}_1 = \tilde{\mathcal{X}}$

4.*iii.* `while` $dist(R_p; I) > \varepsilon_x$ `do`

4.*iv.* `do` Cholesky decomposition: $R_p = L_p\, L_p^T$

4.*v.* `do` backward transform $\mathbb{X}_p^* = L^{-1}\, \mathbb{X}_p \;\rightarrow \tilde{\mathcal{X}}^*$ has zero correlations, wrong moments

4.*vi.* `do` cubic transform of $\tilde{\mathcal{X}}_p^*$ with $TRSFMOM$ as the target moments;
store results as $\tilde{\mathcal{X}}_{p+1}$ → right moments, wrong correlations

4.*vii.* compute correlation matrix R_{p+1}

4.*viii.* `let` $p = p + 1$

4.*ix.* `let` $\tilde{\mathcal{X}} = \tilde{\mathcal{X}}_p \rightarrow \tilde{\mathcal{X}} \in \mathcal{RV}(TRSFMOM; R_p)$ with correlation error $dist(R_p; I) \leq \varepsilon_x$

A root-mean-squared-error is used as the measure $dist()$ in 4.*iii*, see Section 3 for an exact definition. The same distance is used also in Step 5 below (in steps 5.*iii* and 5.*ix*). Since $\tilde{\mathcal{X}}$ is not a final output, the maximum error ε_x in Step 4 is typically higher than the corresponding ε_y in Step 5.

There are two possible outcomes from Step 4: $\tilde{\mathcal{X}}_p$ corresponding to random variables with right moments and slightly off correlations, and $\tilde{\mathcal{X}}_{p-1}^*$ corresponding to random variables with slightly off moments and zero correlations. We start with the latter in Step 5, and denote it $\tilde{\mathcal{X}}^*$.

Step 5

5.*i.* $\tilde{\mathcal{Y}}_1 = L\, \tilde{\mathcal{X}}^* \;\rightarrow$ both moments and correlations are incorrect
(due to higher co-moments different from zero)

[6] As we have not found any better notation, in the rest of this section lower index p denotes an iteration counter, not a matrix column.

5.*ii.* `let` $p = 1$ and let R_1 be the correlation matrix of $\tilde{\mathcal{Y}}_1$

5.*iii.* `while` $dist(R_p; R) > \varepsilon_y$ `do`

5.*iv.* `do` Cholesky decomposition: $R_p = L_p\, L_p^T$

5.*v.* `do` backward transform $\mathbb{Y}_p^* = L_p^{-1}\, \mathbb{Y}_p$
$\rightarrow \tilde{\mathcal{Y}}_p$ has zero correlations, incorrect moments

5.*vi.* `do` forward transform $\mathbb{Y}_p^{**} = L\, \mathbb{Y}_p^*$
$\rightarrow \tilde{\mathcal{Y}}_p$ has correct correlations (R), incorrect moments

5.*vii.* `do` cubic transform of $\tilde{\mathcal{Y}}_p^{**}$ with MOM as the target moments;
store results as $\tilde{\mathcal{Y}}_{p+1} \rightarrow \tilde{\mathcal{Y}}_{p+1} \in \mathcal{RV}(MOM; R_{p+1})$

5.*viii.* `let` $p = p + 1$

5.*ix.* `let` $\tilde{\mathcal{Y}} = \tilde{\mathcal{Y}}_p \;\rightarrow\; \tilde{\mathcal{Y}} \in (MOM; R_p)$ with correlation error $dist(R_p; R) \leq \varepsilon_y$

Note that we can again choose two different outcomes from Step 5 (and thus from the whole algorithm). After Step 5.*ix*, $\tilde{\mathcal{Y}}$ has the right moments and (slightly) off correlation. If we prefer exact correlations, we can either use the last $\tilde{\mathcal{Y}}_p^{**}$, or repeat Steps 5.*iv* – 5.*vi* just before we go to Step 5.*ix*.

