Log-normal approximations in the Solvency II Standard Formula

Peder T. Haagensli
Master’s Thesis, Spring 2017
This master’s thesis is submitted under the master’s programme *Modelling and Data Analysis*, with programme option *Finance, Insurance and Risk*, at the Department of Mathematics, University of Oslo. The scope of the thesis is 30 credits.

The front page depicts a section of the root system of the exceptional Lie group $E_8$, projected into the plane. Lie groups were invented by the Norwegian mathematician Sophus Lie (1842–1899) to express symmetries in differential equations and today they play a central role in various parts of mathematics.
Abstract

On the 1st of January 2016, the Solvency II Directive regulating the European insurance industry came into force. Along with a host of reporting requirements, quantitative algorithms for calculating key quantities are provided in the Solvency II documentation. One of these key quantities is the Solvency Capital Requirement (SCR) providing an insurance company with a "soft" floor in terms of capital that it needs to hold. A vital element in calculating the SCR for an insurance company is the so-called Standard formula based on Gaussian risks which is used iteratively to aggregate the different risk elements in the business.

An alternative to the Standard formula was proposed by Bolviken and Guillen who introduced log-normal distributions to better capture the skewness often present in insurance risks. They used classical moment matching to approximate the distribution of a sum of log-normal risks. In this thesis we will consider another log-normal alternative making use of moment-generating functions. This method will then be shown to offer superior accuracy compared to the Standard formula and the method of Bolviken and Guillen.

Keywords: Cholesky decomposition, Gauss-Hermite quadrature, moment-generating function, principal components, Solvency II, spectral decomposition, Standard formula, sum of log-normals.
## Contents

1 Introduction 1

2 The Solvency II approach 2
   2.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
   2.2 The Standard Formula . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

3 A log-normal standard formula 4
   3.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
   3.2 The Fenton-Wilkinson approximation . . . . . . . . . . . . . . . . . . . . 4
   3.3 Matching moment-generating functions . . . . . . . . . . . . . . . . . . . . 5
      3.3.1 Cholesky decomposition . . . . . . . . . . . . . . . . . . . . . . . . 6
      3.3.2 Spectral decomposition . . . . . . . . . . . . . . . . . . . . . . . . . 8
      3.3.3 Dimension reduction . . . . . . . . . . . . . . . . . . . . . . . . . . 9

4 Numerical experiments 9
   4.1 Accuracy . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
   4.2 Design of the MGF method . . . . . . . . . . . . . . . . . . . . . . . . . . 10
   4.3 Computing time experiments . . . . . . . . . . . . . . . . . . . . . . . . . . 11

5 Conclusion 12

References 13

Appendices 14

A Figures - Numerical experiments 14

B PDF and MGF 18
   B.1 PDF of our log-normal model . . . . . . . . . . . . . . . . . . . . . . . . 18
   B.2 The moment-generating function of Y and Gauss-Hermite . . . . . . . 18

C The Cholesky decomposition 19
   C.1 Change of variable . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
   C.2 Applying Gauss-Hermite . . . . . . . . . . . . . . . . . . . . . . . . . . . 19

D The Spectral decomposition 20
   D.1 Change of variables . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20
   D.2 Applying Gauss-Hermite . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
   D.3 Dimension reduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21

E Variance relations 22
   E.1 Variance of Y . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
   E.2 Relationship between the correlation of ε and Y . . . . . . . . . . . . . 22

F Solvency II correlation matrices 23
R-code

Function library .................................................. 25
Program 1 - Density plots for different \( t \) ........................................ 34
Program 2a - Accuracy in the \( K = 6 \) problem ................................ 35
Program 2b - Accuracy in the \( K = 12 \) problem ............................ 37
Program 3 - Testing \( t \)-values for the \( K = 12 \) problem .................... 38
Program 4 - Testing \( t \)-values for the \( K = 7 \) problem ...................... 40
Program 5a - Computing time experiments for the \( K = 12 \) problem ....... 42
Program 5b - Computing time experiments for the \( K = 7 \) problem ........ 44
1 Introduction

Solvency II is a set of regulations from the European Insurance And Occupational Pensions Authority (EIOPA) placed upon insurance companies conducting business in the EU. The directive institutes a collection of algorithms for calculating a set of quantities which will dictate the amount capital an insurance company will be required to hold to be allowed to offer insurance. Solvency II also issues a set of protocols with respect to reporting to national authorities and improving public disclosure in the insurance business. With its introduction EIOPA seeks to establish an EU wide license supporting the idea of a single market in the EU.

Risk aggregation in Solvency II takes a modular approach. Referring to Figure 1 which is taken from Bølviken (2017). Consider the overall risk of a modern insurance company which may be broken down into many different types of risks. Take for example the risk linked with market investments. This market risk may be viewed as a sum of sub-risks associated with the interest rate, equity, property and so on. On the other hand, the overall market risk may also be viewed as a part of a larger aggregated risk along with life risk, non-life risk etc.

The Solvency Capital Requirement (SCR) is a key quantity in Solvency II. The SCR represents the amount of capital an insurance company is required to hold in order to reliably be able to meet its liabilities. The 99.5 percent principle is a key feature in Solvency II which means that there should in principle be no more than a 0.5 percent chance of the insurance company going bankrupt within one year. In mathematical terms, we may think of the SCR as a percentile of an associated risk variable. An insurance company is required to calculate its top-level company SCR regularly. This is done by starting at the lowest possible level of Figure 1. Solvency II provides us with algorithms for approximating the SCRs connected to all of the input nodes at the lowest level. Then after obtaining all the risks making up a particular node one level up we aggregate these risks into a single new SCR for that node. For example after obtaining SCRs for Health, Health and Catastrophe these are used to aggregate up one level, giving us the Health SCR. After doing this for the other nodes Market, Life, Non-Life etc. these SCRs are used to aggregate up one level again. This continues until all of the risks making up the business have been aggregated to one single parent node. Details beyond what is presented in this thesis regarding this process can be found in EIOPA (2014) and Commission Delegated Regulations (2015).

This aggregation process uses a mathematical formula referred to as the Standard formula which is mathematically correct if the underlying risks are Gaussian. Consequently, the Standard Formula may seem fitting when considering for example financial risk which often exhibits approximate normality. Insurance risk however, will generally not fit any Gaussian assumption. This breach of assumption may reveal that the 99.5 percent principle in Solvency is misleading.

Bølviken and Guillen (2017) proposed an extension based on log-normal risks. They used the method of matching the moments of the sum of the underlying log-normal risk variables with that of a new log-normal variable. There is however another alternative from engineering literature presented by Methe et al. (2007). This method also creates a new log-normal variable, but through matching the moment-generating function of the new variable with that of the sum of the underlying log-normal variables. The purpose of this thesis is to develop and apply this technique to the model in Bølviken and Guillen (2017). Furthermore, the performance of the resulting method will be compared to the
method in Bølviken and Guillen (2017) and the Standard formula. The procedures will be implemented using the programming language R for which the source code will be provided on page 25. A large part of the time dedicated to this thesis was spent producing the code for these methods from scratch.

Figure 1: Risk evaluations in Solvency II and its aggregation. There is also another level below for some nodes (not in figure).

2 The Solvency II approach

2.1 Introduction

We will now try to formalize the methodology described in section 1. Consider a random variable $Z$ representing the total company net loss for the coming year. If it is negative then the business is profitable that year. The expectation $E[Z]$ is predictable and the Solvency II approach is to examine

$$Y = Z - E[Z].$$

The Solvency Capital Requirement is then the solution of the equation

$$P(Y > \text{SCR}) = \epsilon$$

with $\epsilon = 0.5\%$. In order to reasonably assume the company to be able to meet its obligations Solvency II require the insurance company to hold assets valued at least $E[Z] + \text{SCR}$ at the start of the year.

Consider the upper layer in Figure 1 with risk variables

$$Z = Z_{\text{oper}} + Z_{\text{basic}} + Z_{\text{adjust}}.$$

Subtracting the means then yields the variables of interest

$$Y = Y_{\text{oper}} + Y_{\text{basic}} + Y_{\text{adjust}}.$$
Equation (1) may be solved through the use of Monte Carlo, but that involves much modelling and is not the Solvency II way. Instead, SCRs on the lowest levels are calculated and aggregated iteratively in order to end up with the SCR for the entire company.

Consider for example the top-level SCR\textsuperscript{company} which is immediately comprised of and calculated from SCR\textsuperscript{oper}, SCR\textsuperscript{basic} and some adjustment term Adj. Now, SCR\textsuperscript{basic} is computed from the SCRs from market, life and so on, and those are in turn computed from the SCRs of their own underlying variables. Those may have other risk variables under them and the same technique of merging risks from their percentiles are used there. This scheme deals with many types of risk and the true distribution of these different risks may vary. Financial risk will often emit Gaussian traits while risk connected with say natural catastrophes in insurance will be much more skewed. With these considerations notwithstanding, the risks are handled in the same way in regards to risk aggregation. How is this carried out?

The general case is $K$ risk variables $Y_1, ..., Y_K$ with mean equal to zero for which their Solvency Capital Requirements SCR$_1, ..., SCR_K$ are to be aggregated to the corresponding SCR for the sum

$$Y_S = Y_1 + ... + Y_K.$$  

Of course, for the bottom nodes where the scheme starts the SCRs must be found by other methods. How this is done is not relevant to the scope of this thesis, but one may consult EIOPA (2014) to see how this is done for each node. Our objective is the aggregation of percentiles from one level in Figure 1 to another.

### 2.2 The Standard Formula

We still assume that we have $K$ nodes representing the dependent risk variables $Y_1, ..., Y_K$ making up a parent node and risk variable $Y_S$. These random variables $Y_k$ have expectation equal to zero and 99.5 percentiles SCR$_1, ..., SCR_K$. The dependence structure is handled through the use of correlation coefficients

$$\rho_{ij} = \text{cor}(Y_i, Y_j)$$

which are supplied by Solvency II documentation. See EIOPA (2014) or some of the samples on page 23. We then seek the corresponding SCR$_S$ or 99.5 percentile of the sum $Y_S = \sum Y_k$ that we obtain through the Standard formula

$$\text{SCR}_S = \left( \sum_{i=1}^{K} \sum_{j=1}^{K} \rho_{ij} \times \text{SCR}_i \times \text{SCR}_j \right)^{1/2}, \tag{2}$$

If the risk variables $Y_k$ are Gaussian with mean zero, then (2) is mathematically exact. To see this, observe that if we let $\phi$ be the 99.5th percentile of the standard normal distribution and denote the standard deviation of risk $k = 1, ..., K$ as $\sigma_k$ then

$$\text{SCR}_k = \phi \times \sigma_k, \quad k = 1, ..., K.$$  

Now take the variance formula for the sum $Y_S$, which is

$$\sigma_S = \left( \sum_{i=1}^{K} \sum_{j=1}^{K} \rho_{ij} \sigma_i \sigma_j \right)^{1/2},$$
and multiply both sides by $\phi$ and you arrive at (2).

The Standard formula may therefore seem fitting when considering financial risk which often exhibits approximate normality. Insurance risk, however, will generally not fit any Gaussian assumption. This breach of assumption may mean that the 99.5 percent principle in Solvency II is misleading. However, in spite of this, there is no denying the simplicity of (2) in terms of implementation and its ability to quickly aggregate many sources of risks quickly and easily. There is also the question of whether or not describing the dependence through the use of correlation coefficients is appropriate or not for all the different types of risk. We will not address that question in this paper. In the next section we will explore alternative methods to the Standard formula which will abandon any Gaussian assumption of the underlying risks $Y_k$ making up the sum. We will then compare the accuracy of these methods and eventually discuss any trade-offs with these methods in regards to practical implementation.

3 A log-normal standard formula

3.1 Introduction

As noted in the previous section the mathematics behind the Standard Formula which the risk aggregation in Solvency II is built upon assumes Gaussian risks. An alternative approach is to assume the underlying risks to be log-normal. Our model (3) is taken from Bolviken and Guillen (2017) and should better accommodate non-Gaussian risks skewed to the right which is common in insurance. We will assume risks of the form

$$Y_k = \sigma_k \frac{e^{-\tau_k^2/2+\tau_k \epsilon_k}}{\sqrt{e^{\tau_k^2} - 1}}, \quad k = 1, ..., K$$

(3)

where the elements in $\epsilon = (\epsilon_k)$ are Gaussian with mean zero, standard deviation 1 and correlation matrix $\Sigma$. Furthermore, $\sigma_k$ is the standard deviation of $Y_k$ and $\tau_k$ is a shape parameter. Given the shape parameter $\tau_k$ the skewness $\gamma_k$ is given by the formula

$$\gamma_k = (e^{\tau_k^2} + 2)\sqrt{e^{\tau_k^2} - 1}.$$  

One may show by application of L’Hôpital’s rule that $Y_k$ would approach a normal distribution if we were to let $\tau_k$ tend to zero. Choosing the log-normal distribution over a Gaussian represents a step towards more skewed distribution which should be closer to reality when it comes to insurance risk. The log-normal distribution in particular also has some properties that makes it easy to work with. Note that $Y_k$ has expectation zero.

