DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Simulation of Multivariate Distributions with Various Univariate Marginals and Copulas

Alec Constantin Gliga

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Simulation of Multivariate Distributions with Various Univariate Marginals and Copulas

Simulation von mehrdimensionalen Verteilungen mittels verschiedener eindimensionaler Randverteilungen und Copulas

Author:	Alec Constantin Gliga
Supervisor:	Univ.-Prof. Dr. Thomas Huckle
Advisor:	Dr. Georg Meyer (Aquantec AG)
	Dr. Tobias Neckel
Submission Date:	16.09.2019

I confirm that this bachelor's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 16.09.2019 Alec Constantin Gliga

Acknowledgments

Throughout the process of writing this thesis I have received a great deal of support from a number of people. First off, I would like to thank my supervisor Dr. T. Neckel for his guidance throughout the whole process of this thesis and his constant willingness to help me out regardless of the issue at hand.
Furthermore, I would like to express my gratitude to Dr. G. Meyer, who allowed me to write this thesis within his company and provided me with all the necessary resources to complete it. Among this resources was G. Straub, who probably spend the most time helping me with various problems, which he treated as if they were his own.
Last but not least, I would like to thank my family for supporting me throughout this period, as well as my friends who would always be there when some distraction was appropriate.

Abstract

This thesis reviews recent methods employed for the simulation of Elliptical and Archimedean copulas and provides guidance for the implementation and testing of such simulations. The ultimate purpose of copula simulations is the generation of samples from a multivariate (joint) distribution, through the Monte Carlo sampling method. Simulations of such multivariate distributions have proven to be useful, when the number of marginals contained in a multivariate distribution is high, and analytic approaches get highly expensive in terms of computational effort or do not even exist. A second important advantage of copulas is their ability to decouple the dependence structure and marginals from one another, such that those can be modeled separately. Copulas are prominently used in quantitative finance, but have found applications in various other fields such as climate and weather research, civil engineering and medical research.

Kurzfassung

Diese Arbeit betrachtet aktuelle Methoden zur Simulation von Archimedischen und Elliptischen Copulas und schlägt eine Möglichkeit vor um diese zu implementieren und zu testen. Das letzendliche Ziel von Copula Simulationen ist Stichproben von einer multivariaten Verteilung durch Monte-Carlo-Simulationen zu generieren. Diese Simulationen gelten als bewährte Modellierungsverfahren für Verteilungen mit einer hohen Anzahl an Dimensionen, da analytische Methoden hierfür rechnerisch zu aufwendig sind oder gar nicht existieren. Ein weiterer Vorteil von Copulas liegt in der Fähigkeit eine multivariate Verteilung in eine Copula und Randverteilungen zu zerlegen, sodass diese separat modelliert werden können. Copulas werden hauptsächlich in der quantitativen Finanzwirtschaft verwendet, aber auch in verschiedensten anderen Bereichen, wie Klima- und Wetterforschung, Bauingeneurwesen und Medizinforschung.

Contents

1. Introduction

Modeling multivariate distributions involving d random variables, with $d \geq 2$, is a challenging problem, since the model complexity increases with the number of dimensions d. A way of dealing with this issue, is to rely on Monte Carlo simulations instead of analytical solutions. With the help of these simulations it is possible to create very general models that have the capability to capture complex structures. However, in order to make use of this technique, an efficient simulation scheme is required. Such efficient schemes exist for the simulation of copulas, which constitute a way of modeling the dependence structure inherent to multivariate distributions. It is crucial to understand that copulas themselves are only modeling the dependence among the random variables of a multivariate distribution, without taking their marginal distributions into account. This gets obvious when looking at the marginals of a copula, which are all uniformly distributed on the interval $[0, 1]$. While this might look like an oversimplification, it actually constitutes the strongest argument for the usage of copulas, as it allows one to model the marginal distributions and the dependence structure separately. By exploiting the fact that one can transform uniformly distributed random numbers into any desired univarate probability distribution through the application of the generalised inverse of the cumulative distribution function, commonly denoted by F^{-1}, onto the uniforms of the copula, one is able to obtain samples from an arbitrary multivariate distribution, without any further constraints.
The eventual goal of this thesis is to provide the theoretical background alongside with technical guidance for simulations of d-dimensional multivariate distributions $F(x_1, ..., x_d)$ with a copula C and marginals $F_1, ..., F_d$, where $F(x_1, ..., x_d) = C(F_1(x_1), ..., F_d(x_d))$. This results in the following two-step approach to obtain one sample vector of F:

$$\begin{aligned} &\text{Set } (U_1, ..., U_d) = \text{sample from C} \\ &\text{Return } (F_1^{-1}(U_1), ..., F_d^{-1}(U_d)) \end{aligned}$$

Note that it has been proven by Sklar's theorem (see section 2.1) that any multivariate distribution (with continuous marginals) can be decomposed into a copula and univariate marginals. Hence, no simplifying assumptions, with regards to the model and subject being modeled, have to be made theoretically.
To illustrate the importance of correct modeling of multivariate distributions, we describe in the next section a well-known inappropriate use of the so-called Gaussian copula in finance - with devastating consequences.

The 2008 Financial Crisis and the Gaussian Copula

Felix Salmon's famous article "A recipe for disaster: The formula that killed Wall Street" from 2009 displayed rigorously how the financial crisis of 2008 came about and uncovered the foolish mistakes that led to the devastating event [1]. The recipe Salmon refers to metaphorically, is the Gaussian copula, which will be discussed in detail in chapter 2. The Gaussian copula was introduced to the world of finance by Chinese mathematician David X. Li in 2000 through his paper "On Default Correlation: A Copula Function Approach" [2]. Inspired by the way insurance companies modeled the risk of mortality with regards to life insurances, Li proposed a similar approach to model the correlation between defaults in the credit market [3]. Practitioners were very quick to adapt Li's proposal, as it allowed institutions to sell huge amounts of seemingly risk-free financial products, called CDOs (Collatarised Debt Obligations) to investors, generating a huge bubble fed by the then booming housing/mortgage market. These CDOs, which essentially are pools of debt, such as mortgages, are sold to investors who then get payed off with the mortgage/debt payments of the debt holder. Those products were so appealing to investors, because they allowed them to avoid taking on idiosyncratic risks of individual defaults on mortgages by diversifying through pools. CDOs and other related financial products existed long before the financial crisis. The only problem was that modeling such complex products and pricing them according to those models, was an impediment to their popularity... Until the Gaussian copula was introduced to the financial world and used to model CDOs [4]. As a consequence, huge amounts were issued and sold to private and institutional investors, who were looking to profit from the bullish housing market. The main difficulty of modeling a pool of debt or mortgages is that one has to capture the correlations between the individual mortgages in order to estimate the risk of default correctly. A CDO can be seen as a multivariate distribution, where the marginals consist of the individual default probabilities of the debt holders. The Gaussian copula offered a tool to model the dependencies among the individual marginals of a CDO, which F. Salmon describes as mathematically beautiful, simple, tractable, yet fatally flawed. Especially, because the Gaussian copula lacks what is called "heavy tails", which describes a strong correlation of the variables in the extremes. It therefore failed to capture the probability of many mortgages defaulting at once, which eventually was the case when the crisis came about and the model fell apart. Since then, other more sophisticated types of copulas have been employed to model correlations in the financial industry, some of which will be introduced and implemented as part of this thesis.

Outline of Thesis

This thesis is structured as follows: First, the mathematical theory behind the concept of copulas, specific types, their application, alongside with all the additional tools necessary for performing simulations will be given in chapter 2. This includes the simulation algorithms as well. After that chapter 3 will outline a possible approach to implement and test the simulations. Chapter 4 will be about the evaluation of the implementation. Finally, in chapter

5 we are going to highlight the main findings of this thesis and give an outline on possible points for further research.

2. Copula - Definitions and Sampling

The word "copula" comes from Latin and means as much as "link" or "tie" [5]. Copulas, which can be of various types depending on the properties of the phenomenon being modeled, determine how the dependence between univariate marginal distributions is structured. One of the biggest advantages of copulas is given by the fact that the marginal distributions and their dependence are modeled separately. With regards to the application of copulas in financial risk modeling this provides a significant benefit, as it enables model building through a so-called "bottom-up approach". In the field of financial risk modeling, this proves to be crucial, as most of the time the marginal behaviour of the different variables included in a model can be observed rather easily, compared to the dependency among them. As an example, consider a credit risk model: the underlying marginals consist of the numerous obligors who hold loans and will be able to pay them back (or not) with a certain probability, which can be estimated rather easily through fundamental information on the debt holder, as well as historical data. Meanwhile the dependence between those obligors is much harder to estimate, especially considering the high dimensionality involved in this problem, which becomes an issue when looking at institutions that issue large amounts of loans [6].
This section is meant to recapitulate the mathematical background on which the simulation of copulas builds. The notations, definitions and theorems used are usually taken from [7] and [6], if not indicated otherwise.

2.1. Dependence Modeling with Copulas

Having outlined the advantageous concept of modeling multivariate distributions in a dyadic way, by isolating the dependence structure and the marginals, it is appropriate to formally introduce the copula:

Definition 1 (Copula) *A function* $C : [0,1]^d \to [0,1]$ *is called copula if there is a random vector* $(U_1, ..., U_d)$ *such that each component* $U_k \sim U(0,1)$ *for all k = 1,...,d, is uniformly distributed and*

$$C(u_1, ..., u_d) = \mathbb{P}(U_1 \leq u_1, ..., U_d \leq u_d), \quad u_1, ..., u_d \in [0,1]. \tag{2.1}$$

The mathematical foundation for the application of copulas to multivariate distributions was provided by Sklar (1959) through his famous theorem, which goes as follows:

Theorem 1 (Sklar's Theorem) *A function* $F : \mathbb{R}^d \to [0,1]$ *is the distribution function of some random vector* $(X_1, ..., X_d)$ *if and only if there is a copula* $C : [0,1]^d \to [0,1]$ *and univariate distribution functions* $F_1, ..., F_d : \mathbb{R} \to [0,1]$ *such that*

$$C(F_1(x_1), ..., F_d(x_d)) = F(x_1, ..., x_d), \quad x_1, ..., x_d \in \mathbb{R}, \tag{2.2}$$

where C is unique if $F_j, j = 1, ..., d$ *is continuous.*

Sklar's Theorem shows the decomposition of a multivariate distribution into two components, namely the marginal distributions ($F_i, i = 1, ..., d$) and the dependence structure (C). Additionally, the theorem indicates, that every multivariate probability distribution can be split into marginals and copula [8].

2.2. Special Cases of Copulas

As first concrete examples of copulas, it makes sense to consider the most fundamental types, in part because they are connected to the more sophisticated ones. When reasoning about the correlation of two random variables, there are three trivial cases that have to be considered:

- Independence: The variables are independent from each other and do not correlate at all. Analytically this copula is given by:

$$C_{\prod}(u_1, ..., u_d) = \prod_{i=1}^{d} u_i, \tag{2.3}$$

  and can be sampled by simply drawing d independent uniform random variables. A 2-dimensional plot of a simulation would therefore look like figure 2.1.

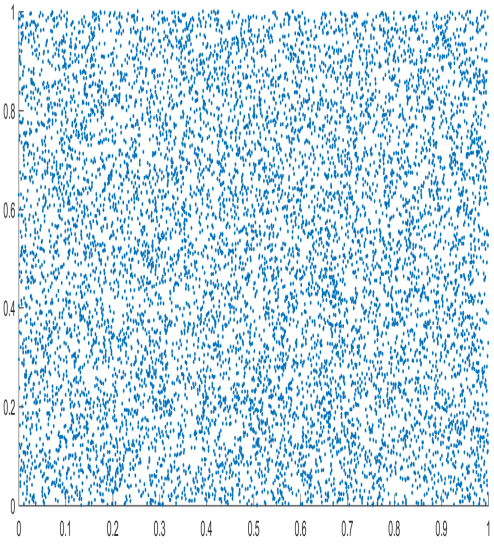

Figure 2.1.: MATLAB plot of a simulated Independence copula

- Comonotonicty: The variables show perfect positive dependence. The functional form of this special type of copula is as follows:

$$C_M(u_1, ...u_d) = \min\{u_1, ..., u_d\}. \tag{2.4}$$

The trivial simulation procedure here is given by drawing one uniform random variable and setting every value of the other d dimensions equal to the first draw, such that the value of all variables is equal. A simulation of a 2-dimensional comonotonicity copula can be seen in figure 2.2.