Note also that Steps 5.*v* and 5.*vi* are written as individual steps for the sake of clarity. Since we have always $s > n$ (usually $s \gg n$), it is much more efficient to join the two steps and calculate $\mathbb{Y}_p^{**} = \left(L \times L_p^{-1}\right)\, \mathbb{Y}_p$ directly.

2.5.1. Convergence

The issue of convergence to the target moments and correlations is difficult. First, we know that the method cannot converge in general, since it is possible to specify combinations of moments that cannot exist. In addition, we need to have enough scenarios, where the minimal number of scenarios depends not only on the number of random variables n, but also on the structure of the problem. For example, random variables with distribution close to the normal, and with small correlations, typically need fewer scenarios than random variables with fat tails and high correlations.

We have not succeeded in producing a convergence proof, but more than two years of active use in Gjensidige NOR has so far never left us with an unsolvable case. The algorithm is stopped whenever a prespecified number of iterations has been performed, or the convergence criteria are satisfied. Whatever the reason, it is straightforward to test if the resulting scenario tree has the required properties (and also if it is arbitrage-free). Hence, when a tree is used in an optimization model, it always has the required properties.

Therefore, the concern is not that we risk using a tree with bad properties, but rather that we are left with no tree at all because the algorithm either converges to the wrong solution or diverges.

If the algorithm does not converge to the right solution, our first approach is always to try to rerun the algorithm a few times, in reality, trying to start the whole process

with a different set of independent random variables in Step 4.*i*. If that does not work, we try to increase the number of scenarios. Unless we know for sure that the specified set of properties exist (for example because they are calculated from a data set or from given distributions) we next try to find out if we may have defined random variables that do not exist. By these three means, we have always found a solution.

It is worth noting that when the method is used on a continuous basis, the user learns how many scenarios are needed for his problem. Also, he will know whether or not he has specified the properties in such a way that the implied distribution exists. This will limit the possible actions. Our experience is that it is enough to first check for actual errors (bugs) in the specifications, and then rerun the algorithm a few times from different starting points.

A numerical test of convergence is in Section 3.1.

3. Numerical results

This section presents the times needed to generate scenario trees with different numbers of random variables and scenarios.

To see how the generation time depends on the number of random variables and the number of scenarios, we have generated trees with 4, 8, 12 and 20 random variables and 40, 100, 200 and 1000 scenarios. Except the case of four random variables, we have used actual data from Gjensidige NOR as input. In addition, we have used two different data sets in the cases of 12 and 20 random variables to improve the estimates. Since we did not have any distinct case with only four random variables, we have used the first four variables from the first 12-variable case. A more detailed description of the input data can be found in `http://home.himolde.no/~wallace/reports.htm`.

The algorithm is implemented in `AMPL` with `MINOS 5.5` as a solver. The test was done on a Pentium III 500 MHz machine with 256 MB memory, running Windows NT 4.0. As a distance used for evaluating the quality of the distributions in Steps 4 and 5 of the algorithm, we have used a root-mean-squared-error defined as

$$RMSE = \sqrt{\frac{1}{N_{el}} \sum_k (value_k - TARGET_k)^2}$$

where N_{el} is the number of elements in the sum. The distance was evaluated separately for moments ($N_{el} = 4n$) and for correlations ($N_{el} = \frac{n(n-1)}{2}$).

The stopping-values were $\varepsilon_x = 0.1$ ($\varepsilon_x = 0.2$ in the case of 20 r.v.) and $\varepsilon_y = 0.01$. Note that the distances are evaluated inside the algorithm, where all the data are scaled to variance equal to 1, so they are scale-independent. Using the formulas in Section 2.4.1, one can show that the maximum error of the i'th moment in the final output is $TARMOM_i^i \cdot \varepsilon_y$.

Results of the tests are in Table 1. As we see, we can find trees with as much as 1000 scenarios in less than a minute. It is worth observing that as the number of scenarios increases, the computing time decreases. This strange behavior is caused by a better convergence for larger trees, i.e. the fact that the number of iterations decreases with

Table 1: Running times for different numbers of random variables and scenarios. Times are given in format `mm:ss`.