Our goal is then to approximate the sum $\sum_{k=1}^{K} Y_k$ with a new log-normal variable

$$Y_S = \sigma_S \frac{e^{-\tau_S^2/2+\tau_S \epsilon_S}}{\sqrt{e^{\tau_S^2} - 1}}$$

for some new parameters $(\sigma_S, \tau_S)$. This problem has no exact solution. However, it has been studied a great deal and several methods have been developed.

3.2 The Fenton-Wilkinson approximation

Bolviken and Guillen (2017) have already studied this particular problem through the use of the so-called Fenton-Wilkinson scheme. They considered the skewness and standard
deviation of the sum of the log-normals $\sum Y_k$ and selected the new parameters $(\sigma_S, \tau_S)$ in order to match the skewness of $\sum Y_k$ and $Y_S$. They selected $\sigma_S$ through the ordinary variance-formula for a sum of correlated variables.

$$\sigma_S = \left( \sum_{i=1}^{n} \sum_{j=1}^{n} \rho_{ij} \sigma_i \sigma_j \right)^{1/2}$$

(5)

The skewness coefficients $\gamma_k$ is readily obtained through (4) and they showed the skewness of the sum $\sum Y_k$ to be given by

$$\gamma_S = \sum_i \alpha_i \beta_i \left( e^{\tau_i^2} - 1 \right)^2 + 3 \sum_{i \neq j} \alpha_i \alpha_j \beta_i \beta_j \left( e^{\tau_i^2} - 1 \right) \left( e^{\tau_j^2} - 1 \right) + 6 \sum_{i<j<k} \alpha_i \alpha_j \alpha_k \beta_i \beta_j \beta_k - \beta_i \beta_j - \beta_i \beta_k - \beta_j \beta_k + 2$$

(6)

where

$$\alpha_i = \frac{\sigma_i}{\sigma_S \sqrt{e^{\tau_i^2} - 1}} \quad \text{and} \quad \beta_{ij} = 1 + \rho_{ij} \sqrt{\left( e^{\tau_i^2} - 1 \right) \left( e^{\tau_j^2} - 1 \right)}.$$

For their proof see the appendix of Bolviken and Guillen (2017)

They then used these results to match the skewness of the new random variable $Y_S$ with $\sum Y_k$. Finally, with $\gamma_S$ in hand they then obtain $\tau_S$ by solving equation (4) with respect to $\tau_k$. This then fully specifies the new random variable $Y_S$ such that skew($\sum Y_k$) = skew($Y_S$).

This method may be generalized to distributions other than the log-normal. For eligible distributions, an exact expression for $\gamma_S$ might not exist in which case numerical methods may be introduced in order to solve the equations. Bolviken and Guillen (2017) also demonstrate how copulas may be incorporated instead of correlation coefficients to better capture different dependence structures. All in all this approach is a simple and flexible solution to the problem and their results suggest promising accuracy for problems with $K = 3$ log-normal risks for different degrees of skewness.

3.3 Matching moment-generating functions

We will now develop another way of approximating the sum of log-normals with a new log-normals. The method will have similarities with the methods of moment-matching in Bolviken and Guillen (2017), but will rather obtain the parameters $(\sigma_S, \tau_S)$ by matching the moment-generating functions at some point $t$.\(^1\) This concept is introduced in Metha et al. (2007), but we will also suggest a modification in an attempt to reduce the computational burden of problems with a large number of dimensions $K$.

We will consider the same model as Bolviken and Guillen. The properties of the log-normal distribution are well-studied. The pdf is given by

$$f_{Y_k}(y) = \frac{1}{\sigma} f_{\tilde{V}_k} \left( \frac{y}{\sigma} + 1 \right), \quad -\tilde{\sigma}_k < y < \infty$$

\(^1\)We will more specifically use the characteristic function which is a special case of the moment-generating function. However, we will not make any distinction between these two and treat them as one.
where we have defined $V_k := \exp(-\tau_k^2/2 + \tau_k \epsilon_k)$ and $\tilde{\sigma}_k := \sigma_k/\sqrt{e^{\tau_k^2} - 1}$ to simplify the expression. The pdf of $V_k$ is readily known as

$$f_{V_k}(v) = \frac{1}{v \tau_k \sqrt{2\pi}} \exp \left[-\frac{(\ln v - (\tau_k^2/2))^2}{2\tau_k^2}\right].$$

The moment-generating function (MGF) is written as

$$M_{Y_k}(t) = E \left[e^{-ty} \right] = \int_{-\tilde{\sigma}_k}^{\infty} \exp(-ty)f_{Y_k}(y)dy. \quad (7)$$

We may here interpret the exponential function as a weight function with a tuning parameter $t$. If we then set the MGF of $Y_S$ equal to the MGF of the sum $\sum Y_k$ we may solve for the parameters $(\tau_S, \sigma_S)$ at some point $t$.

We may also relate this to the skewness matching method in Bolviken and Guillen (2017) by noting that matching variance and skewness is equivalent to solving the equations

$$\int_{-\infty}^{\infty} w_1(y)f_{Y_S}(y)dy = \int_{-\infty}^{\infty} w_1(y)f_{\sum Y_k}(y)dy \quad (8)$$

for the corresponding weight functions $w_1(y) = (y - E[Y_k])^2 = y^2$ and $w_2(y) = (y - E[Y_k])^3 = y^3$. Different weight functions will translate to different penalization schemes. Using a weight function like $(y - E[Y_k])^3$ will penalize the tail portion of the density more than for example $\ln(y)$ which will penalize the head portion more. By choosing $\exp(-ty)$ as our weight function we will then be able to tune which parts of the density we want to emphasize. In particular, our focus will be the 99.5th percentile in Solvency II. This makes the method more flexible than similar methods which use a fixed weight function. On Figure 2 equation (8) is solved using the weight function $\exp(-ty)$ for different values of $t$ with the Cholesky approach described in section 3.3.1. Simulation of the sum directly using Monte Carlo is also included. Here we see that by taking $t = 5$, or weight function $\exp(-5y)$, errors in the tail of the distribution is penalized less. While with $t = 0.5$ we fit the tail much better. For values $t = 0.1$ and $t = 0.02$ it may seem that we need to go even further along the tail to a reach an area where the scheme benefits from such small values. On the other hand, $t = 5$ fits the head portion of the distribution quite well in comparison to the smaller values of $t$. The next step in the process is now to derive approximations for the MGF of the sum $\sum Y_k$ and the fitting variable $Y_S$. This will be done through the use of Gauss-Hermite expansions of the integrals.

### 3.3.1 Cholesky decomposition

Now, the MGF of the sum $\sum Y_k$ may be written on the following form

$$M_{\sum Y_k}(t) = E \left[e^{-t\sum Y_k} \right] = E \left[\prod_k \exp(-t\tilde{\sigma}_k(\exp(X_k) - 1))\right]$$

$$= \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} \prod_{k=1}^{K} \exp(-t\tilde{\sigma}_k(\exp(x_k) - 1))$$

$$\frac{1}{(2\pi)^{K/2}|C|^{1/2}} \exp \left\{ -\frac{1}{2}[x - (-\tau_k^2/2)]^T C^{-1} [x - (-\tau_k^2/2)] \right\} dx_1...dx_K \quad (9)$$
where $\boldsymbol{\tau} = (\tau_k)$, $\mathbf{X} = (X_k)$ is the Gaussian vector with $X_k = -\tau_k^2/2 + \tau_k \epsilon_k$ and the matrix $\mathbf{C}$ is the covariance matrix of $\mathbf{X}$.

There is no general closed-form expression for (7) and (9) but we may apply Gauss-Hermite approximations to the integrals for real values of $t$. For the 1-dimensional case of (7) we perform a standard change of variable to get an integrand compatible with Gauss-Hermite integration. We then end up with the following approximation.

$$M_{Y_k}(t) \approx \Psi_{Y_k}(t; \sigma_k, \tau_k) = \sum_{n=1}^{N} w_n \sqrt{\pi} \exp \left[ -t\tilde{\sigma}_k \left( \exp \left( \sqrt{2}\tau_k \alpha_n - \frac{\tau_k^2}{2} \right) - 1 \right) \right]$$  \hspace{1cm} (10)

$a_n$ and $w_n$ are the abscissas and weights of the Gauss-Hermite expansion of order $N$, respectively. The derivation of (10) may be found in the appendix section B on page 18.

The MGF of the sum is more complicated because of the covariance matrix $\mathbf{C}$ which we need to get rid of before Gauss-Hermite expansion may be applied. One way of doing this is by applying the following Cholesky transformation as done in Mehta et al. (2007)

$$x_k = \sqrt{2} \sum_{i=1}^{K} l_{ki} z_i - \tau_k^2/2, \quad k = 1, \ldots, K$$

where $L = (l_{ki})_{k,i=1}^{K}$ is the Cholesky decomposition of $\mathbf{C}$, i.e. $\mathbf{C} = \mathbf{L}\mathbf{L}^T$. This removes the covariance matrix from the last exponent in (9) and makes the integration variables independent in a sense. After iteratively applying Gauss-Hermite expansion with respect to the variables of integration this approach yield the following approximation to the MGF of the sum $\sum Y_k$.

$$M_{\sum Y_k}(t) \approx \Psi_{\sum Y_k}(t; \boldsymbol{\sigma}, \boldsymbol{\tau}, \mathbf{C}) = \sum_{n_K=1}^{\mathbf{N}} \cdots \sum_{n_1=1}^{\mathbf{N}} \frac{w_{n_1} \cdots w_{n_K}}{\pi^{K/2}} \prod_{k=1}^{K} \exp \left\{ -t\tilde{\sigma}_k \left[ \exp \left( \sqrt{2} \sum_{j=1}^{K} l_{kj} \alpha_{n_j} - \frac{\tau_k^2}{2} \right) - 1 \right] \right\}$$  \hspace{1cm} (11)
Details in the appendix section C on page 19. Then given parameters $\tau$, $\sigma$, $\sigma_S$ and covariance matrix $C$ we may take some value of $t$ and solve

$$\Psi_{\sum Y_k}(t; \sigma, \tau, C) = \Psi_{Y_S}(t; \sigma_S, \tau_S)$$

with respect to $\tau_S$. $\sigma_S$ is already given by

$$\sigma_S^2 = \text{Var} \left( \sum Y_k \right) = \sum_{i,j=1}^{K} \text{cov}(Y_i, Y_j) = \sum_{i,j=1}^{K} \rho_{ij} \sigma_i \sigma_j.$$

### 3.3.2 Spectral decomposition

We will now introduce an alternative to the expression in (11) based on spectral decomposition rather than Cholesky decomposition. This will open up the possibility of easing the computational burden by reducing the number of sums in an expression like (11) from $K$ to some smaller number $L$. The expression in (11) sums $N^K$ products. In Solvency II we will encounter $K = 12$ dimensional problems and these computations will quickly become non-trivial even for a modern computer.