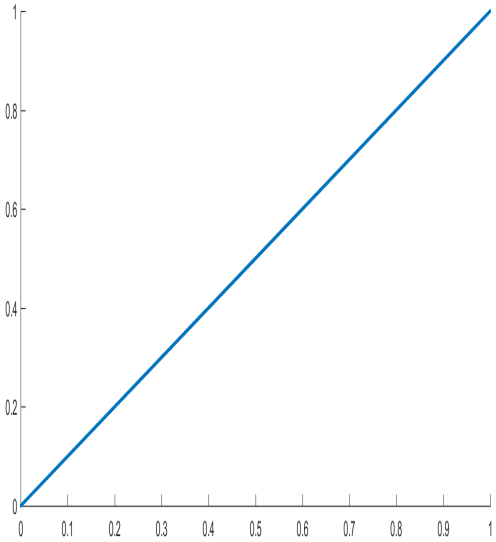

Figure 2.2.: MATLAB plot of a simulated Comonotonicity copula

- Countermonotonicity: The variables show perfect negative dependence. This type of dependence can only be exhibited by two random variables and is therefore only defined for two marginals.

$$W(u_1, u_2) = \max\{u_1 + u_2 - 1, 0\}. \tag{2.5}$$

Once again, a straight forward sampling approach is to sample one uniform random variable U_1 for the first dimension and set the second one equal to $(1 - U_1)$. The graphical representation can be seen in figure 2.3.

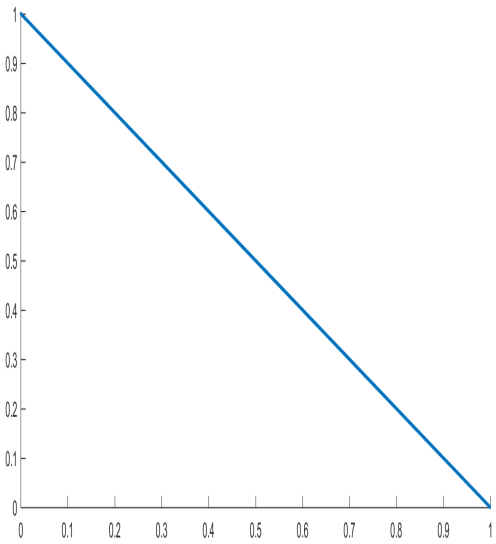

Figure 2.3.: MATLAB plot of a simulated Countermonotonicity copula

2.3. Elliptical Copulas

Elliptical copulas earn their names due to the fact that they are related to some elliptical distribution. More precisely, they are obtained with the help of Sklar's theorem where the distribution function F is classified as elliptical [7] and most commonly Gaussian or Student's t. Note that this family of copulas is often also referred to as *implicit copulas*, as they are implicitly given through some distribution. Formally, this class of copulas is defined as follows:

Definition 2 *An elliptical copula is a copula, where F follows an elliptical distribution and is given by:*

$$C(u_1, ..., u_d) := F(F_1^{-1}(u_1), ..., F_d^{-1}(u_d)), \quad (u_1, ..., u_d) \in [0,1]^d, \tag{2.6}$$

where F_k^{-1} are the univariate quantile functions, $k = 1, ..., d$.

2.3.1. Gaussian Copula

The most famous type of copula is the Gaussian copula, which was already introduced informally in the introductory chapter. Mathematically it is defined according to the following definition:

Definition 3 *The Gaussian copula C_{P}^{Gauss} is the copula of $X \sim \mathcal{N}(0, \mathrm{P})$, with P as the correlation matrix. The functional definition is given by*

$$C_{\mathrm{P}}^{Gauss}(u_1, ..., u_d) := F_{\mathrm{P}}(\Phi^{-1}(u_1), ..., \Phi^{-1}(u_d)), \tag{2.7}$$

where $(u_1, ..., u_d) \in [0,1]^d$, F_{P} *is the joint distribution function of* X, *and* Φ^{-1} *is the quantile function of the standard normal distribution.*

A simulation from a 2-dimensional Gaussian copula can be seen in figure 2.4.

2.3.2. *t*-Copula

The *t*-copula is also part of the family of elliptical copulas, but has one significant key feature that the Gaussian copula lacks, namely tail-dependence. Tail dependence generally describes the asymptotic behaviour of multivariate distributions, i.e., how the variables correlate in the extremes. For further reading on tail dependence and explicit calculations for the Gaussian- and *t*-copula, see [6] section 7.2.4 and [9] section 2.13.
The *t*-copula is defined as follows:

Definition 4 *The t-copula* $C^t_{\mathrm{P},\nu}$ *is the copula of* $X \sim t_d(0, \mathrm{P}, \nu)$, *where* P *is a correlation matrix and* $\nu > 0$ *the degrees of freedom. The functional form is given by*

$$C^t_{\mathrm{P},\nu}(u_1, ..., u_d) := t_{\nu,\mathrm{P}}(t_\nu^{-1}(u_1), ..., t_\nu^{-1}(u_d)), \tag{2.8}$$

with $(u_1, ..., u_d) \in [0,1]^d$, $t_{\nu,\mathrm{P}}$ *as the joint distribution function of* X *and* t_ν^{-1} *as the quantile function of the univariate standard t-distribution with* ν *degrees of freedom .*

An example for a simulation with a *t*-copula can be seen in figure 2.4. When looking at the structure of the points in the bottom-left and top-right corner of (b) and (d), the difference in tail-dependence of the two copulas gets visible by the arrangement of the simulated points. The points in (d) have a tighter formation in the two corners, compared to the multivariate distribution with the Gaussian copula in (b).

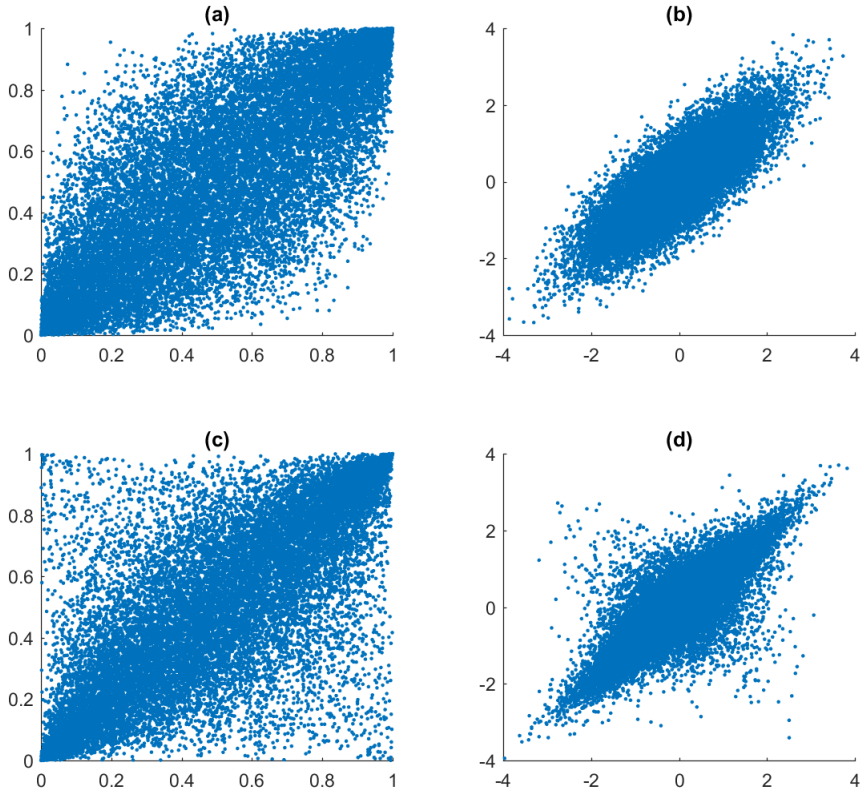

Figure 2.4.: MATLAB plot of 20,000 simulated points from (a) a Gaussian copula with $\rho = 0.8$, (b) a bivariate distribution with the same copula as in (a) and standard normal marginals, (c) a t-copula with parameters $\nu = 2$ and $\rho = 0.8$, (d) a bivariate distribution with the same copula as in (c) and standard normal marginals.

2.4. Archimedean Copulas

Archimedean copulas differ from elliptical copulas in a number of ways. First off, they are not related to any multivariate distribution. Instead, they rely on a so-called generator ϕ for their construction. For that reason, they are often also referred to as *explicit copulas*. The functional representation of an Archimedean copula is given as:

$$C_\phi(u_1, ..., u_d) := \phi(\phi^{-1}(u_1) + ... + \phi^{-1}(u_d)). \tag{2.9}$$

Here, the generator ϕ has to fulfill a number of properties, as C_ϕ needs to have uniformly distributed marginals, in order to fit the definition of a copula. For this to hold we define Archimedean generators as follows:

Definition 5 *An (Archimedean) generator is a function* $\phi : [0, \infty) \to [0, 1]$ *with the following properties:*

1. *$\phi(0)=1$ and* $\lim_{x \nearrow \infty} \phi(x) = 0$,
2. *ϕ is continuous,*
3. *ϕ is decreasing on [0,∞) and strictly decreasing on* $[0, \inf\{x > 0 : \phi(x) = 0\})$, *where* $\inf \emptyset := \infty$.

For a proof that these properties are necessary to obtain a copula with an Archimedean generator, see [7] section 2.2.1. In the following, two popular instances of Archimedean copulas will be introduced, by giving their respective generator functions and highlighting some of their properties. Further tools, necessary for their simulation, will be introduced in later sections and synthesized in section 2.6. Analytical definitions of these copulas are left out, as they do not play any role in the simulations, but can be found in the corresponding chapters of [7].

2.4.1. Clayton Copula

The Clayton copula is characterized through its generator function ϕ and its inverse ϕ^{-1}, which are given as:

$$\phi(x) = (1+x)^{-1/\theta}, \quad \phi^{-1}(x) = x^{-\theta} - 1, \quad \theta \in (0, \infty). \tag{2.10}$$

It possesses lower tail dependence and no upper tail dependence, as can be seen in figure 2.5. Furthermore, the correlation-defining parameter θ is defined on an open right-unbounded interval, which seems counter-intuitive as correlation is usually defined on [-1,1]. This aspect will prove to be interesting with regards to simulations, as limiting cases for θ must be tested. The correlation, as Kendall's Tau (see section 2.7.1), is obtained through the formula $\tau_{C_\theta^{Cl}} = \theta/(2+\theta)$.

2.4.2. Gumbel Copula

The second Archimidean copula that is subject to this thesis is the Gumbel copula with the following generator function ϕ and its inverse ϕ^{-1}:

$$\phi(x) = e^{-x^{1/\theta}}, \quad \phi^{-1}(x) = (-\log(x))^{\theta}, \quad \theta \in [1, \infty). \tag{2.11}$$

This copula exhibits upper tail dependence with no lower tail dependence, as can be verified in figure 2.5. The correlation is given by $\tau_{C_\theta^G} = (\theta - 1)/\theta$. Here, only the limiting case of $\theta \to \infty$ has to be examined in the simulation, since the defining interval of θ is closed.