	Number of scenarios			
r.v.	40	100	200	1000
4	00:01	00:01	00:01	00:05
8	00:04	00:04	00:03	00:10
12	00:09	00:07	00:06	00:16
20	01:05	00:44	00:31	00:48

the size of a tree: with 40 scenarios we need typically 1–2 iterations in Step 4 of the algorithm (generation of $\tilde{\mathcal{X}}$) and 2–3 iterations in Step 5 (generation of $\tilde{\mathcal{Y}}$), whereas with 1000 scenarios we usually need only one iteration in each part. Since we cannot have less than one iteration, there will be no more improvement in the convergence for trees with more than 1000 scenarios. We can thus expect approximately linear time-dependency for very large trees.

It may be of interest to compare these times with what we observe by using the method in (Høyland and Wallace, 2001). Table 2 shows the results for 1000 scenarios. As we see, the savings are substantial.

Table 2: Running times for creating 1000 scenarios. In the first column are times for the algorithm from (Høyland and Wallace, 2001), in the second column times for the new algorithm. The last column presents the speed-up. Times are given in format `[h:]mm:ss`.

r.v.	old alg.	new alg.	speed-up
4	00:35	00:05	7.5×
8	08:39	00:10	53×
12	17:54	00:16	66×
20	1:34:46	00:48	119×

3.1. Convergence test

As mentioned in Section 2.5.1, we do not have a convergence proof for the iterative algorithm presented in Section 2.5. We have thus tested the convergence numerically. The iterative procedure is used in steps 4 and 5 of the algorithm. Step 4 is, however, used only to obtain a reasonable starting point for Step 5, so the convergence is not as important there. Hence, we only present convergence tests for Step 5.

For the test, we have chosen the smallest and the biggest cases from the data used in the main test, namely 40 scenarios for 4 random variables and 1000 scenarios for 20

random variables. For both sets, we have run the algorithm 25 times, each time with a different initial set of scenarios.

We have measured an error in moments after the matrix transformation and an error in correlations after the cubic transformation. The other two combinations are not interesting, since both transformations set "their" errors to zero. Hence, the two errors are equal to the total errors after the given transformation.[7]

Results of the test are in Figure 3. Every iteration consists of the matrix transformation (Steps 5.*iv*...5.*vi*) and the cubic transformation (Step 5.*vii*). We see that both the errors are monotonously decreasing. Hence, also the total error at the end of an iteration is monotonously decreasing. This is actually true not only for the average values depicted in the graphs, but for all the 25 runs, in both test cases.

4. Future work

The most important subject for future research is the convergence of the algorithm. Since we do not have a convergence proof, we should try to obtain a better understanding of the convergence. In particular, we should try to identify statistical properties that would guarantee/prevent convergence. Another interesting problem is the minimum number of scenarios needed to achieve given precision for a given set of properties.

In many cases we are interested in multi-period scenario trees. It is a future challenge to see how the presented method can be used as a building block for a larger tree, particularly how to preserve time series properties.

Even though we have achieved a substantial computational speed, the tested implementation is still far from optimal. In the current `AMPL` implementation, the algorithm spends most of its time with the Cholesky transformation and the communication with the solver before and after the cubic transform.

Both these critical times can be eliminated or decreased by implementing the algorithm in a compiled form. A `C++` implementation is currently being developed at `SINTEF`[8] research institute, and according to the first tests is more than 10 times faster than our `AMPL` code. This implementation is part of OMEGA-IST-1999-12088, an EU financed project on electricity production i deregulated markets.

The presented algorithm is well suited for parallel implementation, because most of its parts can be processed independently for each random variable. The only step of the algorithm that uses considerable amount of time and cannot be parallelized in this simple way is the Cholesky decomposition of the correlation matrix, but parallel codes for the Cholesky decomposition can also be found.

[7] We define the total error as a sum of the errors in moments and correlations. After every transformation one of the errors is zero, hence the total is equal to the other one.