Consider the correlation matrix of $\epsilon$, $\Sigma$, and its spectral decomposition

$$\Sigma = \Gamma \Lambda \Gamma^T$$

where the $i$-th column vector of $\Gamma$ is the $i$-th eigenvector of $\Sigma$ with corresponding eigenvalue $\lambda_i$ sorted by descending eigenvalue. $\Lambda$ is the matrix with eigenvalues along the diagonal in descending order. Instead of the Cholesky decomposition used above we instead consider the transformation

$$\delta = \Gamma^T \epsilon \quad \text{or} \quad \epsilon = \Gamma \delta.$$

We may then write the model using this transformation

$$Y_k = \tilde{\sigma}_k \left( \exp \left( -\tau_k^2/2 + \tau_k \sum_{j=1}^{K} \gamma_{kj} \delta_j \right) - 1 \right)$$

with $\gamma_{kj}$ being the element $(k, j)$ in $\Gamma$. This is useful because since $\delta \sim N(0, \Lambda)$ the elements $\delta_j$ are independent. We may also write the model as

$$Y_k = \tilde{\sigma}_k \left( \exp \left( -\tau_k^2/2 + \tau_k \sum_{j=1}^{K} \gamma_{kj} \sqrt{\lambda_j} \eta_j \right) - 1 \right)$$

where $\eta \sim N(0, I)$. Using this expression the MGF of the sum $\sum Y_k$ can be written as

$$M_{\sum Y_k}(t) = \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} \prod_{k=1}^{K} \exp \left\{ -t \tilde{\sigma}_k \left( \exp \left[ -\tau_k^2/2 + \tau_k \sum_{j=1}^{K} \gamma_{kj} \sqrt{\lambda_j} \eta_j \right] - 1 \right) \right\} \times \frac{1}{(2\pi)^{K/2}} \exp \left( -\frac{1}{2} \eta^T \eta \right) d\eta_1 \cdots d\eta_K. \quad (13)$$

Then utilizing a small change of variable and applying Gauss-Hermite expansion iteratively, as done in appendix D on page 20, yields

$$M_{\sum Y_k}(t) \approx \hat{\Psi}_{\sum Y_k}(t; \sigma, \tau, \Sigma)$$

$$= \sum_{n_K=1}^{N} \cdots \sum_{n_1=1}^{N} \frac{w_{n_1} \cdots w_{n_K}}{\pi^{K/2}} \prod_{k=1}^{K} \exp \left\{ -t \tilde{\sigma}_k \left( \exp \left[ -\tau_k^2/2 + \tau_k \sum_{j=1}^{K} \gamma_{kj} \sqrt{2\lambda_j a_{n_j}} \right] - 1 \right) \right\}. \quad (14)$$
3.3.3 Dimension reduction

The spectral decomposition gives us the option of dropping the smallest principal components and only including the $L < K$ largest ones. This gives us the following approximation:

$$
\epsilon_k = \sum_{j=1}^{K} \gamma_{kj} \delta_j \approx \sum_{j=1}^{L} \gamma_{kj} \delta_j
$$

The idea is that for $L < k \leq K$, $\text{Var}(\delta_k) = \lambda_k$ will be close to zero and $\delta_k$ will be approximately $\mathcal{N}(0,0)$. Moving forward with this yields the approximate MGF of the sum

$$
M_{\sum Y_k}(t) \approx \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} \prod_{k=1}^{K} \exp \left\{ -t \tilde{\sigma}_k \left( \exp \left[ -\frac{\tau_k^2}{2} + \tau_k \sum_{j=1}^{L} \gamma_{kj} \sqrt{\lambda_j} \eta_j \right] - 1 \right) \right\}
\times \frac{1}{(2\pi)^{L/2}} \exp \left( -\frac{1}{2} \eta^T \eta \right) \, d\eta_1 \cdots d\eta_L
$$

and the Gauss-Hermite expansion

$$
M_{\sum Y_k}(t) \approx \tilde{\Psi}_{\sum Y_k}(t; \sigma, \tau, \Sigma)
= \sum_{n_L=1}^{N} \cdots \sum_{n_1=1}^{N} \frac{w_{n_1} \cdots w_{n_L}}{(2\pi)^{L/2}} \prod_{k=1}^{K} \exp \left\{ -t \tilde{\sigma}_k \left( \exp \left[ -\frac{\tau_k^2}{2} + \tau_k \sum_{j=1}^{L} \gamma_{kj} \sqrt{2\lambda_j} a_{n_j} \right] - 1 \right) \right\}
$$

The efficiency gain in using (16) instead of (14) will be examined below.

4 Numerical experiments

4.1 Accuracy

In Figure 3\textsuperscript{2} the absolute error of the different methods are compared in a problem with $K = 6$ risks and $K = 12$ risks. The x-axis denotes the number of large risks with a large scale ($\sigma_k = 4$) and heavy skewness ($\tau_k = 1.2$). The rest of the $K$ risks where smaller and with much lighter skewness ($\sigma_k = 1$ and $\tau_k = 0.25$). For each set of parameters the resulting 99.5th percentiles where compared with that of a Monte Carlo simulation of the sum with a size of $10^7$ simulations. For the $K = 6$ problem a $t$-value of 0.1 where chosen, while in the $K = 12$ problem $t$ was set to 0.08 for both the Cholesky and full Spectral (PC) approach. For the PC model with $L = K - 1$ $t$ was set equal to 0.05. These values are believed to provide good results based on analysis in section 4.2. The correlation matrices of $(\epsilon_k)$ where set equal to the 6- and 12-dimensional correlation matrices from Solvency II documentation, see EIOPA (2014) or page 23. In Solvency II these matrices are meant to define correlation on the risk-level, i.e. correlation of $(Y_k)$. However, the resulting correlation matrix of $(\epsilon_k)$, given the correlation of $(Y_k)$, will not always end up being positive definite for any choice of $\sigma_k$ and $\tau_k$. We avoid this by directly defining $\Sigma = \text{cor}(\epsilon_k)$ equal to the correlation matrix in Solvency II. By starting with a valid correlation structure of $(\epsilon_k)$ the resulting correlation matrix of $(Y_k)$ will be necessarily positive definite. In this experiment $\text{cor}(Y_k)$ will change for each set of $(\tau_k)$ and $(\sigma_k)$ along

\textsuperscript{2}This section refers to figures placed towards the end of the document on page 14.
the x-axis. However, the resulting matrices are easily calculated through the results in the appendix on page 23, specifically using result (E.8), or via the R-code on page 35.

Firstly note the poor performance of the classical Solvency II Standard formula. The Standard formula especially has problems with the heavily skewed risks. Since the risks stray further and further away from normality as \( \tau_k \) increases this is expected. The method introduced in Bolvikken and Guillen (2017) provide very good results in the case of only lightly skewed risks. However, when larger, more heavy-tailed risks are introduced the Bolvikken-Guillen (BG) method deteriorates abruptly in performance. This method along with the Standard formula seem to do worse and worse as more and more heavy-tailed risks are introduced. The BG method also seems to have trouble when introduced to a large number of risks as in the \( K = 12 \) problem.

In the \( K = 6 \) problem in Figure 3a the MGF-methods arguably perform better than BG as they handle the large risks better. Figure 3a might also suggest that the values of \( \sigma_k \) and \( \tau_k \) plays a part in determining the value of \( \tau \) that minimizes the absolute error.

The \( K = 12 \) problem provides similar results. The BG method performs very well when all risks are small and slightly skewed. However, when introduced to large risks the method again immediately suffers. The MGF methods also show a slightly similar tendency of performing the best when only having small risks with small skewness. Else the absolute accuracy seems relatively constant as we introduce more and more large-scaled risks. This shows considerable improvement on BG and especially the Standard formula. Using the PC-decomposition and discarding the last component also yield good results, albeit slightly worse than using the full model \( L = K \). One can note for zero large risks the optimal \( \tau \) in the \( L = K - 1 \) case is relatively far from the selected value \( \tau = 0.05 \) in 3b which explains the low accuracy in that point. The sensitivity in the design parameter \( \tau \) will be discussed below in section 4.2.

One may differentiate the large and small risks even more by building a new correlation structure. Consider for example letting the large risks be more strongly correlated with each other and much less correlated with the smaller ones. However, this does not seem to have a great impact on the performance of the different methods. This was tested, but not plotted in this document.

The MGF methods also seem to scale down to small-dimensional problems well with \( K \) as low as 3 being tested. The resulting plots showed a similar story to that of Figure 3 (Not plotted).

### 4.2 Design of the MGF method

When applying the MGF method to a problem with given parameters we need to make a choice of \( N \), the Gauss-Hermite degree, and the point \( \tau \) for which to solve (12). We start by considering \( \tau \) which will dictate what part of the density we will approximate the best as discussed on page 6.

Consider the \( K = 12 \) problem on Figure 4 where different \( \tau \)-values were tested and the corresponding 99.5th percentile is plotted for the MGF methods. An even mix of large and small risks were used as in section 4.1. Increasing \( \tau \) makes the resulting density of the new variable \( Y \) fit the lower end of the distribution well, but the 99.5th percentile will be underestimated. On the other hand, a \( \tau \)-value that is too small will provide a good fit further out than our target percentile. In which case the lower part of the distribution will be affected by this which will mean that our target 99.5th percentile will be overestimated. For this problem we seem to hit a sweet spot somewhere between 0.05 and 0.1. Choice of
$N$ seem to interact with the resulting plots when comparing the $N = 4$ case in Figure 4b and the $N = 3$ case in Figure 4a.

Discarding principal components has a noticeable effect, especially on low $t$-values. However, decent accuracy is still obtainable in the $K = 12$ problem. Another interesting aspect is how much the Cholesky approach degrades for very high $t$-values compared to the PC approach. This is also observable in the $K = 6$ problem in Figure 3a.

For the $K = 6$ problem the similar balanced large-small risk distributions were tested along with letting every risk be small. For the balanced scenario, the results were similar to above with the exception of the reduced model $L = 6$ which performed very close to the full models. When using only light risks both the Cholesky and PC approach (with $L = K$) are very flexible in terms of $t$-values that yield near-perfect results. On Figure 5b we observe that any $t$-value up to 0.5 yield great results, with significant error building up only with values higher than this. This suggests that the nature of the risks has a significant impact on how the methods react to different values of $t$.

### 4.3 Computing time experiments

Along with $K$, which is already given by the problem one will be facing, $N$ will dictate the computational complexity of the MGF methods. By considering the first $K$ sums in the expression in (11) we already end up with $N^K$ terms. This is before including the product and sum in the middle and right end of the expression. With this in mind, $N$ and $K$ will both dramatically increase the number of operations necessary to evaluate the resulting expressions. Similar observations are valid for the Spectral approach with the expressions (14) and (16). Minimizing $N$ as much as possible while still achieving a reasonable accuracy should therefore be a high priority.

The time needed for the different procedures in the $K = 12$ and $K = 7$ problems is plotted on Figure 6 both for the Cholesky approach and the Spectral approach with different values of $L$. This is shown as a function of $N$. For this problem an even mix of large, heavy-tailed risks and smaller, lightly-tailed risks were chosen. The details may be found in the R-code on page 42 and 44. Here, $t$ was set equal to 1. Note that changing the risk parameters $\sigma_k$, $\tau_k$, $\Sigma$ or $t$ have negligible effect on the time spent computing. However, it is worth mentioning that not all values of $t$ provide the problem with a solution. Low values of $L$ in particular does not yield a solution to (12) for low values of $t$.

First we will consider the $K = 12$ problem on Figure 6a. The computation time seems to suddenly spike for the Cholesky approach and Spectral approach for large values of $L$. This makes sense considering the number of terms $N^K$ (or $N^L$) noted above. Note that the Cholesky approach seems to be more efficient than using a Spectral decomposition with $L = K$. However, by letting $L = 11$ we may potentially reduce the time required even further. For the $K = 12$ problem going from $N = 3$ to $N = 4$ represents a big jump in computational time for the Cholesky and Spectral approaches. Yet, we notice from section 4.2 that $N = 3$ can achieve good results with less than 30 seconds of computation time.

Now for the smaller $K = 7$ problem in Figure 6b we obtain reasonably small computation times even for very large $N$. Further experiments indicate that here is little to no improvement in increasing $N$ further than 4. For the $K = 7$ this problem will then be solved in 1 second. We still note that the Cholesky approach edges out the full Spectral approach in terms of time needed to solve the problem.
5 Conclusion

The MGF methods consistently provide the best results in terms of accuracy. In the previous section the MGF methods have been shown to provide good accuracy in across a number of different scenarios. We have looked at scenarios with only lightly skewed risks, with only heavily skewed risks and with a mixture of lightly and heavily skewed risks. The method of Bølviken and Guillen (2017) handles the first scenario well, but its accuracy suffers when introduced to more heavily skewed risks. The MGF methods presented here does not seem to have this problem but instead produce consistently good results across all of the scenarios.

The Standard formula handled the log-normal risks quite poorly, but on the other hand it is, along with the BG method, quite simple to implement. The MGF methods, on the other hand, are relatively demanding to implement and require a design parameter $t$ to be set. The sensitivity of the choice of $t$ is dependent on the risks and should be studied further if one were to consider implementing the MGF methods into a system like Solvency II which handles different types of risk. EIOPA would either have to set the value of $t$ pre-determinately or provide some form of automatic procedure which would provide a $t$-value. If an insurance company could freely choose its $t$-values then it could knowingly underestimate the SCRs by intentionally setting them too high as shown on Figure 2.

For a problem with a high number of risks computation time for the MGF methods may be a concern for some. The time required was shown to be non-trivial in the 12-dimensional problem. A dimension reduction method was developed with such problems in mind and tested along with the full models. Combining this with reducing the Gauss-Hermite parameter $N$ to 3 was shown to significantly reduce computation time while still offering good accuracy mostly on par with the full-model approaches.
References


Appendices

A Figures - Numerical experiments


(b) The $K = 12$ problem. $N = 3$

Figure 3: Distance between the 99.5th percentiles of the Monte Carlo simulations and for the methods in the $K = 6$ and $K = 12$ problems with increasing number of large, heavy-tailed risks.
Figure 4: Resulting 99.5th percentiles for different values of t in the K = 12 problem. Note non-linear x-axis.
(a) 3 heavy risks with \((\sigma_k, \tau_k) = (4, 1.2)\) and 4 light risks with \((\sigma_k, \tau_k) = (1, 0.25)\).