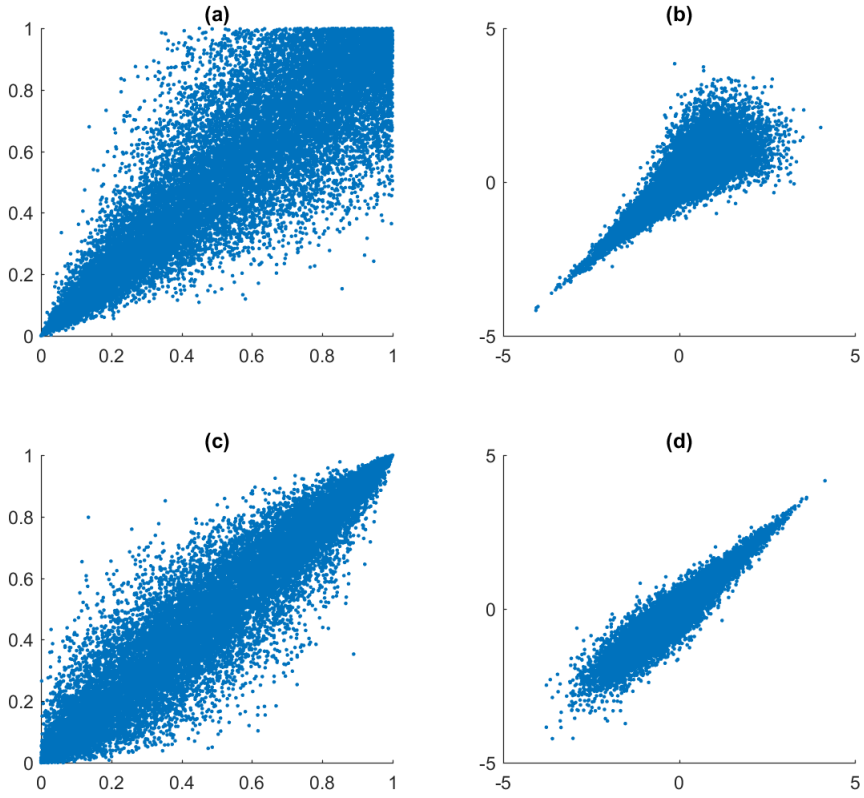

Figure 2.5.: MATLAB plot of 20,000 simulated points from (a) a Clayton copula with $\theta = 5$, (b) a bivariate distribution with the same copula as in (a) and standard normal marginals, (c) Gumbel copula with $\theta = 5$, (d) a bivariate distribution with the same copula as in (c) and standard normal marginals.

2.5. Generation of Non-Uniform Random Numbers

The generation of non-uniform random numbers plays a crucial role in the simulation of copulas and is an unavoidable procedure that is part of every sampling scheme regardless of the copula type. Two major methods for this cause, are firstly, the *Inverse Transform Method* or *Inversion Method* and secondly, the *Acceptance-Rejection Method*, both of which will be used for the simulations performed in this thesis. Generally speaking, one would prefer the *Inverse Method*, as it constitutes a straight forward, analytical approach. However, for the generation of some distributions, it is more sensible to rely on the *Acceptance-Rejection Method*, as the inverse of some distributions is expensive to evaluate or, in some cases, might not even exist [7].

2.5.1. The Inverse Transform Method

The theoretical foundation for the inverse transform method relies on Theorem 3.1 of [10] and makes use of the fact that the cumulative distribution function F of every univariate distribution is distributed uniformly. Therefore, we can take the inverse F^{-1} of F, plug in uniformly distributed numbers and thereby obtain numbers with the desired distribution F. Formally, this looks as follows: Let $U \sim U(0,1)$ and F be the cdf of some univariate distribution with F^{-1} its inverse. Then $F^{-1}(U)$ has the distribution function F.

Generating Exponentially Distributed Random Numbers

The generation of $\text{Exp}(\lambda)$ distributed random numbers can be performed perfectly through the inversion method. Its inverse distribution function is given as $F^{-1}(u) = -\frac{\log(1-u)}{\lambda}$, $0 \leq u < 1$. Therefore, having a random variable $U \sim U(0,1)$, we can obtain the desired exponential random variable $Y \sim \text{Exp}(\lambda)$ by using $Y := -\log(U)/\lambda$. Note that $1-U$ and U have the same distribution and were exchanged in the above formula for simplification purposes. The uniform numbers U plugged into this formula should be verified beforehand for their exact range, as $\log(0)$ is undefined.

Generating Normally Distributed Random Numbers

The inverse cumulative distribution function of the normal distribution has no explicit analytical representation. It is therefore unavoidable, to either use a method of another category than inversion, such as the *Box-Muller Method*, or rely on approximations of the inverse commonly denoted by Φ^{-1}. There exist many such approximations, plenty of which are mentioned in [7]. However, the algorithm of choice for this thesis is the *PPND7* algorithm from [11], which yields results that are accurate and perform well in terms of runtime. Note that the runtime is very important here, as a bivariate simulation of 20,000 points with standard normal marginals, requires 40,000 calls on this function. Real use cases of simulations would usually involve a much higher number of marginals and evaluations of the funtion.

Generating Chi-Square and *t*-Distributed Random Numbers

The generation of a chi-squared distributed random variable χ^2_ν is required when one wants to obtain a multivariate Student's t-distribution or even a single Student's t-distributed random number. Once again there are a number of ways to solve this problem, as there is no way to compute the inverse directly. For this thesis, an approximation of the inverse cumulative distribution function through a Taylor series expansion was used. It was first proposed by Best and Roberts in 1975 [12] and later on refined by Shea (1991) [13]. The algorithms and the code are openly available and will therefore not be shown here.
To further obtain a Student's t-distributed random number, one has to generate a standard normally distributed random variable $Z \sim N(0,1)$, as well as a chi-squared distributed variable $V \sim \chi^2_\nu$ and then return $X := \frac{Z}{\sqrt{V/\nu}} \sim t_\nu$.

2.5.2. The Acceptance-Rejection Method

There are a couple of reasons one would want to refrain from using the inverse transform method: Firstly, for some distributions, there exist only approximations for the inverse density function F^{-1}. Secondly, there are cases where evaluating F^{-1} is very time-consuming and requires excessive computational effort. This is where the acceptance-rejection method (AR method) offers an alternative to circumvent these issues.
John von Neumann proposed the method in 1951. The theoretical foundation can be looked up in [10] Theorem 3.3 with the proof for it attached subsequently and a generalized algorithm for the method as well. Specific ones that were implemented will be given in the following.

Generating Gamma-Distributed Random Numbers

Obtaining a random variable with a gamma distribution $X \sim \Gamma(\alpha, \eta)$ can be broken down into two sub-problems, according to [7], where we compute $\Gamma(\alpha, 1)$ with $\alpha < 1$ and $\Gamma(n, 1)$ with $n \in \mathbb{N}$. This enables us to obtain a gamma distribution with any desired parameters, due to:

- The scaling property: $X \sim \Gamma(\alpha, \eta) \Rightarrow cX \sim \Gamma(\alpha, \eta/c)$ for any $c > 0$,
- $\Gamma(\alpha, 1) \sim \Gamma(\lfloor\alpha\rfloor, 1) * \Gamma(\alpha - \lfloor\alpha\rfloor, 1)$, where the operator "$*$" stands for convolution.

For the integer part of the calculation, Cheng's algorithm (see section 6.8.1 of [7] for more detail) is used, which goes as follows:

Algorithm 1 (input: $\alpha > 1$):

1. Generate $U_1, U_2 \sim U(0, 1)$.
2. $Y := \frac{1}{\sqrt{2\alpha - 1}} \log(\frac{U_1}{1 - U_1})$.
3. $X := \alpha e^Y$.
4. $Z := U_1^2 U_2$.
5. $R := \alpha - \log(4) + (\alpha + \sqrt{2\alpha - 1})Y - X$.
6. If $(R \geq 4.5Z - 1 - \log(4.5))$ or $(R < \log(Z))$ then return X.
7. Else goto 1.

We are now left with the problem of generating a random variable $X \sim \Gamma(\alpha, 1)$ with $0 < \alpha < 1$. For this part, two Algorithms will be considered and later compared. The first of which is taken from [7]:

Algorithm 2 (input: $0 < \alpha < 1$):

1. $a := (1-\alpha)\alpha^{\alpha/(1-\alpha)}$.
2. Generate $E_1, E_2 \sim Exp(1)$.
3. $X := E_1^{1/\alpha}$.
4. If $(X > E_1 + E_2 - a)$ return X.
5. Else goto 2.

Due to reasons further outlined in chapter 4, a second approach to the generation of a gamma-distributed random variable was reviewed in this thesis. It is taken from [14] and supposed to work well with small α values. The algorithm requires the two following functions, which will be defined up front, as they would otherwise hinder the readability of the pseudo-code:

$$h_\alpha(z) = ce^{-z-e^{-x/\alpha}}, \quad z \in (-\infty, \infty),$$

$$\eta_\alpha(z) = \begin{cases} ce^{-x}, & \text{for } z \geq 0, \\ cw\lambda e^{\lambda x}, & \text{for } z < 0, \end{cases}$$

where $\lambda := \alpha^{-1} - 1$ and $c = \Gamma(\alpha + 1)$ with Γ denoting the gamma function. The pseudo-code looks as follows:

Algorithm 3 (input: $0 < \alpha < 1$):

1. $\lambda := \alpha^{-1} - 1$.
2. $w := \alpha / e(1-\alpha)$.
3. $r := (1+w)^{-1}$.
4. Generate $U_1, U_2, U_3 \sim U(0,1)$.
5. If $U_1 \leq r$ then $z = -\log(U_1/r)$.
6. Else $z = \log(U_2)/\lambda$.
7. If $h_\alpha(z)/\eta_\alpha(z) > U_3$ then return $e^{-z/\alpha}$.
8. Else goto 4.

2.5.3. Generating Stable-Distributed Random Numbers

The generation of a stable-distributed random number $X \sim S(\alpha, 1, 0)$ has an elegant method of its own, which is neither an Inversion- nor an AR Method. The algorithm is defined by the following steps:

Algorithm 4 (input: $0 < \alpha \leq 1$):

1. Generate $U \sim U(0, 1)$.
2. Generate $E \sim Exp(1)$.
3. $U := \pi(U - \frac{1}{2})$, such that $U \sim U(-\frac{\pi}{2}, \frac{\pi}{2})$.
4. Return $X := \frac{\sin(\alpha(\pi/2+U))}{\cos(U)^{1/\alpha}} \left(\frac{\cos(U-\alpha(\pi/2+U))}{E}\right)^{(1-\alpha)/\alpha}$.

It will be further needed for the sampling of Gumbel copulas.

2.6. Additional Operations and Functions involved in Copula Simulations

Cholesky Decomposition

As we will see in later sections, the generation of elliptical copulas, requires us to compute the Cholesky decomposition of the correlation matrix, in order to obtain the lower triangular matrix of the decomposition and with it eventually, a multivariate normal distribution with a certain degree of correlation. The goal is to decompose some matrix A into a lower triangular matrix L, such that $LL^T = A$. Note that this operation is an optimized version of the LU decomposition, which can only be applied to *symmetric* and *positive-definite* matrices, such as correlation or covariance matrices. This decomposition requires $N^3/6$ operations to be executed, with N denoting the size of the matrix. This is about half as many operations as required for the LU decomposition. For more detail and an implementation, see [15].

Calculating the CDF of the Normal Distribution

Additionally to the calculation of the inverse $\Phi^{-1}(x)$, we will also need an efficient way of evaluating the standard normal cumulative distribution function $\Phi(x)$ for the generation of a multivariate normal distribution, which, in turn, is required for the Gaussian and t-copula. This function, just like its inverse (section 2.5.1.), has no analytical representation. It is

therefore necessary to rely on approximations, which there are a number of. In this thesis, one provided by W.J. Cody in 1969 [16] is used for this cause.

Calculating the CDF of the Student's *t*-Distribution

The Student's t-distribution's cumulative distribution function will be needed for the simulation of t-copulas. There is a well-known way to evaluate it, which can be looked up in [15], where even code for its calculation is provided.

2.7. Simulation Algorithms

In this section the simulation algorithms will be presented. Those are mainly based on [6], but can generally be regarded as the most efficient and current ones for the purpose of copula simulations, as other recent books such as [7], recommend essentially the same procedures. Note that this section will only highlight the schemes. The implementation itself will be presented in the next chapter.