[8] The Foundation for Scientific and Industrial Research at the Norwegian Institute of Technology

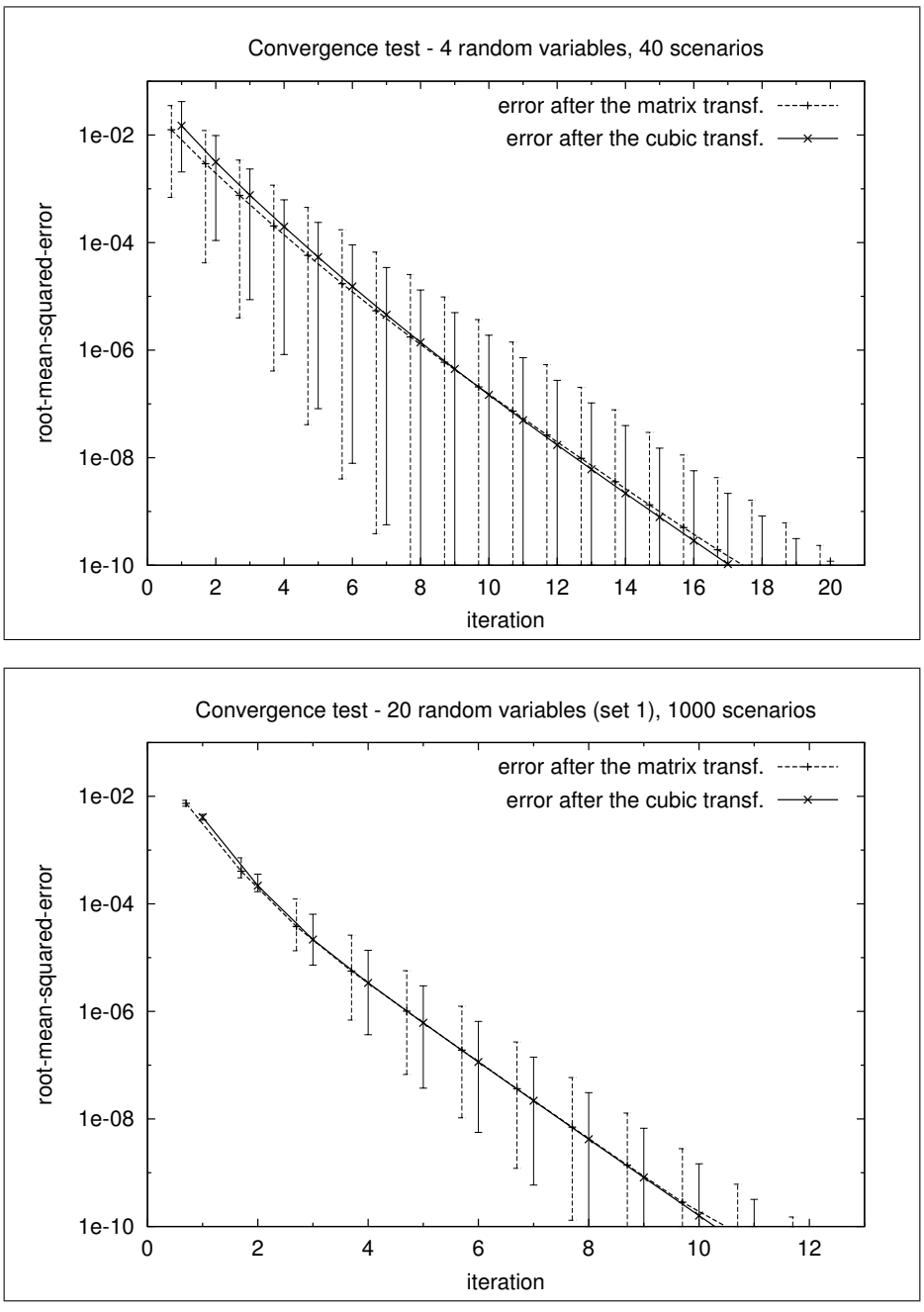

Figure 3: Convergence of the iterative algorithm. In both cases, the algorithm was run 25 times. Lines represent average errors after every iteration, bars represents the best and the worst cases. The dashed lines represents errors in moments after the matrix transformation, the solid line errors in correlations after the cubic transformation. The root-mean-squared-error is defined in Section 3.