(b) Only light risks

Figure 5: Resulting 99.5th percentiles for different values of \(t\) in the \(K = 7\) problem. Note non-linear x-axis.
(a) The $K = 12$ problem. Using 6 risks with $(\sigma_k, \tau_k) = (4, 1.2)$ and 6 with $(\sigma_k, \tau_k) = (1, 0.25)$.

(b) The $K = 7$ problem. Using 3 risks with $(\sigma_k, \tau_k) = (1.4, 4)$ and 4 with $(\sigma_k, \tau_k) = (0.25, 1)$.

Figure 6: Time used to solve $\Psi_{\sum Y_k} = \Psi_{Y_S}$ for different values $N$. Comparing the Cholesky approach to the spectral decomposition approach for different values of $L$. Using correlation matrix from Solvency II documentation.
B PDF and MGF

B.1 PDF of our log-normal model

Our model variable $Y$ is given as a function of a regular log-normal variable $V$ through the transformation

$$Y = \sigma \frac{e^{-\tau^2/2+\tau} - 1}{\sqrt{e^{\tau^2} - 1}} = \tilde{\sigma}(V - 1) = g(V) \quad (B.1)$$

where $\tilde{\sigma} = \frac{\sigma}{\sqrt{e^{\tau^2} - 1}}$ and $V = e^{-\tau^2/2+\tau} \sim \ln \mathcal{N}(-\tau^2/2, \tau)$. Then by the change of variable formula the density of $Y$ is readily obtained through

$$f_Y(y) = \left| \frac{d}{dy} g^{-1}(y) \right| f_V(g^{-1}(y))$$

$$= \frac{1}{\tilde{\sigma}} f_V \left( \frac{y}{\tilde{\sigma}} + 1 \right) \quad (B.2)$$

where

$$f_V(v) = \frac{1}{v \tau \sqrt{2\pi}} \exp \left[ -\frac{(\ln v - (-\tau^2/2))^2}{2\tau^2} \right].$$

The support of $Y$ is $(-\tilde{\sigma}, \infty)$.

B.2 The moment-generating function of $Y$ and Gauss-Hermite

The moment-generating function of a random variable is defined as

$$M_Y(t) = E \left[ e^{-tY} \right], \ t \in \mathbb{R},$$

wherever this expectation exists. Using equation (B.2) this becomes

$$M_Y(t) = E \left[ e^{-tY} \right] = \int_{-\tilde{\sigma}}^{\infty} \exp(-ty) f_Y(y) dy$$

$$= \int_{-\tilde{\sigma}}^{\infty} \exp(-ty) \frac{1}{\tilde{\sigma}} f_V \left( \frac{y}{\tilde{\sigma}} + 1 \right) \frac{1}{\tau \sqrt{2\pi}} \exp \left[ -\frac{(\ln \left( \frac{y}{\tilde{\sigma}} + 1 \right) + \tau^2/2)^2}{2\tau^2} \right] dy. \quad (B.3)$$

In order to end up with the expression in (10) we need to use substitution to first arrive at an integrand compatible with Gauss-Hermite integration. Set

$$z = \frac{\ln(y/\tilde{\sigma} + 1) + \tau^2/2}{\sqrt{2\tau}}$$

giving

$$y = \tilde{\sigma} \left[ \exp(\sqrt{2}\tau z - \tau^2/2) - 1 \right].$$

We then get

$$\frac{dz}{dy} = \frac{1}{\sqrt{2\tau} (\tilde{\sigma} + y)} \quad \text{or} \quad dy = \sqrt{2\tau} (\tilde{\sigma} + y) dz.$$  

Finally, consider the integration limits. If $y \to -\tilde{\sigma}$ then $\ln(y/\tilde{\sigma} + 1) \to -\infty$ and $z \to -\infty$. Now if $y \to \infty$ then $\ln(y/\tilde{\sigma} + 1) \to \infty$ and $z \to \infty$. Therefore the integral (B.3) becomes

$$M_y(t) = \int_{-\infty}^{\infty} \frac{1}{\sqrt{\pi}} \exp \left[ -t\tilde{\sigma} \left( \exp(\sqrt{2}\tau z - \tau^2/2) - 1 \right) \right] \exp(-z^2) dz$$

$$\approx \sum_{n=1}^{N} \frac{w_n}{\sqrt{\pi}} \exp \left[ -t\tilde{\sigma} \left( \exp(\sqrt{2}\tau x_n - \tau^2/2) - 1 \right) \right] \quad (B.4)$$
which is the expression in (10).

C The Cholesky decomposition

C.1 Change of variable

We have the full expression of the MGF of \( \sum Y_k \) as on page 6:

\[
M_{\sum Y_k}(t) = \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} \prod_{k=1}^{K} \exp(-t\bar{\sigma}_k(\exp(x_k) - 1)) \frac{1}{(2\pi)^{K/2}|C|^{1/2}} \exp \left\{ -\frac{1}{2}[x - (-\tau^2/2)]^T C^{-1} [x - (-\tau^2/2)] \right\} dx_1 \ldots dx_K \tag{C.1}
\]

We will now do two things with one change of variable. Firstly, we want to obtain independence in the integrand, i.e. we want expression like (C.1) but where \( x \) independent. Secondly, we want to get an integrand compatible with Gauss-Hermite integration. In order to achieve the first goal, we consider the Cholesky decomposition in order to decorrelate \( x \). \( \mathbf{X} = (X_k) \) is the Gaussian vector \( X_k = -\tau_k^2/2 + \tau_k \epsilon_k \) and the matrix \( C \) is the covariance matrix of \( \mathbf{X} \). Let \( C = LL^T \) be its Cholesky decomposition. We then know that if we consider an arbitrary vector \( z \) with mean zero and covariance equal to the identity matrix that the transformation \( Lz \) will be a mean-zero vector with covariance matrix equal to \( C \). If we furthermore multiply by \( \sqrt{2} \) and add \(-\tau^2/2 \) to the transformation our integrand will be Gauss-Hermite compatible. Our change of variable then becomes

\[
x = \sqrt{2}Lz - \tau^2/2
\]

or

\[
x_k = \sqrt{2} \sum_{i=1}^{K} l_{ki} z_i - \tau_k^2/2, \quad k = 1, \ldots, K.
\]

We then get \( \frac{dx_k}{dz_i} = \sqrt{2}l_{ki} \) and the Jacobian becomes \( J(z) = \sqrt{2}L \) and \( |J(z)| = 2^{K/2}|L| = 2^{K/2}|C|^{1/2} \). The MGF then becomes

\[
M_{(\sum Y_k)}(t) = E[e^{-t \sum Y_k}] = \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} \prod_{k=1}^{K} \exp \left\{ -t\bar{\sigma}_k \left( \exp \left[ \sqrt{2} \sum_{i=1}^{K} l_{ki} z_i - \tau_k^2/2 \right] - 1 \right) \right\} \frac{1}{(2\pi)^{K/2}|C|^{1/2}} \exp \left\{ -\frac{1}{2}(\sqrt{2}Lz)^T C^{-1} (\sqrt{2}Lz) \right\} 2^{K/2}|C|^{1/2} dz_1 \ldots dz_K
\]

\[
= \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} \prod_{k=1}^{K} \exp \left\{ -t\bar{\sigma}_k \left( \exp \left[ \sqrt{2} \sum_{i=1}^{K} l_{ki} z_i - \tau_k^2/2 \right] - 1 \right) \right\} \exp(-z^T z) dz_1 \ldots dz_K. \tag{C.2}
\]

C.2 Applying Gauss-Hermite

We factor out \( z_1 \) and apply Gauss-Hermite expansion with respect to \( z_1 \):
\[ M(\sum Y_k)(t) = \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} \frac{1}{\pi^{(K-1)/2}} \exp \left(-\sum_{i=2}^{K} z_i^2\right) \sum_{n_1=1}^{N} w_{n_1} \prod_{k=1}^{K} \exp \left\{ -t\tilde{\sigma}_k \left[ \exp \left( \sqrt{2} \sum_{i=2}^{K} l_{ki} z_i + \sqrt{2} l_{i1} a_{n_1} - \frac{\tau_i^2}{2} \right) - 1 \right] \right\} + R_N^{(1)} \]

where we have some error term in the end and

\[ f_1(z^{(-1)}, a_{n_1}; t) = \prod_{k=1}^{K} \exp \left(-t\tilde{\sigma}_k \left[ \exp \left( \sqrt{2} \sum_{i=2}^{K} l_{ki} z_i + \sqrt{2} l_{i1} a_{n_1} - \frac{\tau_i^2}{2} \right) - 1 \right] \right). \]

Then again with \( z_2 \)

\[ M(\sum Y_k)(t) = \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} \frac{1}{\pi^{(K-2)/2}} \exp \left(-\sum_{i=3}^{K} z_i^2\right) \sum_{n_2=1}^{N} \sum_{n_1=1}^{N} w_{n_1} w_{n_2} \prod_{k=1}^{K} \exp \left\{ -t\tilde{\sigma}_k \left[ \exp \left( \sqrt{2} \sum_{i=3}^{K} l_{ki} z_i + \sqrt{2} \sum_{m=1}^{2} l_{im} a_{n_m} - \frac{\tau_k^2}{2} \right) - 1 \right] \right\} + R_N^{(2)} \]

where

\[ f_2(z^{(-2)}, a_{n_1}, a_{n_2}; t) = \prod_{k=1}^{K} \exp \left(-t\tilde{\sigma}_k \left[ \exp \left( \sqrt{2} \sum_{i=3}^{K} l_{ki} z_i + \sqrt{2} \sum_{m=1}^{2} l_{im} a_{n_m} - \frac{\tau_k^2}{2} \right) - 1 \right] \right). \]

By continuing this we end up the expression

\[ M(\sum Y_k)(t) = \sum_{n_K=1}^{N} \cdots \sum_{n_1=1}^{N} \frac{w_{n_1} \cdots w_{n_K}}{\pi^{K/2}} \prod_{k=1}^{K} \exp \left\{ -t\tilde{\sigma}_k \left[ \exp \left( \sqrt{2} \sum_{j=1}^{K} l_{kj} a_{n_j} - \frac{\tau_k^2}{2} \right) - 1 \right] \right\} + R_N^{(K)} \cdot \]

(C.3)

This then justifies the expression in (11) where the error term is discarded.

\section*{D The Spectral decomposition}

\subsection*{D.1 Change of variables}

The strategy and technique is the same as with the Cholesky decomposition. We start with the expression in (13) and apply a small change of variables before we iteratively apply Gauss-Hermite expansion with respect to the variables of integration. We start with

\[ M_{\sum Y_k}(t) = \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} \prod_{k=1}^{K} \exp \left\{ -t\tilde{\sigma}_k \left[ \exp \left( -\tau_k^2/2 + \tau_k \sum_{j=1}^{K} \gamma_{kj} \sqrt{\lambda_j} \eta_j \right) - 1 \right] \right\} \]

\[ \times \frac{1}{(2\pi)^{K/2}} \exp \left( -\frac{1}{2} \eta^T \eta \right) d\eta_1 \cdots d\eta_K. \]

Setting \( \eta = \sqrt{2}z \) will remove the fraction in the last exponent. This gives us the Jacobian \( J(z) = \sqrt{2}I \) and \( |J(z)| = 2^{K/2} \) and the expression above becomes
\[ \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} \prod_{k=1}^{K} \exp \left\{ -t\bar{\sigma}_k \left( \exp \left[ -\tau_k^2/2 + \tau_k \sum_{j=1}^{K} \Gamma_{kj} \sqrt{2\lambda_j} z_j \right] - 1 \right) \right\} \]
\[ \times \frac{1}{\pi^{K/2}} \exp \left( -\sum_{k=1}^{K} z_k^2 \right) dz_1 \cdots dz_K. \]

D.2 Applying Gauss-Hermite

First with respect to \( z_1 \):

\[ M_{\Sigma Y_k}(t) = \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} \frac{1}{\pi^{(K-1)/2}} \sum_{n_1=1}^{N} \frac{w_{n_1}}{\pi^{1/2}} f_1(z^{(-1)}, a_{n_1}) \exp \left( -\sum_{k=2}^{K} z_k^2 \right) dz_2 \cdots dz_K + R_N^{(1)} \]

where we have some error term in the end and

\[ f_1(z^{(-1)}, a_{n_1}) = \prod_{k=1}^{K} \exp \left\{ -t\bar{\sigma}_k \left( \exp \left[ -\tau_k^2/2 + \tau_k \sum_{j=2}^{K} \Gamma_{kj} \sqrt{2\lambda_j} z_j + \tau_k \sum_{j=1}^{1} \Gamma_{kj} \sqrt{2\lambda_j} a_{n_j} \right] - 1 \right) \right\}. \]

Then again with respect \( z_2 \):

\[ M_{\Sigma Y_k}(t) = \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} \frac{1}{\pi^{(K-2)/2}} \sum_{n_2=1}^{N} \sum_{n_1=1}^{N} \frac{w_{n_1}w_{n_2}}{\pi^{2/2}} f_2(z^{(-2)}, a_{n_1}, a_{n_2}) \exp \left( -\sum_{k=3}^{K} z_k^2 \right) dz_3 \cdots dz_K + R_N^{(2)} \]

where

\[ f_2(z^{(-2)}, a_{n_1}, a_{n_2}) = \prod_{k=1}^{K} \exp \left\{ -t\bar{\sigma}_k \left( \exp \left[ -\tau_k^2/2 + \tau_k \sum_{j=3}^{K} \Gamma_{kj} \sqrt{2\lambda_j} z_j + \tau_k \sum_{j=1}^{2} \Gamma_{kj} \sqrt{2\lambda_j} a_{n_j} \right] - 1 \right) \right\}. \]

Continuing this yields

\[ M_{\Sigma Y_k}(t) = \sum_{n_K=1}^{N} \cdots \sum_{n_1=1}^{N} \frac{w_{n_1} \cdots w_{n_K}}{\pi^{K/2}} \prod_{k=1}^{K} \exp \left\{ -t\bar{\sigma}_k \left( \exp \left[ -\tau_k^2/2 + \tau_k \sum_{j=1}^{K} \Gamma_{kj} \sqrt{2\lambda_j} a_{n_j} \right] - 1 \right) \right\} + R_N^{(K)} \]

(D.4)

which then justifies the expression in (14) where the error term is discarded.