2.7.1. Multivariate Normal Distribution

The aim of the following algorithm is to simulate a multivariate normal distribution $N \sim N_d(\mu, \Sigma)$, where d corresponds to the dimension of the distribution, i.e., the number of marginals contained in the distribution, μ is the location vector and Σ the covariance matrix.

Algorithm 5 (input: μ, Σ):

1. Compute the Cholesky decomposition of Σ to obtain $\Sigma^{1/2}$, which is the lower triangular matrix of the decomposition.
2. Generate a vector $Z = (Z_1, ..., Z_d)'$ of independent standard normal variates.
3. Return $X = \mu + \Sigma^{1/2} Z$.

2.7.2. Multivariate *t*-Distribution

The next algorithm is meant to generate a multivariate t-distribution $X \sim t_d(\nu, \mu, \Sigma)$ with the same parameters as for the multivariate normal distribution and additionally ν denoting the degrees of freedom.

Algorithm 6 (input: ν, μ, Σ):

1. Generate $Z \sim N_d(0, \Sigma)$ using Algorithm 5.
2. Generate $W \sim \chi^2_\nu$.
3. Return $X = \mu + \sqrt{\nu/W}Z$.

2.7.3. Gaussian Copula

With Algorithm 5 we are now able to generate the Gaussian copula C^{Ga}_{P}, where P is the correlation matrix.

Algorithm 7 (input: P):

1. Generate $Z \sim N_d(0, \mathrm{P})$ using Algorithm 5.
2. Return $U = (\Phi(Z_1), ..., \Phi(Z_d))'$, where Φ is the standard normal cdf.

2.7.4. *t*-Copula

Similarly, the *t*-copula $C^t_{\nu,\mathrm{P}}$ is obtained with the following algorithm:

Algorithm 8 (input: ν, P):

1. Generate $Z \sim t_d(\nu, 0, \mathrm{P})$ using Algorithm 6.
2. Return $U = (t_v(Z_1), ..., t_v(Z_d))'$, where t_ν is the standard cdf of the univariate distribution t_ν.

2.7.5. Clayton Copula

Moving on to the Archimedean copulas, samples of a Clayton copula C^{Cl}_θ can be obtained by means of the following procedure:

Algorithm 9 (input: $\theta \in (0, \infty)$):

1. Generate $V \sim Ga(1/\theta, 1)$.
2. Generate $X_1, ..., X_d$, such that $X_i \sim U(0,1)$, $i = 1, ..., d$.
3. Transform $X_1, ..., X_d$ into independent, standard exponentially ($Exp(1)$) distributed random numbers by applying the Inverse cdf $F^{-1}(x) = -\ln x$ onto each element of the Vector $X_1, ..., X_d$.
4. Return $U = ((1 + (X_1/V))^{-1/\theta}), ..., (1 + (X_d/V))^{-1/\theta}))'$.

2.7.6. Gumbel Copula

And finally, a sample Gumbel copula C_θ^{Gu} can be generated with the next Algorithm:

Algorithm 10 (input: $\theta \in [1, \infty)$):

1. Generate $V \sim St(1/\theta, 1, 0)$.
2. Generate $X_1, ..., X_d$, such that $X_i \sim U(0,1)$, $i = 1, ..., d$.
3. Transform $X_1, ..., X_d$ into independent, standard exponentially ($Exp(1)$) distributed random numbers by applying the Inverse cdf $F^{-1}(x) = -\ln x$ onto each element of the Vector $X_1, ..., X_d$.
4. Return $U = (\exp(-(X_1/V)^{1/\theta}), ..., \exp(-(X_d/V)^{1/\theta}))'$.

Due to reasons that will further be outlined in chapter 4, a second approach was developed as part of this thesis. It is based on Algorithm 10 and 4, with the difference that instead of having these algorithms separated, they were merged and subsequently transformed mathematically, such that exponentiation with $\alpha = 1/\theta$ occurs just once. While the exact transformation steps will be shown in the next chapter, the resulting algorithm goes as follows:

Algorithm 11 (input: $\theta \in [1, \infty)$):

1. Generate $U_1, U_2 \sim U(0,1)$.
2. $E_2 = -\log(U_2)$.
3. $U1 = \pi(U_1 - 0.5)$.
4. $a = \frac{0.5\pi + U_1}{\theta}$.
5. $s = \sin(a)$.
6. $c = \cos(U_1 - a)$.
7. Generate $X_1, ..., X_d$, such that $X_i \sim U(0,1), i = 1, ..., d$.
8. Transform $X_1, ..., X_d$, such that $X_i \sim Exp(1), i = 1, ..., d$ by applying $F^{-1}(x) = -lnx$ onto each element $X_1, ..., X_d$.
9. Return $U = (exp(-\frac{\cos(U_1)E_2}{c}(\frac{X_1 c}{s})^{1/\theta}), ..., exp(-\frac{\cos(U_1)E_2}{c}(\frac{X_d c}{s})^{1/\theta}))'$.

2.8. Statistical Measures of Correlation

Quantifying the dependence between two random variables plays a key role in the simulation of copulas, as every type of copula has some parameter that determines the relationship between the variables in terms of how strongly they depend on each other, if at all. The statistical measures introduced in this section offer the possibility to be computed for empirical samples, as well as through the predefined parameters of each copula. This provides an opportunity to hold the values obtained through simulations against the theoretical ones and thus, test the implementation numerically, as by the law of large numbers, the observed values should be converging towards the theoretical ones with an increasing sample size.
In the following, the two most important measures of dependence, used in the domain of copulas, will be introduced. Those are: Kendall's Tau and Spearman's Rho. Both of these measures are so called *concordance measures*. The term *concordant* stands for two pairs of values $(u_1, u_2), (v_1, v_2) \in [0,1]^2$, where both components of the first pairs of values u_1, u_2, are either greater or smaller than v_1, v_2. Mathematically, this condition is formulated as: $(u_1 - v_1)(u_2 - v_2) > 0$. If it does not hold one would speak of *discordance* instead.
Another common property of the mentioned statistics is their scale of measurement and interpretation on that scale. Concordant measures were introduced by Scarsini in 1984 and are functions from the set of copulas to the interval [-1,1] that satisfy a set of axioms [7]. While the extreme cases of -1 and 1 indicate perfect negative dependence (countermonotonicity copula) and perfect positive dependence (comonotonicity copula) respectively, the case of

0 is mapped onto the independence copula. Values in between those are to be interpreted accordingly [7]. Note that if these measures yield 0, this does not necessarily imply the independence of the marginals [9].

2.8.1. Kendall's Tau

Definition 6 *Let $C \sim (U_1, U_2)$ be a bivariate copula. Kendall's tau of C is given by:*

$$\tau_C := 4\mathbb{E}[C(U_1, U_2)] - 1. \tag{2.12}$$

This formula can be applied to obtain the theoretical values, towards which a simulation should converge. Calculating this measure for a sample $(y_{i1}, y_{i2}), i = 1, ..., n$ is done with the following formula taken from [9]:

$$\tau = \frac{2}{n(n-1)} \sum_{1 \le i < j \le n} [I((y_{i1} - y_{j1})(y_{i2} - y_{j2}) > 0) - I((y_{i1} - y_{j1})(y_{i2} - y_{j2}) < 0)], \tag{2.13}$$

where I is the indicator function. Note that the computation of τ requires $O(n^2)$ operations, which makes it time consuming for large samples.
The relationships between Kendall's tau and the copulas subject to this thesis are shown in table 2.1. For proof on the formula for the Gaussian and t-copula, see [6]. A proof for the Clayton and Gumbel copula can be found in [17].

2.8.2. Spearman's Rho

Definition 7 *Let $C \sim (U_1, U_2)$ be a bivariate copula. Spearman's rho of C is given by:*

$$\rho_C := Corr(U_1, U_2) = 12Cov(U_1, U_2). \tag{2.14}$$

Again, the above formula is used for the calculation of the theoretical value, which can be held against the empirical one of a sample $(y_{i1}, y_{i2}), i = 1, ..., n$, by means of the following formula from [9]:

$$\rho_S = \frac{\sum_{i=1}^{n} r_{i1} r_{i2} - n[(n+1)/2]^2}{n(n^2-1)/12}, \tag{2.15}$$

where r_{i1}, r_{i2} are the ranks and $r_{i1} = k$ if y_{i1} is the kth smallest element among the first sample, $y_{11}, ..., y_{n1}$, $r_{i2} = l$ if y_{i2} is the lth smallest element of the second sample $y_{12}, ..., y_{n2}$. In terms of complexity, this measure is more efficiently computable than Kendall's tau, as the ranks can be computed in $O(n \log(n))$.
However, as can be seen in table 2.1, Spearman's rho can only be computed for the Gaussian copula. Proof for this formula can be found in [6], where it is also mentioned that there is no simple formula for ρ_S, with regards to the t-copula. For the Archimedean copulas, literature does not provide any formula either.

Copula type	τ	ρ_S
C_ρ^{Ga}	$\frac{2}{\pi}\arcsin\rho$	$\frac{6}{\pi}\arcsin\frac{1}{2}\rho$
C_ρ^{t}	$\frac{2}{\pi}\arcsin\rho$	-
C_θ^{Cl}	$\theta/(\theta+2)$	-
C_θ^{Gu}	$1-1/\theta$	-

Table 2.1.: Relationship between statistical measures and different copulas

3. Implementation and Testing

The simulation algorithms described in this thesis, have been implemented for potential use in the context of a risk management application, as used in banks and insurance companies. This chapter is dedicated to describe how the whole process of simulating a multivariate distribution with a copula and marginals, can be implemented with an object oriented programming language such as C#, which was used for this thesis. A key feature of this approach is the extendability to facilitate the addition of new copulas and marginal distributions to the existing implementation. To achieve this, a strong focus will be put on the decomposition of the process into its components (classes) and the design (ordering of classes).

3.1. Code Structure

As the whole mathematical theory behind the concept of copulas can be broken down naturally into pieces as shown in chapter 2, this presents an opportunity to adopt this property with regards to the actual implementation. Moreover, the problem allows for a hierarchical ordering of the components as can be seen in figure 3.1. The Interface *INumberGenerator* specifies a class that is somehow involved in the process of generating random numbers with certain properties, the most basic example being uniformly distributed random numbers. Starting with a uniform number generator, it is possible to transform the numbers obtained through it into a desired univariate distribution through the methods described in section 2.5, and further generate multivariate distributions with the algorithms presented in 2.7.

3.1.1. Pseudorandom Number Generators

In order to sample from copulas and multivariate distributions, we require uniformly distributed random numbers, which we can then transform. However, it is well-known that computers cannot generate truly random numbers, because of their deterministic nature [15]. Therefore, the usage of Pseudo-random number generators (PRNG), as the fundamental building block for all random numbers generated is indispensable and constitutes the bottom layer of the hierarchy in figure 3.1. The interface *IUniformNumberGenerator* specifies a class that can generate a vector of independent uniformly distributed random numbers. Since an extensive review of those generators lies beyond the scope of this thesis, figure 3.1 displays an interface instead of an actual class, such that one can use one of the many existing implementations for PSRNGs. The PSRNG used for the plots displayed in the following is the *xoroshiro128+ 1.0*, which was developed by Blackman and Vigna (2016). The code is openly available under the following reference: `http://xoroshiro.di.unimi.it/xoroshiro128plus.c`.

3.1.2. Elliptical Copulas

Multivariate Normal Sampling

As a preliminary step for the generation of a multivariate Normal distributed sample, the Cholesky decomposition of the covariance matrix should be performed within the constructor of the *MultivariateNormalSampler* and its lower triangular matrix saved as a member variable (_factorMatrix: double[][] in figure 3.1), to avoid doing this computationally expensive operation more than once. Starting at bottom-right of figure 3.1, we are able to obtain a d-dimensional vector $u = (u_1, ..., u_d)$ of independent uniformly distributed random numbers from an implementation of choice of the interface *IUniformNumberGenerator*. Having obtained the uniforms, those are transformed into a sample $z = (z_1, ..., z_d)$ of independent standard normal numbers by applying Φ^{-1} (see section 2.5.1) onto each element of u and thereby complete step 2 of Algorithm 5. To obtain the desired multivariate normal distribution, all that is left to do, is perform the basic mathematical operations described in step 3 (Algorithm 5).