5. Conclusions

We have presented an algorithm to generate scenarios for multivariate random variables. The purpose of the algorithm is to speed up an existing scenario generation algorithm, which constructs multi-dimensional scenario trees with specified moments and correlations.

The original algorithm constructs the multidimensional scenario tree by solving a single, potentially very large, least squares problem. The main idea of the new algorithm is to decompose the least squares problem so that each marginal distribution is constructed separately. To combine the different marginal distributions so that the joint distribution satisfies the specified correlations, we apply a Cholesky decomposition and a cubic transformation in an iterative procedure.

Even if we cannot guarantee convergence of this procedure, our experience shows that it does converge if the specifications are possible and there are enough scenarios. In addition, a potential divergence or convergence to the wrong solution is easy to detect. Hence, we never end up using an incorrect tree in the optimization procedure.

Testing shows that our algorithm is reasonably fast. We can find trees with 1000 scenarios representing 20 random variables in less than one minute.

Acknowledgments

We would like to thank Erik Kole from the Maastricht University for pointing out an error in an earlier version of the paper, and Matthias Nowak from `SINTEF`, Trondheim, for suggesting some important changes in notation and structure of the paper. Furthermore, we would like to thank our colleague Halvard Arntzen for helping us with cleaning up the terminology. We are also in debt to anonymous referees. Much of this work was done while Stein W. Wallace visited the Centre for Advanced Study at the Academy of Science and Letters in Oslo.

References

Cariño, D. R. and Ziemba, W. T. (1998). Formulation of the Russell-Yasuda Kasai financial planning model. *Operations Research*, 46(4):443–449.

Consigli, G. and Dempster, M. A. H. (1998). Dynamic stochastic programming for asset-liability management. *Annals of Operations Research*, 81:131–162.

Dert, C. (1995). *Asset Liability Management for Pension Funds, A Multistage Chance Constrained Programming Approach*. PhD thesis, Erasmus University, Rotterdam, The Netherlands.

Fleishman, A. I. (1978). A method for simulating nonnormal distributions. *Psychometrika*, 43:521–532.

Higham, N. J. (2000). Computing the nearest correlation matrix—A problem from finance. Numerical Analysis Report No. 369, Manchester Centre for Computational Mathematics, Manchester, England.

Høyland, K. and Wallace, S. W. (2001). Generating scenario trees for multistage decision problems. *Management Science*, 47(2):295–307.

Lurie, P. M. and Goldberg, M. S. (1998). An approximate method for sampling correlated random variables from partially-specified distributions. *Management Science*, 44(2):203–218.

Mulvey, J. M. (1996). Generating scenarios for the Towers Perrin investment system. *Interfaces*, 26:1–13.

Vale, C. David & Maurelli, V. A. (1983). Simulating multivariate nonnormal distributions. *Psychometrika*, 48(3):465–471.

Appendix

A. Cubic transformation $\tilde{Y} = a + b\tilde{X} + c\tilde{X}^2 + d\tilde{X}^3$

The purpose of this transformation is to produce a univariate random variable $\tilde{Y}_i$ with specified first four moments $\mathbb{E}\big[\tilde{Y}_i^k\big], k = 1, \ldots, 4$, given the random variable $\tilde{X}_i$ with known first 12 moments $\mathbb{E}\big[\tilde{X}_i^k\big], k = 1, \ldots, 12$.