D.3 Dimension reduction

The same strategy applies in this case, but still written out here for completeness. We start with expression (15).

\[ M_{\Sigma Y_k}(t) \approx \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} \prod_{k=1}^{K} \exp \left\{ -t\bar{\sigma}_k \left( \exp \left[ -\tau_k^2/2 + \tau_k \sum_{j=1}^{L} \gamma_{kj} \sqrt{\lambda_j} \eta_j \right] - 1 \right) \right\} \]
\[ \times \frac{1}{(2\pi)^{L/2}} \exp \left( -\frac{1}{2} \eta^T \eta \right) d\eta_1 \cdots d\eta_L \] (D.5)
We make use the same change of variables \( \eta = \sqrt{2z} \) yielding the same Jacobian (but with \( L \) instead of \( K \), naturally) and we get

\[
\int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} \prod_{k=1}^{K} \exp \left\{ -t\tilde{\sigma}_k \left( \exp \left[ -\frac{\tau_k^2}{2} + \tau_k \sum_{j=1}^{L} \Gamma_{kj} \sqrt{2\lambda_j} \right] - 1 \right) \right\} \times \frac{1}{\pi^{L/2}} \exp \left( -\sum_{k=1}^{L} z_k^2 \right) dz_1 \cdots dz_L.
\]

Applying Gauss-Hermite again with respect to \( z_1 \) yields

\[
M_{\sum Y_k}(t) = \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} \frac{1}{\pi^{(L-1)/2}} \sum_{n_1=1}^{N} \frac{w_{n_1}}{\pi^{L/2}} f_1(z^{(-1)}, a_{n_1}) \exp \left( -\sum_{k=2}^{L} z_k^2 \right) dz_2 \cdots dz_L + R_N^{(1)}
\]

where we have some error term in the end and

\[
f_1(z^{(-1)}, a_{n_1}) = \prod_{k=1}^{K} \exp \left\{ -t\tilde{\sigma}_k \left( \exp \left[ -\frac{\tau_k^2}{2} + \tau_k \sum_{j=2}^{L} \Gamma_{kj} \sqrt{2\lambda_j} z_j + \tau_k \sum_{j=1}^{1} \Gamma_{kj} \sqrt{2\lambda_j} a_{nj} \right] - 1 \right) \right\}.
\]

By doing this for \( z_2, \ldots, z_L \) as well we arrive at the expression in (16).

### E Variance relations

#### E.1 Variance of \( Y_S \)

In the program code on page 25 a formula for obtaining \( \text{Var}(\sum Y_k) \) from \( \tau, \sigma \) and \( \Sigma \) were needed. We let \( X_k = -\frac{\tau_k^2}{2} + \tau_k \epsilon_k \), \( \tilde{\sigma}_k = \sigma_k / \sqrt{e^{\tau_k^2} - 1} \) and \( c_{ij} = \text{cov}(X_i, X_j) = \tau_i \tau_j \Sigma_{ij} \).

Then by the formula on top of page 150 of Bolviken (2014) we have the following:

\[
w_{ij} = \text{cov}(e^{X_i}, e^{X_j}) = e^{-\left( \tau_i^2 + \tau_j^2 \right)/2 + (c_{ii} + c_{jj})/2} \left( e^{c_{ij}} - 1 \right)
\]

\[
\Rightarrow v_{ij} = \text{cov}(Y_i, Y_j) = \text{cov}(\tilde{\sigma}_i \exp(X_i) - 1, \tilde{\sigma}_j \exp(X_j) - 1)
\]

\[
= \text{cov}(\tilde{\sigma}_i \exp(X_i), \tilde{\sigma}_j \exp(X_j)) = \tilde{\sigma}_i \tilde{\sigma}_j v_{ij}
\]

Then the variance of the sum follows from the usual formula for variance of a sum of correlated variables.

\[
\sigma_S^2 = \text{Var}(Y_S) = \text{Var}(\sum Y_k) = \sum_{i,j=1}^{K} \text{cov}(Y_i, Y_j) = \sum_{i,j=1}^{K} v_{ij}
\]

#### E.2 Relationship between the correlation of \( \epsilon \) and \( Y \)

In some of the programs on page 25 we want to calculate the resulting correlation matrix of \( \epsilon_k \) given a predetermined correlation matrix on the risks \( Y_k \). We have

\[
\text{cov}(e^{X_i}, e^{X_j}) = E \left[ e^{X_i + X_j} \right] - E \left[ e^{X_i} \right] E \left[ e^{X_j} \right]
\]

\[
= \exp \left\{ -\left( \tau_i^2 + \tau_j^2 \right)/2 + (\tau_i^2 + \tau_j^2 + 2\Sigma_{ij} \tau_i \tau_j) \right\} - 1
\]

\[= \exp \left\{ \tau_i \tau_j \Sigma_{ij} \right\} - 1 \quad (E.6)
\]
where we have used $X_i + X_j \sim N\left(-\left(\tau_i^2 + \tau_j^2\right)/2, \tau_i^2 + \tau_j^2 + 2\Sigma_{ij}\tau_i\tau_j\right)$. We also have

$$\text{cov}(Y_i, Y_j) = \tilde{\sigma}_i \tilde{\sigma}_j \text{cov}(e^{X_i}, e^{X_j}).$$

(E.7)

Using (E.6) and (E.7) we may then solve for $\Sigma_{ij}$:

$$\text{cov}(Y_i, Y_j) = \tilde{\sigma}_i \tilde{\sigma}_j (\exp \{\tau_i \tau_j \Sigma_{ij}\} - 1)
\Leftrightarrow \Sigma_{ij} = \ln \left(\frac{\text{cov}(Y_i, Y_j)}{\tilde{\sigma}_i \tilde{\sigma}_j} + 1\right) / \tau_i \tau_j$$

We also have $\frac{\text{cov}(Y_i, Y_j)}{\tilde{\sigma}_i \tilde{\sigma}_j} = \text{cor}(Y_i, Y_j) \sqrt{\left(e^{\tau_i^2} - 1\right)\left(e^{\tau_j^2} - 1\right)}$ meaning

$$\Sigma_{ij} = \frac{\ln \left(\text{cor}(Y_i, Y_j) \sqrt{\left(e^{\tau_i^2} - 1\right)\left(e^{\tau_j^2} - 1\right)} + 1\right)}{\tau_i \tau_j}.$$  

(E.8)

With this equation and a vector $\tau$ we may obtain $\Sigma$ and continue the calculations in $\mathbb{R}$ with $\Sigma$. However one has to be careful here. In some cases the entries in the the Solvency II correlation matrices $\text{cor}(Y_i, Y_j)$ are negative meaning that one has to make sure to choose $\tau$ such that

$$\text{cor}(Y_i, Y_j) \sqrt{\left(e^{\tau_i^2} - 1\right)\left(e^{\tau_j^2} - 1\right)} + 1 > 0.$$  

Another problem which is mentioned on page 9 is that (E.8) may not necessarily produce positive-definite matrices for all choices of $\tau$. (E.8) also allow us to go from $\Sigma$ to $\text{cor}(Y_i, Y_j)$.

### F Solvency II correlation matrices

Here we explicitly state the correlation matrices used in the numerical experiments on page 9. They can all be found in EIOPA (2014)
Table 1: Solvency II correlation matrices used in this thesis. $K = 5$, $K = 6$, $K = 7$ and $K = 12$. 

| Table 1: Solvency II correlation matrices used in this thesis. K = 5, K = 6, K = 7 and K = 12. |
R-code

The code used to produce the figures in the thesis will be presented below along with some short comments where found necessary. Note that, aside from the figures plotted, the programs will often output some form of progress indication when running the program. This is solely done as a way to estimate time remaining and check that the program has not encountered any problems. Aside from this, no output will be sent to the console.

The following R-packages are needed:

```r
require(iterpc)
require(MASS)
```

Function library

In this section a full list of user-defined functions used in the programs will be presented.

Gauss-Hermite weights and abscissas - gh

This function computes the weights \( w \) and abscissas \( x \) for the Gauss-Hermite expansion of order \( n \). The function uses results from Press et al. (2007).

```r
gh = function(n){
  w <- x <- rep(NA, n)
  m = floor((n+1)/2)
  iter_cap = 100
  tol = 1E-14

  for (i in 1:m){
    # Initial guess for root
    if (i == 1){
      z = sqrt(2*n+1) - 1.85575*(2*n+1)^(-1/6)
    } else if (i == 2){
      z = z - 1.14*n^0.426/z
    } else if (i == 3){
      z = 1.86*z - 0.86*x[1]
    } else if (i == 4){
      z = 1.91*z - 0.91*x[2]
    } else {
      z = 2*z - x[i-2]
    }

    for (its in 1:iter_cap){
      # Calculating the Hermite polynomial
      p1 = 1/pi^0.25
      p2 = 0
      for (j in 1:n){
        p3 = p2
        p2 = p1
        p1 = z * sqrt(2/(j))*p2 - sqrt((j-1)/(j))*p3
      }
      # Newtons Method
  }
}
```
\[ p_p = \sqrt{2j \cdot p^2} \]
\[ z_1 = z \]
\[ z = z_1 - \frac{p_1}{p_p} \]

```r
# Is convergence reached?
if (abs(z-z1) <= tol) {break}
```

```r
x[i] = z
x[n-i+1] = -z  # symmetrical abscissas
w[i] = \( \frac{2}{pp^2} \)
w[n-i+1] = w[i]  # symmetrical weights
```

```r
list(x = x, w = w)
```

---

**The 1-dimensional MGF - \( \text{mgf}_{1d \cdot N} \)**

Takes input \( t, N, \sigma_S \) and \( \tau_S \) and calculates (10).

```r
\text{mgf}_{1d \cdot N} = \text{function}(t, N, \sigma_S, \tau_S)\{ 
\quad \sigma_tilde = \sigma_S / \sqrt{\exp(\tau_S^2) - 1} 
\quad \text{tmp} = \text{gh}(N) 
\quad w = \text{tmp}$w 
\quad x = \text{tmp}$x 
\quad \text{f} = \exp(-t \cdot \sigma_tilde * 
\quad (\exp(\sqrt{2} \cdot \tau_S \cdot x - \tau_S^2/2) - 1)) / \sqrt{\pi} 
\quad \text{integral} = \text{sum}(w \cdot f) 
\} 
```

---

**Solving the problem using the Cholesky decomposition**

**Solving** \( \Psi_{\sum Y_k}(t; \sigma, \tau, C) = \Psi(t; \sigma_S, \tau_S) \) **with respect to** \( t - \text{mgf}\_\text{solve} \)