A couple of measures for performance optimization were implemented here:
1) At every generation of a single random vector, the lower triangular factor matrix has to be multiplied with the vector of standard normal variables according to step 3 of Algorithm 5. This is by far the most time-consuming operation within the whole simulation algorithm, without considering the Cholesky decomposition. Especially since, in practice, simulations are run in high dimensions, which increases the duration of this operation even more. An optimization in this case is possible due to the fact that the factor matrix is a lower triangular matrix. The naive implementation of a matrix-vector multiplication has to perform n^2 double multiplications for a $n \times n$ factor matrix. Exploiting the matrix's property, a more efficient implementation reduces the number of multiplications required to $\frac{1}{2}n^2 + \frac{1}{2}n$. The exact code looks as follows:

```
private double[] FactorMatrixMultiplication(double[][] lowerTriangularMatrix,
                                            double[] vector)
       {
           double[] result = new double[vector.Length];
           for (int i = 0; i < vector.Length; i++)
           {
               for (int j = 0; j <= i; j++)
               {
                   result[i] += lowerTriangularMatrix[i][j] * vector[j];
               }
           }
           return result;
       }
```

2) In order to optimize memory allocation, the method *NextVector*, which provides the random numbers of one sampling iteration, is a call by reference function. This makes a difference,

as the method is called within for loops that might invoke it several thousands of times, depending on the sample size desired for a particular simulation.

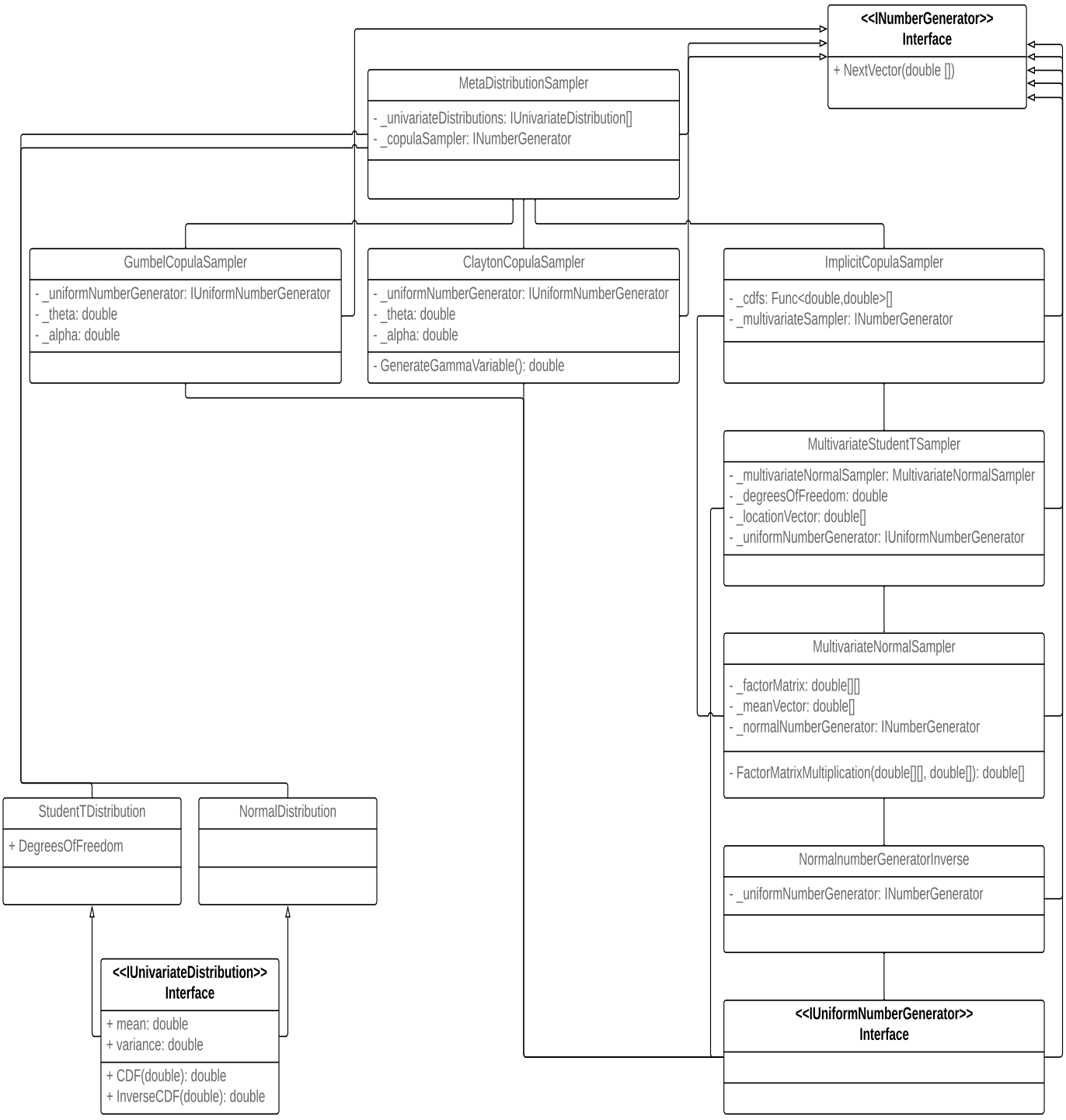

Figure 3.1.: UML Diagram of classes involved in simulations of copulas

Mutlivariate t Sampling

Moving upwards on the hierarchy of classes in the right-column displayed in figure 3.1, the *MultivariateStudentTSampler* enables the generation of correlated t variables according to Algorithm 6. Since the first step of the algorithm requires us to generate a multivariate Normal distribution, the usage of the *MultivariateNormalSampler* is evident. As a second step a $W \sim \chi^2_\nu$ distributed variable has to be generated where $\nu \in \mathbb{R}^+$ represents the degrees of freedom of the multivariate t distribution. In order to generate W make use of the inversion method proposed in section 2.5.1 on an independently generated uniform number (hence, the additional uniform number generator in the class). This approach is used to circumvent the need for a gamma variable, as gamma variables are more difficult to simulate and prone to numerical errors, with regards to limiting cases of parameters, as we will see later on. Having completed these steps, there are only the basic computations of step 3 left to solve for the multivariate t-distribution to be obtained.

Implicit Copula Sampling

With the help of the two previously described classes, it is now possible to generate multivariate Gaussian and t-distributions with any desired correlation and dimension. However, this is still not a copula, which can easily be noticed due to the fact that the points obtained by means of those classes are on the interval $(-\infty; \infty)$. The vector with random points of the copula is build by applying either the cdf of the standard normal distribution ($N(0,1)$) in the case of a Gaussian copula or the cdf of t-distribution with the specified degrees of freedom ν, as described in Algorithm 7 and 8. This step is implemented in the *ImplicitCopulaSampler*, which invokes *NextVector* on either one of the multivariate samplers, then applies the corresponding cdf onto the sampled values it obtains from the underneath layer and thereby, constructing a sample from a copula.

3.1.3. Archimedean Copulas

Due to the fact that Archimedean copulas have a fundamentally different nature than elliptical ones, the implementation approach differs as well. A first observation that can be made, when looking at the simulation algorithms, is that the univariate distributions involved differ completely. While elliptical copulas rely on the generation of a correlated multivariate normal distribution, which involves the computationally expensive vector-matrix multiplication (see Algorithm 5), Archimedean copulas seem to be generated more effortlessly at a first glance, since we only need a sample of independent standard exponentially distributed random variables as a basis. Consequently, the code structure results in a flatter hierarchy.

Clayton Copula Sampling

The first and, as we will see in the next chapter, most challenging step for the simulation of a Clayton copula C^{Cl}_θ consists of the generation of a gamma-distributed random variable

$V \sim \Gamma(1/\theta = \alpha, 1)$. There are two proposed methods in section 2.5.2. for this, which will be evaluated in the next chapter. The method *GenerateGammaVariable* provides the gamma variable, using the member variable *_alpha* as a parameter. Next, uniformly distributed random numbers $(U_1, ..., U_d)$ are needed, which are subsequently transformed to exponential ones $(E_1, ..., E_d)$ using the straight forward approach outlined in section 2.5.1.. Finally, the inverse of the generator function Φ^{-1} is applied to each element of the vector $(\frac{E_1}{V}, ..., \frac{E_d}{V})$ and to complete the process of generating random numbers from the Clayton copula.

Gumbel Copula Sampling

In this thesis, as already mentioned, two approaches for the simulation of a Gumbel copula will be reviewed. The first one makes use of Algorithm 4, for the generation of a Stable distributed random variable V, which would be implemented in an extra method, which is not displayed as part of the *GumbelCopulaSampler* in 3.1. Having obtained this variable, the only thing left to do is take the same steps as for the simulation of a Clayton copula and apply the generator function of the Gumbel copula on the exponential variables divided by V.
The second approach is a derived version of the just mentioned one and was developed in order to avoid multiple exponential operations with α or $1/\theta$. By substituting X of step 4 Algorithm 4 into the variable V of Algorithm 10, one can make mathematical transformation and use the relationship between α and θ to obtain Algorithm 11. To prove the correctness of this, all transformation steps are provided below.

$$
\begin{aligned}
& \exp\left(-\left(\frac{X_i}{V}\right)^{\alpha}\right) \\
= \; & \exp\left(-\left(\frac{X_i}{\frac{\sin(\alpha(\pi/2+U))}{\cos(U)^{1/\alpha}}\left(\frac{cos(U-\alpha(\pi/2+U))}{E}\right)^{(1-\alpha)/\alpha}}\right)^{\alpha}\right) \\
= \; & \exp\left(-\left(\frac{X_i}{\frac{\sin(\alpha(\pi/2+U))}{\cos(U)^{1/\alpha}}\left(\frac{cos(U-\alpha(\pi/2+U))}{E}\right)^{(1/\alpha-1)}}\right)^{\alpha}\right) \\
= \; & \exp\left(-\left(\frac{X_i}{\frac{\sin(\alpha(\pi/2+U))}{\cos(U)^{1/\alpha}}\left(\frac{cos(U-\alpha(\pi/2+U))}{E}\right)^{1/\alpha}\frac{E}{cos(U-\alpha(\pi/2+U))}}\right)^{\alpha}\right) \\
= \; & \exp\left(-\left(\frac{X_i^{\alpha}}{\left(\frac{\sin(\alpha(\pi/2+U))E}{\cos(U-\alpha(\pi/2+U))}\right)^{\alpha}\frac{\cos(U-\alpha(\pi/2+U))}{\cos(U)E}}\right)\right) \\
= \; & \exp\left(-\left(\left(\frac{X_i\cos(U-\alpha(\pi/2+U))}{\sin(\alpha(\pi/2+U))E}\right)^{\alpha}\frac{cos(U-\alpha(\pi/2+U))}{E}\right)\right)
\end{aligned}
$$

Meta Distributions

With the just presented algorithms, one can generate samples from any of the four copulas that are part of this thesis. However, as the capability of defining ones own marginal distributions is part of the simulation of arbitrary multivariate distributions, a last transformation step is necessary, where we transform the uniformly distributed numbers obtained through the copula into some other distributions of choice, while maintaining the copula's dependence structure. The *MetaDistributionSampler* class at the top of the hierarchy in figure 3.1, provides this functionality by containing the set of desired marginal distributions and applying their inverse cdfs on the values obtained by one of the copula samplers.

Bottom-up sampling procedure

In order to offer a sequential perspective on how the simulations can be run with the presented implementation, the following shows the general procedure:

1. Generate an instance of the class *MetaDistributionSampler*, alongside with an array of univariate distributions (marginals) and a copula with the necessary parameters.
2. Sample by invoking *NextVector* on the *MetaDistributionSampler* as often as desired.