The problem is to find the transform parameters a_i, b_i, c_i and d_i. For this we have to express $\mathbb{E}\big[\tilde{Y}_i^k\big]$ as functions of $\mathbb{E}\big[\tilde{X}_i^k\big]$. The formulas can be stated either for univariate random variables $\tilde{Y}_i$, or as a vector equations for the random vector $\tilde{Y}$. The only difference is the presence/absence of index i in all elements. To make the formulas easier to read, we use the vector form.

$$
\begin{aligned}
\mathbb{E}\big[\tilde{Y}\big] &= a + b\mathbb{E}\big[\tilde{X}\big] + c\mathbb{E}\big[\tilde{X}^2\big] + d\mathbb{E}\big[\tilde{X}^3\big] \\
\mathbb{E}\big[\tilde{Y}^2\big] &= d^2\mathbb{E}\big[\tilde{X}^6\big] + 2cd\mathbb{E}\big[\tilde{X}^5\big] + (2bd + c^2)\mathbb{E}\big[\tilde{X}^4\big] + (2ad + 2bc)\mathbb{E}\big[\tilde{X}^3\big] \\
&+ (2ac + b^2)\mathbb{E}\big[\tilde{X}^2\big] + 2ab\mathbb{E}\big[\tilde{X}\big] + a^2 \\
\mathbb{E}\big[\tilde{Y}^3\big] &= d^3\mathbb{E}\big[\tilde{X}^9\big] + 3cd^2\mathbb{E}\big[\tilde{X}^8\big] + (3bd^2 + 3c^2d)\mathbb{E}\big[\tilde{X}^7\big] + (3ad^2 + 6bcd + c^3)\mathbb{E}\big[\tilde{X}^6\big] \\
&+ (6acd + 3b^2d + 3bc^2)\mathbb{E}\big[\tilde{X}^5\big] + (a(6bd + 3c^2) + 3b^2c)\mathbb{E}\big[\tilde{X}^4\big] \\
&+ (3a^2d + 6abc + b^3)\mathbb{E}\big[\tilde{X}^3\big] + (3a^2c + 3ab^2)\mathbb{E}\big[\tilde{X}^2\big] + 3a^2b\mathbb{E}\big[\tilde{X}\big] + a^3 \\
\mathbb{E}\big[\tilde{Y}^4\big] &= d^4\mathbb{E}\big[\tilde{X}^{12}\big] + 4cd^3\mathbb{E}\big[\tilde{X}^{11}\big] + (4bd^3 + 6c^2d^2)\mathbb{E}\big[\tilde{X}^{10}\big] \\
&+ (4ad^3 + 12bcd^2 + 4c^3d)\mathbb{E}\big[\tilde{X}^9\big] + (12acd^2 + 6b^2d^2 + 12bc^2d + c^4)\mathbb{E}\big[\tilde{X}^8\big] \\
&+ (a(12bd^2 + 12c^2d) + 12b^2cd + 4bc^3)\mathbb{E}\big[\tilde{X}^7\big] + (6a^2d^2 + a(24bcd + 4c^3) \\
&+ 4b^3d + 6b^2c^2)\mathbb{E}\big[\tilde{X}^6\big] + (12a^2cd + a(12b^2d + 12bc^2) + 4b^3c)\mathbb{E}\big[\tilde{X}^5\big] \\
&+ (a^2(12bd + 6c^2) + 12ab^2c + b^4)\mathbb{E}\big[\tilde{X}^4\big] + (4a^3d + 12a^2bc + 4ab^3)\mathbb{E}\big[\tilde{X}^3\big] \\
&+ (4a^3c + 6a^2b^2)\mathbb{E}\big[\tilde{X}^2\big] + 4a^3b\mathbb{E}\big[\tilde{X}\big] + a^4
\end{aligned}
$$

Note that since the matrix transformation $\tilde{Y}_i = L\,\tilde{X}_i$ does not change the first two moments, we have $E[\tilde{X}_i] = 0$ and $E[\tilde{X}_i^2] = 1$. We could thus simplify the above formulas slightly by assuming that the first two moments are exactly 0 and 1. In our implementation we have, however, used the formulas in the presented form, computing the actual moments.