This function takes in all parameters in the model and solves equation (12) using \texttt{uniroot}, returning \( \sigma_S \) and \( \tau_S \).

```r
\text{mgf}\_\text{solve} = \text{function}(t, \text{par}\_\text{input})\{ 
\quad \text{par} = \text{par}\_\text{input} 
\quad \text{par}\_$\sigma_tilde = \text{par}\_\text{input}\_$\sigma / \sqrt{\exp(\text{par}\_\text{input}\_$\tau^2) - 1} 
\quad \text{par}\_$K = \text{length}(\text{par}\_\text{input}\_$\tau) 
\quad \text{par}\_$t = t 
\quad \text{tau.m} = \text{matrix}\( \text{par}\_$\tau, \text{nrow}=\text{par}\_$K, \text{ncol}=\text{par}\_$K) 
\quad \text{if} (!\text{is.null(\text{par}\_\text{input}\_$C))) \{ 
```
# If C is supplied use it
par$C = par_input$C

else {
  # Else calculate C from supplied SIGMA
  SIGMA = par_input$SIGMA
  par$C = tau.m * t(tau.m) * par_input$SIGMA
}

# Calculate sigma_S
V = par$C * NA
for (i in 1:par$K) { for (j in 1:par$K) {
  w = exp(-(par$tau[i]^2 + par$tau[j]^2)/2 +
          (par$C[i,i]+par$C[j,j])/2)*(exp(par$C[i,j])−1)
  V[i,j] = par$sigma_tilde[i]*par$sigma_tilde[j]*w
}
par$sigma_S = sqrt(sum(V))
par$mgf_Y_vector = mgf(par$t, par$K, par$N, par$C, par$sigma, par$tau)

d = uniroot(solvable, interval=c(0.1, 10),
            extendInt="yes", maxiter = 1000, par = par)
  # return sigma_S and tau_S
  list(sigma_S=par$sigma_S, tau_S=d$root)
}

---

**Help-function of mgf_solve - solvable**

This function takes in model parameters and the value of \( \Psi_{\sum \gamma_k}(t; \sigma, \tau, C) \) (see (11), computes \( \Psi(t; \sigma_S, \tau_S) \) (see (10) and returns \( \Psi_{\sum \gamma_k}(t; \sigma, \tau, C) - \Psi(t; \sigma_S, \tau_S) \). This function is fed into uniroot in mgf_solve.

```r
solvable = function(tau_S, par){
  N = par$N; sigma_S = par$sigma_S; t = par$t
  K = par$K; C = par$C; sigma = par$sigma; tau = par$tau
  mgf_Y_S = mgf_1d_N(t, N, sigma_S, tau_S)
  return(mgf_Y_S - par$mgf_Y_vector)
}
```

---

**The K-dimensional MGF: Cholesky - mgf**

The function that handles the computation of the Gauss-Hermite expansion approximation of the K-dimensional MGF - expression (11). Note that in this expression we have \( K \) sums where each index \( n_k \) goes from 1 to \( N \). This means that for some combinations of \( N \) and \( K \) storing all of permutations of \( (n_k)_k=1 \) at once might not be feasible. We therefore create an iterpc object (using the iterpc-package) which lets us extract only a subset of the different permutations of the indexes at a time.

```r
mgf = function(t, K, N, C, sigma, tau){
```
tmp = g h (N)
w = tmp$w
a = tmp$x
sigma_tilde = sigma / sqrt(exp(tau^2) - 1)
L = t(chol(C)) # C = L * t(L)

sum_indexes = iterpc(N, K, labels=NULL, ordered=T, replace=T)
n = getnext(sum_indexes, 1e5)
S = 0
while (!(is.null(n))){
    for (i in 1:nrow(n)){
        S = S + calcer(n[i,], w, a, L, t, sigma_tilde, N, tau)
    }
    n = getnext(sum_indexes, 1e5)
}
S

Help-function of mgf - calcer

This function is called upon by mgf to calculate
\[
\frac{w_{n_1} \cdots w_{n_K}}{\pi^{K/2}} \prod_{k=1}^{K} \exp \left\{ -t\tilde{\sigma}_k \left[ \exp \left( \sqrt{2} \sum_{j=1}^{K} l_{kj}a_{n_j} - \frac{\tau_k^2}{2} \right) - 1 \right] \right\}
\]
in (11) given sum-indexes \((n_k)\).

calcer = function(n, w, a, L, t, sigma_tilde, N, tau){
    K = length(n)
    n_matrix = matrix(a[n], ncol=K, nrow=K, byrow=T)
    sums = rowSums(L*n_matrix)
    tmp = exp(-t*sigma_tilde*(exp(sqrt(2)*sums-tau^2/2)-1))
    prod(w[n]) * prod(tmp)/(sqrt(pi)^K)
}

Solving the problem using the Spectral decomposition

Solving \(\hat{\Psi}_{\Sigma} Y_k(t; \sigma, \tau, \Sigma) = \hat{\Psi}(t; \sigma_S, \tau_S)\) with respect to \(t\) - mgf_pc_solve

Analogous to mgf_solve. This function takes in all parameters in the model and solves
the problem with the Spectral decomposition using uniroot, returning \(\sigma_S, \tau_S\), the final
value of solvable_pc and the number of uniroot iterations. Handles both \(L = K\) and
\(L < K\).

mgf_pc_solve = function(t, par_input){
    par = par_input
    par$sigma_tilde = par_input$sigma/sqrt(exp(par_input$tau^2) - 1)
    par$K = length(par_input$tau)
    par$t = t
if (is.null(par_input$SIGMA)) {
    par$SIGMA = cor_Y2Sigma(par$V, par$tau, par$sigma)
}

# Calculate sigma_S
V = par$SIGMA*NA
for (i in 1:par$K) {
    for (j in 1:par$K) {
        w = exp(-((par$tau[i]^2 + par$tau[j]^2))/2 +
            (par$tau[i]^2*par$SIGMA[i,i] +
            par$tau[j]^2*par$SIGMA[j,j])/2)*
            (exp(par$tau[i]*par$tau[j]*par$SIGMA[i,j]) - 1)
        V[i,j] = par$sigma_tilde[i]*par$sigma_tilde[j]*w
    }
}
par$sigma_S = sqrt(sum(V))

par$mgf_Y_vector = mgf_pc(par$t, par$K, par$N,
    par$SIGMA, par$sigma, par$tau, par$L)

d = uniroot(solvable_pc, interval = c(1e-6, 10),
    extendInt = "yes", maxiter = 1000, par = par)
list(sigma_S = par$sigma_S, tau_S = d$root,
    solve.at.root = d$f.root, iter = d$iter)

---

**Help-function of mgf_solve - solvable_pc**

Analogous to solvable. Handles both L = K and L < K.

```r
solvable_pc = function(tau_S, par){
    N = par$N; sigma_S = par$sigma_S; t = par$t
    K = par$K; L = par$L; sigma = par$sigma; tau = par$tau
    mgf_Y_S = mgf_lid_N(t, N, sigma_S, tau_S)
    return(mgf_Y_S - par$mgf_Y_vector)
}
```

---

**The K-dimensional MGF: Cholesky - mgf_pc**

Analogous to mgf. Handles both L = K and L < K.

```r
mgf_pc = function(t, K, N, Sigma, sigma, tau, L){
    if (is.null(L)) {
        L = K-1
    }
    tmp = gh(N)
    w = tmp$w
    a = tmp$x
    sigma_tilde = sigma/sqrt(exp(tau^2) - 1)
    d = eigen(Sigma)
```
lambda = d$values
Gamma = d$ vectors

sum_indexes = iterpc(N, L, labels= NULL, ordered=T, replace=T)
n = getnext(sum_indexes, 1e5)
S = 0
while (!is.null(n)){
    for (i in 1:nrow(n)){
        S = S + calcer_pc(n[i,], w, a,
        lambda, Gamma, t, sigma_tilde, tau)
    }
n = getnext(sum_indexes, 1e5)
} S

Help-function of mgf - calcer_pc

Analogous to calcer. Calculates
\[
\frac{w_{n_1} \cdots w_{n_K}}{\pi^{K/2}} \prod_{k=1}^{K} \exp \left\{ -t\tilde{\sigma}_k \left( \exp \left[ -\tau_k^2/2 + \tau_k \sum_{j=1}^{K} \gamma_{kj} \sqrt{2\lambda_j a_{n_j}} \right] - 1 \right) \right\}
\]
in (14) or its reduced expression in (16) if a model with L < K is used.

calcer_pc = function(n, w, a, lambda, Gamma, t, sigma_tilde, tau){
    L = length(n)
    K = length(tau)

tmp.lambda = matrix(lambda[1:L], K, L, byrow=T)
tmp.a_n = matrix(a[n], K, L, byrow=T)
tmp.sums = rowSums(Gamma[,1:L]*sqrt(2*tmp.lambda)*tmp.a_n)
tmp = -t*sigma_tilde*(exp(-tau^2/2+tau*tmp.sums)-1)
tmp = exp(tmp)
prod(w[n])*prod(tmp)/(sqrt(pi)\^L)
}

Going from \(\text{cor}(\epsilon_i, \epsilon_j)\) to \(\text{cor}(Y_i, Y_j)\) or vice versa - Sigma2cor_Y and cor_Y2Sigma

Using result (E.8).
cor_Y2Sigma = function(cor_Y, tau, sigma){
    K = length(tau)
    tau1 = sqrt(matrix(exp(tau^2)-1, K,K))
tau1_prod = tau1 * t(tau1)
tau2 = matrix(tau, K, K)
tau2_prod = tau2 * t(tau2)
cor_eps = log(cor_Y*tau1_prod+1)/tau2_prod
cor_eps
}

\[
\text{Sigma2cor}_\text{Y} = \text{function}(\text{Sigma}, \tau, \text{sigma})\
\begin{align*}
\tau.m &= \text{matrix}(\tau, \text{length}(\tau), \text{length}(\tau)) \\
\tau.m2 &= t(\tau.m) \\
\text{cov}_\text{exp} &= \exp(\text{Sigma}\times\tau.m\times\tau.m2) - 1 \\
\text{cor}_\text{Y} &= \frac{\text{cov}_\text{exp}}{\sqrt{\left(\exp(\tau.m^2) - 1\right)\left(\exp(\tau.m2^2) - 1\right)}}
\end{align*}
\]

\[
\text{cor}_\text{Y}
\]

**Computing the 99.5th percentile of } Y_S \text{ given } (\sigma_S, \tau_S) - \text{tau2percentile**}

\[
\text{tau2percentile} = \text{function}(\tau, \text{sigma}, \text{perc}=99.5/100)\
\begin{align*}
q &= q\text{norm(perc)} \\
\text{sigma/sqrt(exp(}\tau^2\text{) - 1)\times(exp(-}\tau^2/2+\tau\times q) - 1)
\end{align*}
\]

**Simulating } Y_S \text{ m times with Monte Carlo - MC_sim**

\[
\text{MC_sim} = \text{function}(\tau, \text{sigma}, \text{Sigma}, m=10^7)\
\begin{align*}
n &= \text{length}(\text{sigma}) \\
\text{eps} &= \text{mvrmvnorm}(m, \text{rep}(0, n), \text{Sigma} = \text{Sigma}) \\
\text{sigma_tilde} &= \text{sigma/sqrt(exp(}\tau^2\text{) - 1)} \\
\text{X} &\leftarrow Y \leftarrow \text{eps*NA} \\
\text{for (k in 1:n)}\
\begin{align*}
\text{X}[i,k] &= -\tau[i]^2/2+\tau[i]\times\text{eps}[i,k] \\
\text{Y}[i,k] &= \text{sigma_tilde}[k]\times(\exp(\text{X}[i,k]) - 1)
\end{align*}
\end{align*}
\]

\[
\text{Y}_S = \text{rowSums(Y)}
\]

**Completing a matrix - complete**

This function takes a matrix and makes it symmetrical by setting the upper triangular equal to the lower triangular. Used when only the lower triangular of the Solvency II matrices were written in the code to minimize the possible errors when typing them in.

\[
\text{complete} = \text{function}(\text{cor}_\text{Y})\
\begin{align*}
\text{V} &= \text{cor}_\text{Y} \\
\text{V[upper.tri(V)]} &= \text{NA} \\
\text{for (i in 1:nrow(\text{cor}_\text{Y}))}\{\text{for (j in 1:ncol(\text{cor}_\text{Y}))}\{ \\
\text{if (is.na(V[i,j])}\{V[i,j] = \text{cor}_\text{Y}[j,i]\}
\}
\}\}}
\end{align*}
\]

\[
\text{return(V)}
\]

31
Computing the SCR using the Standard formula - standard_formula

```r
standard_formula=function (cor_Y, sigma) {
  C = matrix(sigma*qnorm(0.995), length(sigma), length(sigma))
  C_S = sqrt(sum(cor_Y*C*t(C)))
  C_S
}
```