An example of a simulation would then look as follows:

```
public static void simulationMain(){

    var RV1 = new NormalDistribution(0, 1);
    var RV2 = new StudentTDistribution(4);
    IUnivariateDistribution[] univariateDistributions = new[] { RV1, RV2 };

    var correlationParam = 0.7;
    var correlationMatrix = new double[2][] {
                        new double[2] {1.0, correlationParam},
                        new double[2] {correlationParam, 1.0}
                    };

    copula = new StudentTCopula(3.5, correlationMatrix);

    var uniformNumberGeneratortype =
              NumberGeneratorUniformVariates.Xoroshiro128Plus;
    var seed = 0.0;

    var sampler = new MetaDistributionSampler(
                    univariateDistributions,
                    copula,
                    uniformNumberGeneratortype,
                    seed);

    var samplesize = 10_000;
    double[][] sample = new double[samplesize][];

    for (int i = 0; i < samplesize; i++)
    {
        sample[i] = new double[univariateDistributions.Length];
        sampler.NextVector(sample[i]);
    }

}
```

The above code simulates 10,000 realizations from a bivariate joint distribution consisting of the marginals $RV1 \sim \mathcal{N}(0,1)$ and $RV2 \sim t(4)$, with a correlation of $\rho = 0.7$ and a t-copula with $\nu = 3.5$.

The *NextVector* call inside the for-loop gets delegated all the way down to the uniform num-

ber generator within the *NormalNumberGeneratorInverse*, which obtains $U(0,1)$ distributed random numbers and transforms those into $\mathcal{N}(\mu, \Sigma)$ distributed random numbers. If the copula is a t-copula, the numbers are further processed to fit $t_\nu(\mu, \Sigma)$. The *ImplicitCopulaSampler* then applies either the Student's t-distribution's cdf or the normal distribution's cdf onto the obtained values, thus mapping them back into the range $[0,1]$. If the sample should have non-uniform marginals, the *MetaDistributionsSampler* can transform those in the last step. This can also be skipped by just starting the whole simulation with the *ImplicitCopulaSampler*.

3.2. Testing

Since the above implementation involves a lot of calculations and different components, rigorous testing is necessary in order to validate the algorithm's correct performance. Copulas allow for a two-fold approach to testing to be used. On one hand visual tests of scatterplots can be used, as well as numerical tests, which involve statistical measures such as the introduced Kendall's tau and Spearman's rho. Both of these have their strength and weaknesses. Hence, conclusions about the validation of the implementation should only be drawn in accordance with both, visual and numerical tests. The before mentioned modularity and hierarchical ordering of the presented implementation proves to be advantageous with regard to testing, as each component can be tested separately. For instance one can check if the uniform number generator really provides uniformly distributed random numbers, then put the next layer on top and check for correctness again. Bugs are thereby easily recognized and located during the implementation process.

3.2.1. Visual Tests

When testing the simulation algorithms visually, knowledge about the underlying theory is required, as plots have to be examined with the eye. One approach here is to tweak one parameter at a time, such that desired changes should be visible in the graphs. A second approach that can be used is to plot certain empirical values of the simulation such as mean, variance of the marginals, and check if those are close to the theoretical values that should be obtained and converge with an increasing sample size. The main problem with visual tests is that there obviously is a lack of accuracy, as there is no strict acceptance/rejection threshold. Therefore, the results of these tests should be treated cautiously. However, basic structures in scatterplots of copulas should be recognized and give a first idea about the correctness.

3.2.2. Numerical Tests

The general approach to numerical testing is more straight forward than for visual tests. Chapter 3 already introduced statistical measures, such as Kendall's tau and Spearmann's rho, which are generally used to measure the correlation between two random variables. For most copulas, the theoretical value of some statistical measure is known. Thus, the empirical value can be held against the theoretical value and they should not deviate too much from each other. "Too much" is of course not quantifiable but in general, a high sample-size (>10,000)

should not yield a greater difference than roughly 1-2%. These tests are especially useful for implementations in bigger software projects, where such tests can be deployed to be run periodically.

3.2.3. Stress Tests

In general, copulas rely on one parameter to specify the correlation between the variables. This parameter always has a certain range in which it is defined and algorithms should yield fast and correct simulation results for every possible value within that interval. If there are certain limits to this, they should be known and dealt with accordingly. A simulation, has two more parameters which have to be looked at to ensure that there are no configurations that yield bad results. The first one being the size of the simulation in terms of draws from the joint distribution and the second one being the dimensionality or the number of marginals contained in the joint distribution.

4. Results

This chapter is devoted to examine the performance of the implementation presented in the previous chapter. The first step in the evaluation of the implementation consists of making sure that the simulations perform well in terms of accuracy. This involves the tweaking of different parameters across their respective ranges and verification of the introduced statistical measures with regards to differences between theoretical and empirical values. Larger cases of deviations must be further examined in order to determine the cause. This first evaluation step helps indicate the correctness of the implementation as well as potential limitations, such as numerical instabilities in the simulation algorithms. Upon examining the accuracy, the performance with regard to running time will be reviewed. In particular how an increasing amount of dimensions impacts the runtime of simulations. This aspect is crucial as, in practice, simulations are usually run with a great amount of marginals. To test the speed of the simulations, C#'s stopwatch class was used and the garbage collector activated before every simulation in order to get comparable and coherent results. Furthermore, the average time over 5 simulations was calculated to obtain smoother curves. A second variable that impacts the runtime of a simulation is the sample size i.e. how many vectors are drawn from the distribution. However, it is obvious that a change in sample size impacts the runtime linearly, since this modification just constitutes a change in the number of times the "NextVector" method is invoked within a for-loop. Therefore, there is no need for further examination with regards to that.

4.1. Gaussian Copula

As already briefly indicated in chapter 2, the 2-dimensional Gaussian copula only has the Pearson correlation ρ, contained in the correlation matrix P, as a parameter. Since it is defined on the closed interval $[-1, 1]$, testing for accuracy is straight forward. Figure 4.1 and 4.2 show the results of 21 simulations, with ρ increased by 0.1 after each simulation, starting at -1. The plots clearly indicate that the simulations run correctly throughout the range of the parameter ρ, as the blue(theoretical) and red(empirical) lines seem perfectly aligned with each other.

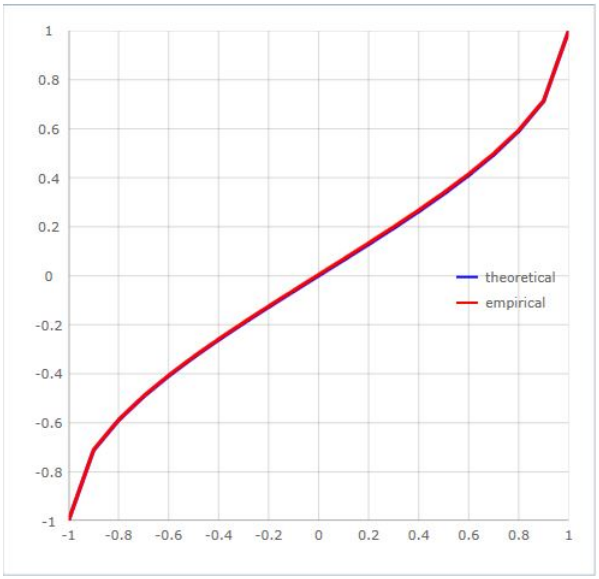

Figure 4.1.: Theoretical and empirical value of Kendall's Tau (y-axis) of a Gaussian copula plotted against $\rho = [-1, 1]$. The simulations were run with 10,000 draws each, at interval steps of 0.1 and interpolated in between.

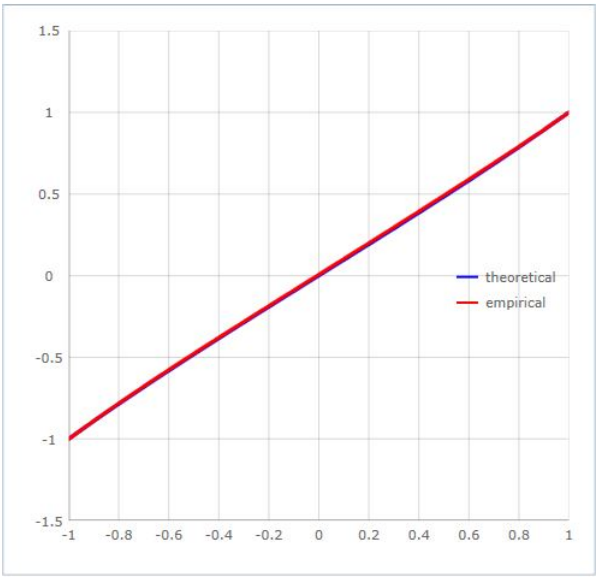

Figure 4.2.: Theoretical and empirical value of Spearman's Rho (vertical axis) of a Gaussian copula plotted against $\rho = [-1, 1]$ (horizontal axis). The simulations were run with 10,000 draws each, at interval steps of 0.1 and interpolated in between.

Examining the speed of the implemented algorithm for the simulation of a Gaussian copula, figure 4.3 shows how the running time is impacted by an increasing number of marginals.

Due to the matrix vector multiplication performed in step 3 of Algorithm 5, we get a runtime complexity of $O(n^2)$ where n denotes the number of dimensions or marginals included in the distribution. Note that this is just the simulation of the copula without any non-uniform marginals added.

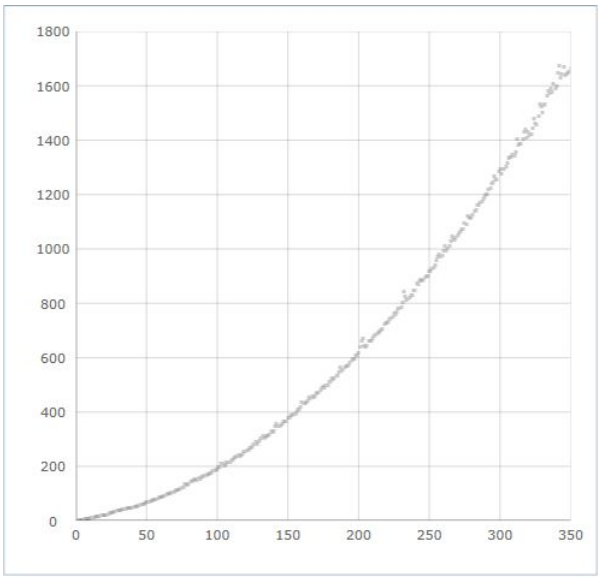

Figure 4.3.: Runtime (vertical axis in milliseconds) of simulations from a Gaussian copula with 3,000 draws and increasing dimensionality (horizontal axis).

4.2. *t*-Copula

In addition to the Pearson correlation ρ contained in the correlation matrix P, the t-copula has the degrees of freedom ν as a second parameter. Table 2.1 indicates that this parameter has no impact on the τ-value of the copula. However, it is still necessary to check limiting cases, as we use the degrees of freedom in the calculations of step 3 in Algorithm 6, where extreme values of ν might cause numerical errors that could lead to incorrect simulations. Figure 4.4 indicates that the implementation works well for values that are usually used as degrees of freedom, such as $\nu = 2.5$.

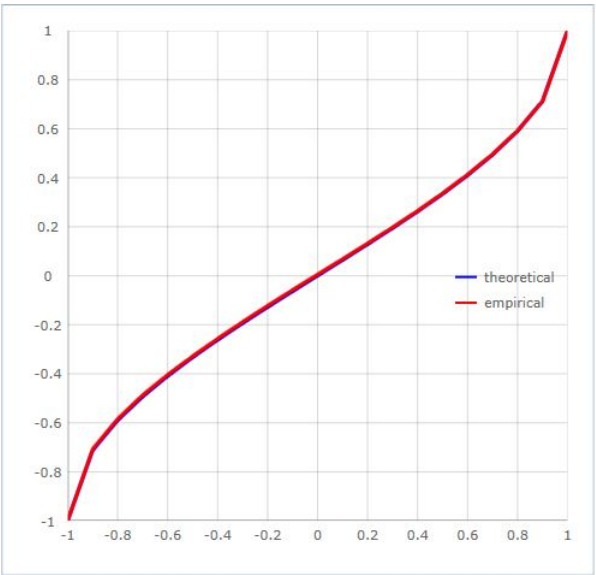

Figure 4.4.: Theoretical and empirical value of Kendall's Tau (vertical axis) of a t-copula with $\nu = 2.5$, plotted against $\rho = [-1, 1]$ (horizontal axis). The simulations were run with 10,000 draws each and at interval steps of 0.1. The curves are interpolated in between.