B. Matrix transformation $\tilde{Y} = L\,\tilde{X}$

We seek an n-dimensional random variable $\tilde{Y}$ with zero means, variances equal to 1, skewness MOM_3, kurtosis MOM_4, and a correlation matrix $R = L\,L^T$, where L is a lower-triangular matrix. To obtain $\tilde{Y}$, we start with an n-dimensional random vector $\tilde{X}$ with independent components $\tilde{X}_i$. Thereafter, $\tilde{Y}$ is computed as $\tilde{Y} = L\,\tilde{X}$. Note that the k'th moment of $\tilde{Y}$ is equal to $\mathbb{E}\big[\tilde{Y}^k\big]$ because of the zero means.

During the algorithm we use the matrix transformation in two different cases. First it is used on distributions, which are abstract objects, so we work only with their distributional properties. In the second part of the algorithm, we transform the outcomes, which are matrices of numbers. The formulas are the same in both cases, the only difference is the use of $\tilde{X}$ in the first case and $\mathbb{X}$ in the latter.

We use the distribution-notation in the rest of the appendix. The matrix transformation in a column-wise form is then:

$$\tilde{Y}_i = \Big(L\,\tilde{X}\Big)_i = \sum_{j=1}^{n} L_{ij}\,\tilde{X}_j = \sum_{j=1}^{i} L_{ij}\,\tilde{X}_j$$

where the last equality comes from the fact that L is a lower-triangular matrix and therefore $L_{ij} = 0$ for $j > i$.

Theorem – properties of $\tilde{Y} = L\,\tilde{X}$

Assume we have an n-dimensional random variable $\tilde{X}$ with the following properties:

i. $\mathbb{E}\big[\tilde{X}^k\big]$ exists for $k = 1 \ldots 4$

ii. $\mathbb{E}\big[\tilde{X}\big] = 0 \quad$ and $\quad \mathbb{E}\big[\tilde{X}^2\big] = 1$

iii. the univariate random variables $\tilde{X}_i$, $\tilde{X}_j$ are independent for $i \neq j$

Assume further that L is a lower-triangular matrix of size n such that $R = L\,L^T$, where R is a correlation matrix, i.e. R is a symmetric positive semi-definite matrix with 1's on the main diagonal.

If we then define a random variable $\tilde{Y}$ as $\tilde{Y} = L\,\tilde{X}$, it has the following properties:

iv. $\mathbb{E}\big[\tilde{Y}^k\big]$ exists for $k = 1 \ldots 4$

v. $\mathbb{E}\big[\tilde{Y}\big] = 0 \quad$ and $\quad \mathbb{E}\big[\tilde{Y}^2\big] = 1$

vi. $\tilde{Y}$ has a correlation matrix $R = L\,L^T$

vii. $$\mathbb{E}\left[\tilde{Y}_i^3\right] = \sum_{j=1}^{i} L_{ij}^3\,\mathbb{E}\left[\tilde{X}_j^3\right]$$

viii. $$\mathbb{E}\left[\tilde{Y}_i^4\right] - 3 = \sum_{j=1}^{i} L_{ij}^4\left(\mathbb{E}\left[\tilde{X}_j^4\right] - 3\right)$$

The proof is straightforward, and is left to the reader.

Consequence

Under the assumptions of the theorem, with an additional assumption that R is regular (and therefore positive-definite), we can express the moments of $\tilde{X}$ as:

ix. $$\mathbb{E}\left[\tilde{X}_i^3\right] = \frac{1}{L_{ii}^3}\left(\mathbb{E}\left[\tilde{Y}_i^3\right] - \sum_{j=1}^{i-1} L_{ij}^3\,\mathbb{E}\left[\tilde{X}_j^3\right]\right)$$

x. $$\mathbb{E}\left[\tilde{X}_i^4\right] - 3 = \frac{1}{L_{ii}^4}\left[\mathbb{E}\left[\tilde{Y}_i^4\right] - 3 - \sum_{j=1}^{i-1} L_{ij}^4\left(\mathbb{E}\left[\tilde{X}_j^4\right] - 3\right)\right]$$

Note that the set of moments $(0, 1, 0, 3)$ is an invariant of this transformation. This confirms the known theoretical result that the linear transformations preserve normality. In the context of our algorithm it means that if we generate normal variables, we can skip Step 3 of the algorithm.