Computing the SCR using the Bølviken and Guillen (2017) - bolvguil

```r
bolvguil = function(tau, sigma, Sigma, gamma = NULL) {
  n = length(sigma)

  # if skewness is supplied, calculate tau based on this
  if (!is.null(gamma)) {
    A = (1+gamma^2/2-sqrt(gamma^4/4+gamma^2))^(1/3)
    tau = sqrt(log(A+1/A-1))
  }

  # sigma_S
  sigma_S = 0
  for (i in 1:n){ for (j in 1:n){
    sigma_S = sigma_S + Sigma[i,j]*sigma[i]*sigma[j]
  }}
  sigma_S = sqrt(sigma_S)

  # gamma_S
  alpha = sigma/(sigma_S*sqrt(exp(tau^2)-1))
  beta = matrix(NA, n, n)
  for (i in 1:n){ for (j in 1:n){
    beta[i,j] = 1 + Sigma[i,j]*sqrt((exp(tau[i]^2)-1)*
      (exp(tau[j]^2)-1))
  }}

  sum_1 <- sum_2 <- sum_3 <- 0
  for (i in 1:n){
    sum_1 = sum_1 + alpha[i]^3*(exp(tau[i]^2)+2)*
      (exp(tau[i]^2)-1)^2
    for (j in 1:n){
      if (i != j){
        sum_2 = sum_2 +
          alpha[i]^2*alpha[j]*(exp(tau[i]^2)*
            (beta[i,j]^2-1) - 2*(beta[i,j]-1))
      }
  }
  sum_3 = sum_1 + sum_2
  C_S = sqrt(sum_3)
  C_S
}
```
for (k in 1:n)
  if ( (i<j) & (j<k) ){
    sum_3 = sum_3 + alpha[i] * alpha[j] * alpha[k] *
    (beta[i,j]*beta[i,k]*beta[j,k] -
    beta[i,j]-beta[i,k]-beta[j,k] + 2)
  }
}

gamma_S = sum_1 + 3*sum_2 + 6*sum_3

# tau_S
A_S = (1+gamma_S^2/2-sqrt(gamma_S^4/4+gamma_S^2))^(1/3)
tau_S = sqrt(log(A_S+1/A_S-1))

SCR = sigma_S/sqrt(exp(tau_S^2)-1)*
    (exp(-tau_S^2/2+tau_S*qnorm(99.5/100))-1)

list(tau_S=tau_S,
     sigma_S=sigma_S,
     gamma_S=gamma_S,
     SCR=SCR)

All-in-one: Cholesky, Spectral and MC solution with optimized t - optimat_t2

This function starts by obtaining the 99.5th percentile of ∑ Y_k by use of Monte Carlo. It will then find the t-values for which the 99.5th percentile in the Cholesky- and Spectral-approach obtains the smallest distance to the Monte Carlo solution. The function returns the MC SCR as well as optimized t-values with accompanying SCR. It will also optimize a t-value for a Spectral-model with reduced L if the argument low_L is supplied. This function is only used in program 2b.

optimal_t2 = function (par_input, low_L = NULL){
  # MC-simming the solution
  MC_sol = quantile(MC_sim(par_input$tau, par_input$sigma,
                  par_input$SIGMA, m=10^-7), probs=99.5/100)

  # minimizing error with respect to t
  chol_opt = optimize(optimal_t2_chol, interval=c(0.01, 0.15),
                      par_input=par_input, MC_sol=MC_sol)
  pc_opt = optimize(optimal_t2_pc, interval=c(0.01, 0.15),
                   par_input=par_input, MC_sol=MC_sol)

  chol_min = chol_opt$minimum
  chol_best_acc = chol_opt$objective

  pc_min = pc_opt$minimum
\[
\text{result} = \text{list}(\text{chol}_{\text{min}} = \text{chol}_{\text{min}},
\text{chol}_{\text{best}} = \text{chol}_{\text{best}},
\text{pc}_{\text{min}} = \text{pc}_{\text{min}},
\text{pc}_{\text{best}} = \text{pc}_{\text{best}},
\text{MC}_{\text{sol}} = \text{MC}_{\text{sol}})
\]

if (!\text{is.null(low}_{L})\}){
\text{par}_{\text{input}}_L = \text{par}_{\text{input}}
\text{par}_{\text{input}}_L$L = low$_L
\text{pc}_{\text{opt}}_L = \text{optimize}(\text{optimal}_t2_{\text{pc}}, \text{interval} = c(0.01, 0.15),
\text{par}_{\text{input}} = \text{par}_{\text{input}}_L, \text{MC}_{\text{sol}} = \text{MC}_{\text{sol}})
\text{result}$_{\text{pc}_{\text{min}}}_L = \text{pc}_{\text{opt}}_L$_{\text{minimum}}
\text{result}$_{\text{pc}_{\text{best}}}_{\text{acc}}_L = \text{pc}_{\text{opt}}_L$_{\text{objective}}
\}
\}
\text{result}
\]

Help-functions for \text{optimal}_t2 - \text{optimal}_t2_{\text{chol}} and \text{optimal}_t2_{\text{pc}}

These two functions are fed into \text{uniroot} in \text{optimal}_t2 to minimize absolute error with respect to \(t\) for the Cholesky-scheme and the Spectral-scheme.

\[
\text{optimal}_t2_{\text{chol}} = \text{function}(t, \text{par}_{\text{input}}, \text{MC}_{\text{sol}})\}{
\text{d} = \text{mgf}_{\text{solve}}(t, \text{par}_{\text{input}})
\text{chol}_{\text{solution}} = \text{tau2percentile}(d$_{\text{tau}}_S, d$_{\text{sigma}}_S)
\text{abs}(\text{MC}_{\text{sol}}-\text{chol}_{\text{solution}})
\}
\]

\[
\text{optimal}_t2_{\text{pc}} = \text{function}(t, \text{par}_{\text{input}}, \text{MC}_{\text{sol}})\}{
\text{d} = \text{mgf}_{\text{pc}_{\text{solve}}}(t, \text{par}_{\text{input}})
\text{pc}_{\text{solution}} = \text{tau2percentile}(d$_{\text{tau}}_S, d$_{\text{sigma}}_S)
\text{abs}(\text{MC}_{\text{sol}}-\text{pc}_{\text{solution}})
\}
\]

Program 1 - Density plots for different \(t\)

This program produces the two plots in Figure 2. It starts by simulating the sum \(\sum Y_k\) directly. Then for each \(t\) in \text{t.vector} it obtains \((\sigma_S, \tau_S)\) by Cholesky and simulates \(Y_S(\sigma_S, \tau_S)\) and for each sequence obtains the quantiles via the \text{quantile} function on line 23.

\[
\text{tau} = c(1, 0.5, 0.7, 1.2, 1.5)
\]

\[
\text{sigma} = \text{rep}(1, \text{length(\text{tau}))}
\]

\[
\text{cor}_Y = \text{complete(S2)}
\]

\[
\text{Sigma} = \text{cor}_Y2\text{Sigma}(\text{cor}_Y, \text{tau}, \text{sigma})
\]

\[
\text{par}_{\text{input}} = \text{list}(\text{tau} = \text{tau}, \text{sigma} = \text{sigma}, \text{SIGMA} = \text{Sigma}, N = 4)
\]

\[
\text{t.vector} = c(0.02, 0.1, 0.5, 5)
\]
7 tau_S.vector <- sigma_S.vector <- t.vector*NA
8 for (i in 1:length(t.vector)){
9   t = t.vector[i]
10   d = mgf_solve(t, par_input)
11   (tau_S.vector[i] = d$tau_S)
12   (sigma_S.vector[i] = d$sigma_S)
13 }

m = 10^-7
15 p = ppoints(1000)
16 Y_S = MC_sim(m, tau, sigma, Sigma)
17 exact = quantile(Y_S, p=p)
18 CDFs = matrix(NA, length(p), length(t.vector))
19 for (i in 1:length(t.vector)){
20   Y_S = MC_sim(m, tau_S.vector[i], sigma_S.vector[i], Sigma = 1)
21   CDFs[,i] = quantile(Y_S, p=p)
22 }

# PLOTTING CDFS
26 lines = c(1, rep(2, ncol(CDFs)))
27 legend_lab = rep("Exact", ncol(CDFs)+1)
28 for (n in 2:length(legend_lab)){
29   legend_lab[n] = sprintf("t=%g", t.vector[n-1])
30 }
31 matplot(cbind(exact, CDFs), 1-p, type="l", lwd = 2, log="y",
32   col = c(1:4,6), lty = lines, yaxt="n", ylab = "",
33   xlab = "Sum of log-normals")
34 axis(2, at=10^((4):0), labels=c("10^(-4)",
35   "10^(-3)", "10^(-2)", "10^(-1)", "10^0"))
36 legend("bottomleft", legend = legend_lab, col = c(1:4,6),
37   lty = lines, lwd = 2)
39
# PLOTTING CDFs
41 matplot(cbind(exact, CDFs), p, type="l", lwd = 3, log="y",
42   col = c(1:4,6), lty = lines, xlim = c(-4, 2),
43   xaxt="n", ylab = "CDF",
44   xlab = "Sum of log-normals", xaxt="n")

Program 2a - Accuracy in the \( K = 6 \) problem

This program produces Figure 3a. It makes heavy use of \texttt{optimal_t2} which is documented on page F.

1 K = dim(S22)[1]
2 big_sigma = 4
3 small_sigma = 1
4 big_tau = 1.2
small_t = 0.25

std_formula <- chol_static <- pc_static <- chol_best <- 1:(K+1)*NA
pc_best <- pc_best_L <- chol_best_t <- pc_best_t <- 1:(K+1)*NA
bg <- pc_static_L <- pc_best_t_L <- pc_static

chol_static_t <- pc_static_t <- 0.1
pc_static_t_L <- 0.1

for (i in 1:(K+1)){
  print(sprintf("%d of %d", i, K+1))
  sigma = c(rep(big_sigma, i-1), rep(small_sigma, K-i+1))
  tau = c(rep(big_t, i-1), rep(small_t, K-i+1))
  Sigma = complete(S22)
  cor_Y = Sigma2cor_Y(Sigma, tau, sigma)
  par_input = list(tau=tau, SIGMA=Sigma, sigma=sigma, N=4, L=K)

  # Optimizing t and solving the problem
  d = optimal_t2(par_input, low_L = K-1)

  chol_best[i] = d$chol_best_acc
  pc_best[i] = d$pc_best_acc
  pc_best_L[i] = d$pc_best_acc_L
  chol_best_t[i] = d$chol_min
  pc_best_t[i] = d$pc_min
  pc_best_t_L[i] = d$pc_min_L
  MC_sol = d$MC_sol

  chol_solved = mgf_solve(chol_static_t, par_input)
  chol_static[i] = abs(MC_sol-
                      tau2percentile(chol_solved$tau_S, chol_solved$sigma_S))

  pc_solved = mgf_pc_solve(pc_static_t, par_input)
  pc_static[i] = abs(MC_sol-
                      tau2percentile(pc_solved$tau_S, pc_solved$sigma_S))

  pc_solved_L = mgf_pc_solve(pc_static_t_L, par_input_L)
  pc_static_L[i] = abs(MC_sol-
                      tau2percentile(pc_solved_L$tau_S, pc_solved_L$sigma_S))

  bg[i] = abs(MC_sol-
              bolvgui(par_input$tau, par_input$sigma,
                      par_input$SIGMA)$SCR)
  std_formula[i] = abs(MC_sol-
                      standard_formula(cor_Y, sigma))
# The plot

```r
result = cbind(chol_best, chol_static, pc_best,
                pc_static, bg, STD_formula)
matplot(0:K, result, type="b", lty=1, lwd=2, pch=16, log="y",
yaxt="n", ylab="Absolute error",
xlab = "Number of large risks",
ylim = 10^(c(-2,2)), cex.lab=1.8, cex.axis=1.5, cex.sub=2)
axis(2, at=10^(c(-3,-2,-1,0,1)), labels=c("10^(-3)","10^(-2)",
"10^(-1)", "10^0", "10^1"), cex.axis=1.2)
legend("bottomright", c("Cholesky (optimized t)",
"Cholesky, t=0.1", "PC (optimized t)", "PC, t=0.1",
"Bolviken-Guillen", "Std formula"), lty=1, lwd=2, col=1:6)
```