The same holds for an arbitrarily chosen limiting case ($\nu = 1000$) on the upper side of the scale, as indicated by figure 4.5.

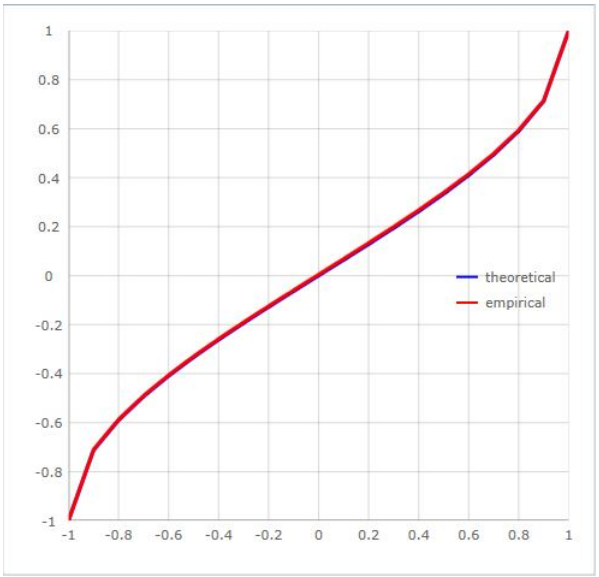

Figure 4.5.: Theoretical and empirical value of Kendall's Tau (vertical axis) of a t-copula with $\nu = 1,000$, plotted against $\rho = [-1, 1]$ (horizontal axis). The simulations were run with 10,000 draws each and at interval steps of 0.1. The curves are interpolated in between.

For values of around $\nu < 0.005$, the accuracy of the implementation starts to vanish more and more, especially in the areas where ρ is close to its boundaries of -1 and 1, as can be observed in figure 4.6. This is due to the fact that the smaller the degrees of freedom value is, the more sample points from the simulated Chi-Squared distribution are equal to zero. Looking at step 3 of Algorithm 6, we have a zero division, which is evaluated to infinity. The square root of infinity is then evaluated to infinity. Therefore, Algorithm 6 returns a number of $+/-\infty$ values, depending on how close to zero ν is. These infinity values then get mapped onto either 0 or 1, according to the sign, when applying the cdf of the Student's t-distribution in step 2 of Algorithm 8.

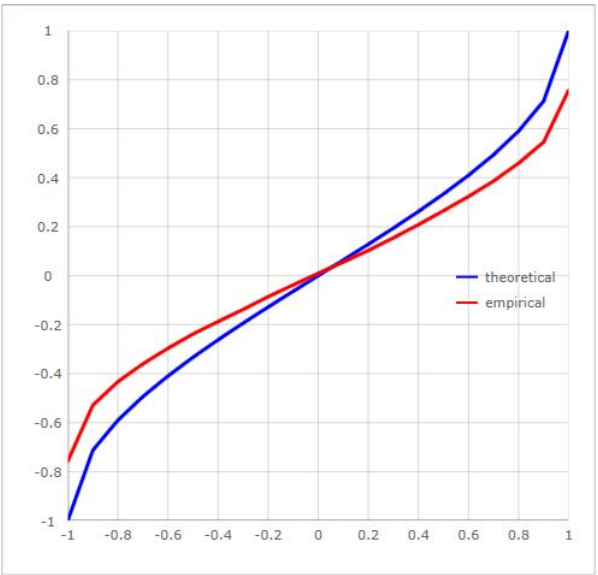

Figure 4.6.: Theoretical and empirical value of Kendall's Tau (vertical axis) of a t-copula with $\nu = 0.001$, plotted against $\rho = [-1, 1]$ (horizontal axis). The simulations were run with 10,000 draws each and at interval steps of 0.1. The curves are interpolated in between.

Figure 4.7 displays how the running time is affected by the number of marginals. As expected simulations take longer than for the Gaussian copula, since the previously identified performance bottleneck in Algorithm 5 cannot be avoided when generating a t-copula according to Algorithm 8. However, the additional operations that are involved cause a significant increase in runtime. While the simulation of a Gaussian copula, with 200 marginals, takes slightly more than 0.6 seconds, a simulation with the same amount of dimensions from the t-copula needs more than 1 second. Despite that the complexity is still $O(n^2)$, as indicated by the slope of the curve, as none of the additional operations has a higher degree of complexity than step 3 of Algorithm 5.

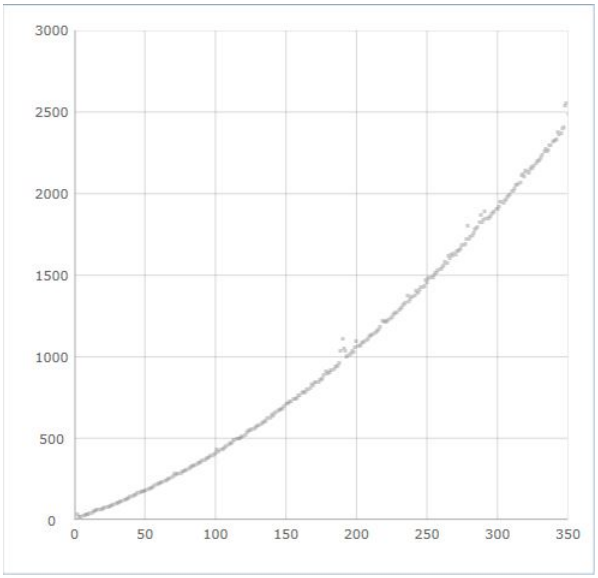

Figure 4.7.: Runtime (vertical axis in milliseconds) of simulations from a *t*-copula with 3,000 draws and increasing dimensionality (horizontal axis).

4.3. Clayton Copula

As mentioned in [14], it gets hard to obtain non-zero numbers for very small α values. Considering this, it makes perfectly sense that Algorithm 9 generates incorrect results for big θ values, as $\alpha = 1/\theta$. When $V = 0$, the division of all $X_d \sim Exp(1)$ by V yields positive infinity in C# according to the rules of the IEEE 754 arithmetic [18]. $\frac{1}{\infty^{1/\theta}}$ consequently evaluates to zero for all sample points. The error caused by this issue gets visible in figure 4.8 where the two curves split more and more apart for increasing θ values. Using the before mentioned *xoroshiro128+ 1.0* PSRNG with seed $= 0.0$, the first zero values in the simulation appear at $\theta = 82.001$. One can filter these zero results out, in order to maintain the possibility of adding non-uniform marginals and get better τ values. However, the simulation would still not result in a mathematically correct copula, due to the marginals not being uniformly distributed as in figure 4.9. Therefore, one would be better advised to use a boundary on the θ values in the implementation or reject zero points and sample those from a comonotonicity copula instead.

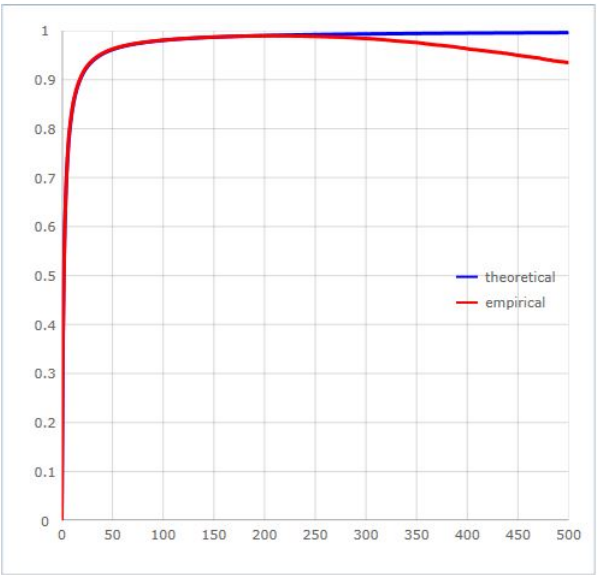

Figure 4.8.: Theoretical and empirical value of Kendall's Tau (vertical axis) of a Clayton copula plotted against $\theta = [0.001, 500.001]$ (horizontal axis). The simulations were run with 10,000 draws each, using Algorithm 2 for the generation of the gamma variate and at interval steps of 1. The curves are interpolated in between.

A simulation with $\theta = 1000$ as in figure 4.9 clearly yields totally incorrect results. It is up for debate to what extend it is an issue that the simulation does not perform accurate anymore for such a great θ value, as it implies a correlation of $\tau \approx 0.998$, which basically implies perfect dependence (comonotonicity copula). However, since the first zero values appear much earlier already, the algorithm should nevertheless be used cautiously, since this prohibits the usage of marginal distributions with the copula due to the fact that $F^{-1}(0) = -\infty$ for common distributions, such as the Gaussian or Student's t.

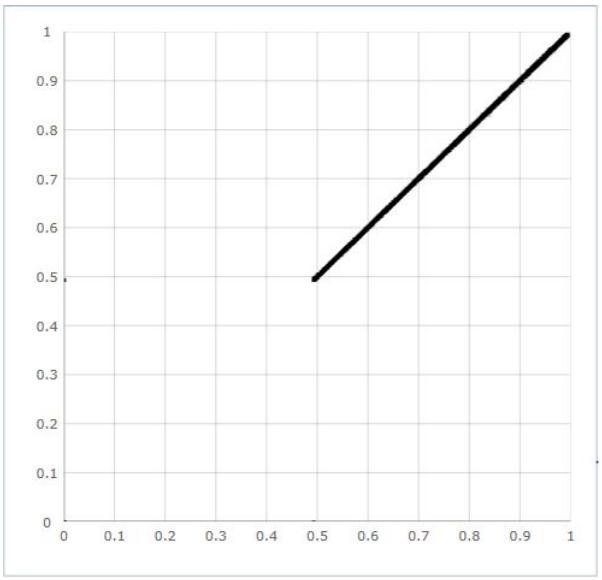

Figure 4.9.: 10,000 simulated points from a Clayton copula with $\theta = 1,000$, using Algorithm 2 for the generation of the gamma variate.

Because of the just presented findings, a second method for the generation of a gamma variate [14] was implemented, which supposedly works well with small α values. The idea was to switch to a better performing algorithm once α $(= 1/\theta)$ reaches a certain threshold. Unfortunately, figure 4.10 shows that simulations of Clayton copulas, with the alternative Algorithm 3 are clearly even more prone to this issue than the initially used Algorithm 2.

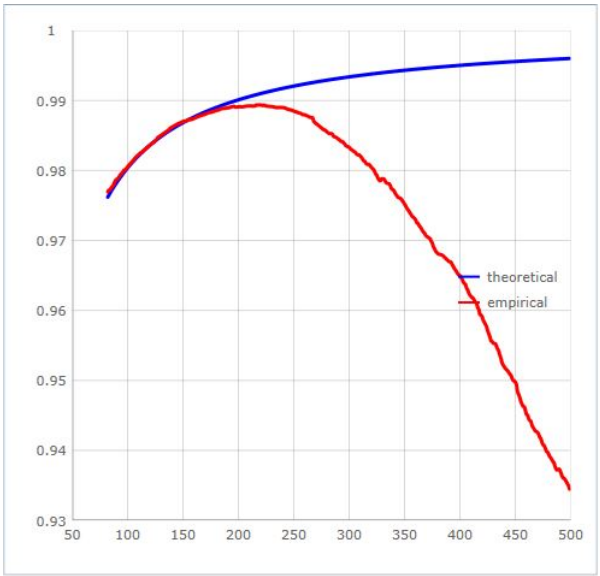

Figure 4.10.: Theoretical and empirical value of Kendall's Tau (vertical axis) of a Clayton copula plotted against $\theta = [82.001, 500.001]$ (horizontal axis). The simulations were run with 10,000 draws each, using Algorithm 3 for the generation of the gamma variate and at interval steps of 1. The curves are interpolated in between.