---

**Program 2b - Accuracy in the $K = 12$ problem**

This program produces Figure 3b. Since no optimization with respect to $t$ is done, `optimal_t2` is not used here. Instead this is straight up application of the functions `mgf_solve` and `mgf_pc_solve`.

```r
K = dim(S22)[1]
big_sigma = 4
small_sigma = 1
big_tau = 1.2
small_tau = 0.25

STD_formula <- chol_static <- pc_static <- 1:(K+1)*NA
bg <- pc_static_L <- 1:(K+1)*NA
chol_static_t <- 0.08
pc_static_t <- 0.08
pc_static_t_L <- 0.05

for (i in 1:(K+1)){
  print(sprintf("%d of %d", i, K+1))
  sigma = c(rep(big_sigma, i-1),rep(small_sigma, K-i+1))
  tau = c(rep(big_tau, i-1),rep(small_tau, K-i+1))
  Sigma = complete(S2222)
  par_input = list(tau=tau, SIGMA=Sigma,
                   sigma=sigma, N=3, L=K)
  par_input_L = list(tau=tau, SIGMA=Sigma,
                   sigma=sigma, N=3, L=K-1)
  sims = MC_sim(10^7, tau, sigma, Sigma)
  MC_sol = quantile(sims, probs=99.5/100)
  chol_solved = mgf_solve(chol_static_t, par_input)
```
chol_static[i] = abs(MC_sol -
    tau2percentile(chol_solved$tau_S, chol_solved$sigma_S))

pc_solved = mgf_pc_solve(pc_static_t, par_input)

pc_static[i] = abs(MC_sol -
    tau2percentile(pc_solved$tau_S, pc_solved$sigma_S))

pc_solved_L = mgf_pc_solve(pc_static_t_L, par_input_L)

pc_static_L[i] = abs(MC_sol -
    tau2percentile(pc_solved_L$tau_S, pc_solved_L$sigma_S))

bg[i] = abs(MC_sol - bolvguil(par_input$tau,
    par_input$sigma, par_input$SIGMA)$SCR)

STD_formula[i] = abs(MC_sol -
    standard_formula(cor_Y, sigma))

}

# Plotting
result = cbind(chol_static, pc_static, bg,
    STD_formula, pc_static_L)
matplot(0:K, result, type="b", lty=c(rep(1,4),2), lwd=2, pch=16,
    log="y", col=c(2,4,5,6,4), yaxt="n",
    ylab="Absolute error",
    xlab = "Number of large risks", ylim=10^(-3),
    cex.lab=1.8, cex.axis=1.5, cex.sub=2)
axis(2, at=10^(-3,-2,-1,0,1), labels=c("10^-3","10^-2",
    "10^-1"), cex.axis=1.2)
legend("bottom", c("Cholesky, t = 0.08", "PC (L=K), t = 0.08",
    "Bolviken-Guillen", "Std formula", "PC (L=K-1), t = 0.05"),
    lty=c(rep(1,4),2), lwd=2, col=c(2,4,5,6,4))

---

Program 3 - Testing \( t \)-values for the \( K = 12 \) problem

This program produces Figure 4. Change the value of \( \text{par}_\text{input}$\text{N} \) on line 11 according to whether you want Figure 4a (\( N = 3 \)) or 4b (\( N = 4 \)). Note that in the case of \( N = 4 \) the program might take several hours to complete.

Sigma = complete(S2222)  # SII cor-matrix
L.v = c(0,10,11,12)
t.v = c(0.01, 0.025, 0.05, 0.1, 0.25, 0.5, 1, 2, 3)
big_sigma = 4
small_sigma = 1
big_tau = 1.2
small_tau = 0.25
sigma = c(rep(big_sigma, 6),rep(small_sigma, 6))
tau = c(rep(big_tau, 6),rep(small_tau, 6))
Sigma = complete(S2222)
par_input = list(tau=tau, SIGMA=Sigma, sigma= sigma, N=4)
time = matrix(NA, nrow=length(t.v), ncol=length(L.v))

# Assigning column names and row names of 
# the matrices storing the results
rnames = t.v*NA; for (n in 1:length(t.v)){
    rnames[n]=sprintf("t=%g", t.v[n])
}
cnames = L.v*NA; for (n in 1:length(L.v)){
    cnames[n]=sprintf("L=%g", L.v[n])
}
cnames[1] = "Cholesky"
rownames(time) = rnames
colnames(time) = cnames

tau_S.values <- sigma_S.values <- time

for (l in 1:length(L.v)){
    par_input$L = L.v[l]
    for (k in 1:length(t.v)){
        tryCatch({
            ptm <- proc.time()
            if (L.v[l] == 0){
                tmp = mgf_solve(t.v[k], par_input)
            } else {
                tmp = mgf_pc_solve(t.v[k], par_input)
            }
            d = proc.time() - ptm
            time[K, L] = d[1]
            tau_S.values[K, L] = tmp$tau_S
            sigma_S.values[K, L] = tmp$sigma_S
            if (L.v[l] == 0){
                print(sprintf("t=%g, L=%s --> Time: %g", t.v[k],
                            "Cholesky", d[1]))
            } else {
                print(sprintf("t=%g, L=%d --> Time: %g", t.v[k],
                            L.v[l], d[1]))
            }
        }, error=function(e){
            # Assume errors are due to univar not finding a solution, 
            # do nothing.
        })
    }
}
percentiles = tau2percentile(tau_S.values, sigma_S.values)

# MC simulation
MC_solution = quantile(MC_sim(10^7, tau, sigma, Sigma),
probs=99.5/100)

# Plotting
colors = colors()[c(24, 26, 33,
41, 139, 62, 473, 142)][c(1, 5:8)]
matplot(1:length(t.v), percentiles, type="b", lty=1, lwd=2,
pch=16, ylim=c(73, 83), xlab="t value",
ylab="99.5th percentile", xaxt="n", col=colors,
cex.lab=1.5, cex.axis=1.2)
axis(1, at=1:length(t.v), labels=t.v, cex.axis=1.2)
abline(h=MC_solution, lty=2, lwd=2) # Add MC solution
legend("topright", legend=c(names, "MC"),
  lty=c(rep(1,4), 2), lwd=2, pch=c(rep(16,4),NA),
col=c(colors[1:length(names)], 1))

Program 4 - Testing t-values for the K = 7 problem

This program produces Figure 5a and 5b.

Sigma = complete(S222) # SII cor-matrix
L.v = c(0, 6, 7)
t.v = c(0.01, 0.025, 0.05, 0.1, 0.25, 0.5, 1, 2, 3)
big_sigma = 4
small_sigma = 1
big_tau = 1.2
small_tau = 0.25
sigma_balanced = c(rep(big_sigma, 3), rep(small_sigma, 4))
tau_balanced = c(rep(big_tau, 3), rep(small_tau, 4))
sigma_light = rep(small_sigma, 7)
tau_light = rep(small_tau, 7)
par_input_balanced = list(tau=tau_balanced, SIGMA=Sigma,
sigma=sigma_balanced, N=4)
par_input_light = list(tau=tau_light, SIGMA=Sigma,
sigma=sigma_light, N=4)
time = matrix(NA, nrow=length(t.v), ncol=2*length(L.v))

# Assigning column names and row names of
# the matrices storing the results
rnames = t.v*NA; for (n in 1:length(t.v)){
  rnames[n] = sprintf("t=%g", t.v[n])
}
cnames = L.v*NA; for (n in 1:length(L.v)){
    cnames[n]=sprintf("L-%g, balanced risks", L.v[n])
    cnames[n+length(L.v)]=sprintf("L-%g, light risks", L.v[n])
}
cnames[1] = "Cholesky, balanced risks"
cnames[1+length(L.v)] = "Cholesky, light risks"

rownames(time) = rnames
colnames(time) = cnames

tau_S.values <- sigma_S.values <- time

for (l in 1:length(L.v)){
    par_input_balanced$L <- par_input_light$L <- L.v[l]
    for (k in 1:length(t.v)){
        ptm <- proc.time()
        tryCatch({
            if (L.v[l] == 0){
                tmp_balanced = mgf_solve(t.v[k], par_input_balanced)
            } else {
                tmp_balanced = mgf_pc_solve(t.v[k], par_input_balanced)
            }
        }, error=function(e){
            # Assume errors are due to uniroot not finding a solution,
            # do nothing
        })
        tryCatch({
            if (L.v[l] == 0){
                tmp_light = mgf_solve(t.v[k], par_input_light)
            } else {
                tmp_light = mgf_pc_solve(t.v[k], par_input_light)
            }
        }, error=function(e){
            # Assume errors are due to uniroot not finding a solution,
            # do nothing
        })

        tau_S.values[k,l] = tmp_balanced$tau_S
        sigma_S.values[k,l] = tmp_balanced$sigma_S
    }
}

d = proc.time() - ptm
if (L.v[l] == 0){
    print(sprintf("t-%g, L-%s --> Time: %g", t.v[k],}
Program 5a - Computing time experiments for the $K = 12$ problem

This program produces Figure 6a.
```r
N.v = 2:10
N_max = c(4,4,5,6,8,10,10,4)
N_max = rev(N_max)
tau = c(rep(1.2,6), rep(0.25,6))
sigma = c(rep(4,6), rep(1,6))
Sigma = complete(S2222)
par_input = list(tau=tau, sigma=sigma, SIGMA=Sigma)
t = 1

# For each value of N
for (n in 1:length(N.v)) {
  N = N.v[n]
  # For each method
  for (l in 1:length(L.v)) {
    L = L.v[l]
    if (N <= N_max[l]) {
      par_input$L = L
      par_input$N = N
      tryCatch({
        ptm <- proc.time()
        if (L == 0) {
          tmp = mgf_solve(t, par_input)
        } else {
          tmp = mgf_pc_solve(t, par_input)
        }
        d = proc.time() - ptm
        time[n,l] = d[1]
        values.tau_S[n,l] = tmp$tau_S
        values.sigma_S[n,l] = tmp$sigma_S
        if (L == 0) {
          print(sprintf("N=%d, L=%s --> Time: %g", N,
                        "Cholesky", d[1]))
        } else {
          print(sprintf("N=%d, L=%d --> Time: %g", N, L, d[1]))
        }
      }, error=function(e){
        # Assume errors are due to uniroot not finding a solution,
        # do nothing
      })
    }
  }
}
```
# Adding names to columns and rows of matrices storing the results

cnames = L.v*NA
for (n in 2:length(L.v)){
    cnames[n]=sprintf("L=%g", L.v[n])
}
cnames[1] = "Cholesky"

rnames = N.v*NA
for (n in 1:length(N.v)){
    rnames[n]=sprintf("N=%d", N.v[n])
}

colnames(values.tau_S) < colnames(time) < cnames
rownames(values.tau_S) < rownames(time) < rnames

# Plotting

colors = colors()[c(24, 26, 33, 41, 139, 62, 473, 142)]
matplot(N.v, time, type="b", lty=1, lwd=2, pch=16, col=colors,
        main=sprintf("Time spent solving the problem,
                      K = 12, t = %g", t),
        xlab="N", ylab="time", yaxt="n", cex.lab = 2,
        cex.axis =1.4)
axis(2, at=c(0,1,2,3,5,7,9)*60,
     labels=c("0", "1min", "2min", "3min",
              "5min", "7min", "9min"), cex.axis=1.2)
legend("topright", legend=cnames, col=colors,
       lty=1, lwd=2, pch=16)

Program 5b - Computing time experiments for the $K = 7$ problem

This program produces Figure 6b.
time <- matrix(NA, ncol=length(L.v), nrow=length(N.v))

values.tau_S <- values.sigma_S <- time

# For each value of N
for (n in 1:length(N.v)){
  N = N.v[n]

  # For each method
  for (l in 1:length(L.v)){
    L = L.v[l]

    if (N <= N_max[l]){  
      par_input$L = L  
      par_input$N = N

      tryCATCH(
        ptm <- proc.time()
        if (L == 0){
          tmp = mgf_solve(t, par_input)
        } else {
          tmp = mgf_pc_solve(t, par_input)
        }
        d = proc.time() - ptm

        time[n,l] = d[1]
        values.tau_S[n,l] = tmp$tau_S  
        values.sigma_S[n,l] = tmp$sigma_S

        if (L == 0){
          print(sprintf("N=%d, L=%s -> Time: %g", N,  
                        "Cholesky", d[1]))
        } else {
          print(sprintf("N=%d, L=%d -> Time: %g", N, L, d[1]))
        }
      }, error=function(e){
        # Assume errors are due to uniroot not finding a solution,
        # do nothing
      })
    }
  }
}

# Adding names to columns and rows of
# matrices storing the results

cnames = L.v*NA
for (n in 2:length(L.v)) {
    cnames[n] = sprintf("L=%g", L.v[n])
}
cnames[1] = "Cholesky"

rnames = N.v*NA
for (n in 1:length(N.v)) {
    rnames[n] = sprintf("N=%d", N.v[n])
}

colnames(values.tau_S) <- colnames(time) <- cnames
rownames(values.tau_S) <- rownames(time) <- rnames

# Plotting
colors = colors()[c(24, 26, 33, 41, 139, 62, 473, 142)]
matplot(N.v, time, type="b", lty=1, lwd=2, pch=16, col=colors, main=sprintf("Time spent solving the problem, K = %d, t = %g", length(tau), t), xlab = "N", ylab="time (seconds)", cex.lab = 2, cex.axis = 1.4)
legend("topleft", legend=cnames, col=colors, lty=1, lwd=2, pch=16)