With regard to performance, as expected, the simulation of a Clayton copula runs a lot faster than any of the Elliptical ones. This is of course primarily because we do not require the computationally expensive generation of a multivariate normal distribution, but also because we require just one gamma distributed random variable, regardless of the number of marginals. Furthermore, no complex computations of inverse cdfs are needed, as the transformation of uniforms to exponentials is easily executed. Thus, we obtain a clearly linear complexity for the simulation of a Clayton copula, as indicated by figure 4.11. The benchmark was run with Algorithm 2 for the Generation of the gamma variate and $\theta = 3.5$.

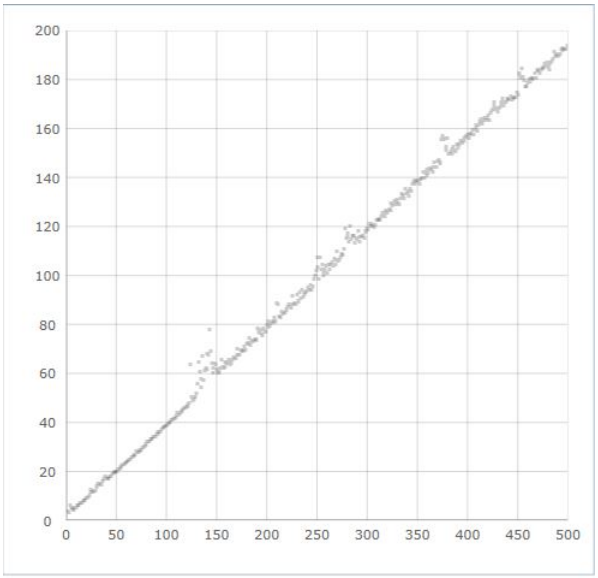

Figure 4.11.: Runtime (vertical axis in milliseconds) of simulations from a Clayton copula with 3,000 draws and increasing dimensionality (horizontal axis).

4.4. Gumbel Copula

Considering the issues encountered with the Clayton copula, which were just presented in the previous section, it would not come as a surprise if similar ones would be encountered for simulations of Gumbel copulas with high θ values or stable variables with close to zero α parameters more precisely. Figure 4.12 clearly shows that there seem to be numerical issues here once again, as the two curves are drifting apart increasingly with θ getting bigger.

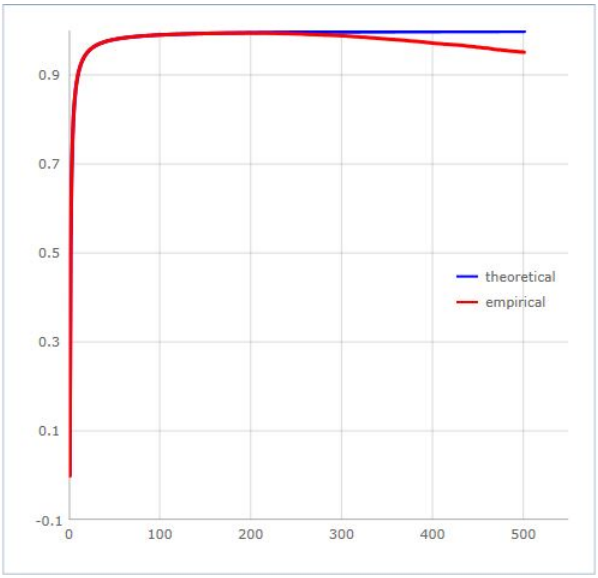

Figure 4.12.: Theoretical and empirical value of Kendall's Tau (vertical axis) of a Gumbel copula plotted against $\theta = [1, 500]$ (horizontal axis). The simulations were run with 5,000 draws each and at interval steps of 1. The curves are interpolated in between.

Looking at a particular instance of a copula with a high parameter, as in figure 4.13, we obtain a completely incorrect result again, since the line should be connecting the points $(0, 0)$ and $(1, 1)$ just like a comonotonicity copula in figure 2.2.

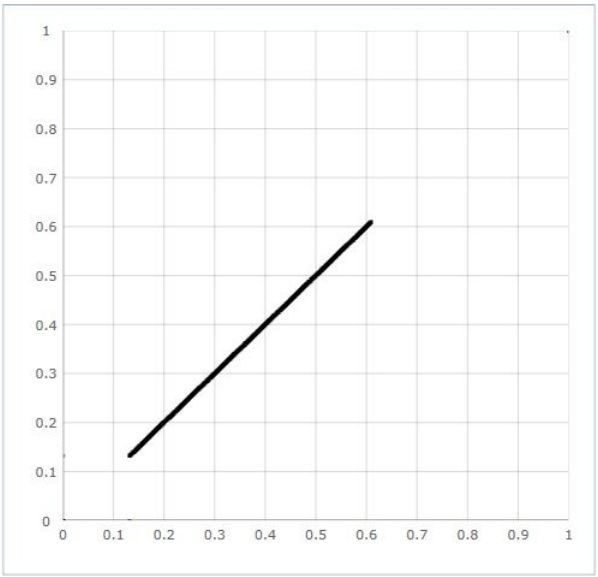

Figure 4.13.: 10,000 simulated points from a Gumbel copula with $\theta = 1,000$, using Algorithm 10.

The problem with Algorithm 10 is that there are two places where $\alpha = 1/\theta$ appears as an exponent. Once in the generation of the stable-distributed random variable, as well as in the last step of Algorithm 10 where we apply the Archimedean generator function of the Gumbel copula. By simulating with Algorithm 11 instead of Algorithm 10 this issue is resolved and simulations with high parameters become numerically stable, due to the exponential operation with α occurring just once. Figure 4.14, shows no significant divergence between the theoretical and empirical values and can therefore be considered better for simulations of a Gumbel copula regardless of the parameter, since zero/one values might start to appear even for higher two digit values of θ, with Algorithm 10, which would prohibit the usage of non-uniform marginals.

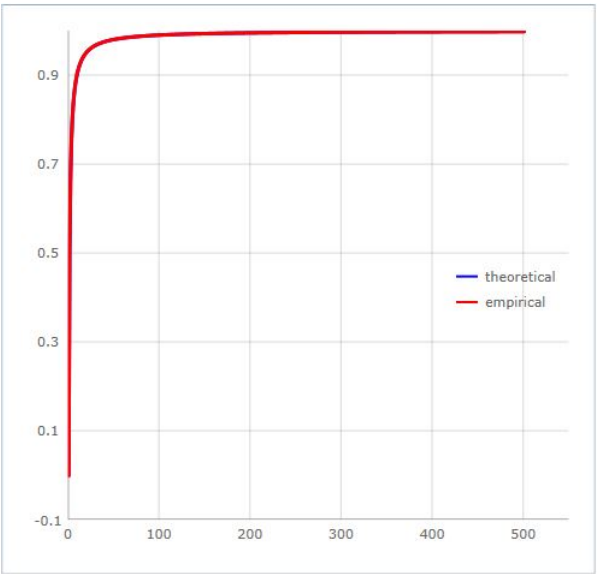

Figure 4.14.: Theoretical and empirical value of Kendall's Tau (vertical axis) of a Gumbel copula plotted against $\theta = [1, 500]$ (horizontal axis). The simulations were run with 5,000 draws each and at interval steps of 1. The curves are interpolated in between.

In terms of runtime, the simulation of the Gumbel copula performs similar to the Clayton copula with a linear complexity. Although, the Clayton copula is simulated slightly faster than the Gumbel. This is very likely due to the vast amount of trigonometric functions that have to be evaluated and do not appear in Algorithm 9 at all. However, even at $\theta = 500$ the difference makes about 0.04 seconds, which is not really significant at the already fast simulation speed of Archimedean copulas.

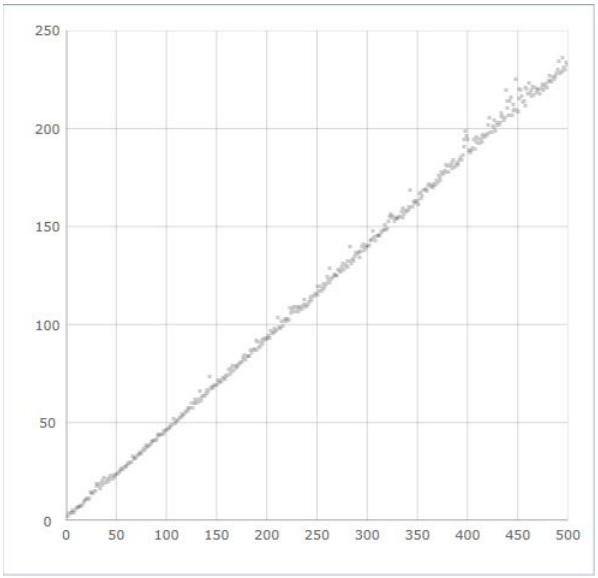

Figure 4.15.: Runtime (vertical axis in milliseconds) of simulations from a Gumbel copula with 3,000 draws and increasing dimensionality (horizontal axis).

5. Conclusions

In this thesis, the approach to modeling and simulating multivariate distributions with copulas was reviewed. Upon recapitulating the necessary mathematical tools and techniques used for this cause, a possible approach to the implementation of the recent copula simulation algorithms was presented. Furthermore, the implementation was tested rigorously using statistical concordance measures to explore the limitations, such that people who use these are aware and able to make adjustment in their implementations.
We are further going to sum up the key takeaways of this thesis, with regards to the simulation of the different copulas subject to it. Beginning with the Gaussian copula, we found that the simulation algorithms run very smooth throughout the range of its parameter, although the run-time was not as good as for the Archimedean copulas, despite efforts for optimizing the necessary processes to sample from it.
The next copula implemented as part of this thesis was the Student t copula. Due to the operations required for its sampling scheme, it performed worst in terms of speed and we obtained inaccurate results for very small degrees of freedom values, because of instabilities inherent to the generation of chi-squared distributed random numbers. Knowing this, different methods for the generation of a chi-square variate could be tried out with this sampling scheme.
For the Clayton copula we reviewed one simulation algorithm combined with two algorithms that generate the gamma distributed random variable necessary for the simulation. Here we discovered that the generation of a gamma variate gets numerically unstable for small shape parameters (α), regardless of the algorithm used and therefore, yields increasingly bad simulations with increasing θ values. For that reason it is advised to set an upper bound on the θ parameter of the copula and reject values where the generated Gamma variable is equal to zero, in order to preserve the possibility of transforming the uniform marginals into some other distribution. Despite those issues, the simulations performed very well in terms of speed with a linear complexity with regards to the number of marginals.
Finally, two approaches for the simulation of a Gumbel copula were implemented and reviewed. One of which relies on the generation of a Stable distributed random variable and a second one, which was derived from the first procedure through mathematical transformations in order to optimize the computations. The results clearly indicate that the first simulation algorithm is prone to numerical instabilities for high θ values, while the alternative algorithm performed accurately throughout the tested range of possible values. Hence, it is advised to use the algorithm developed in this thesis, to obtain correct simulations regardless of the value of θ, while maintaining an efficient linear runtime complexity.

A. Appendix

A.1. Pearson correlation

For two random variables X and Y, the Pearson correlation is given by:

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y}, \tag{A.1}$$

where σ_X denotes the variance of the variable X and $cov(X,Y)$ the covariance between X and Y.

A.2. Correlation Matrices

Having a joint distribution of random variables $F = (X_1, ..., X_d)$ the correlation Matrix P is defined as:

$$\mathrm{P} = \begin{pmatrix} 1 & \rho_{12} & \rho_{13} & ... & \rho_{1d} \\ \rho_{21} & 1 & \rho_{23} & ... & \rho_{2d} \\ \rho_{31} & \rho_{32} & 1 & ... & \rho_{3d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \rho_{d1} & \rho_{d2} & \rho_{d3} & ... & 1 \end{pmatrix}, \tag{A.2}$$

where $\rho_{i,j}$ denotes the Pearson correlation between variable X_i and X_j.

List of Figures

List of Tables

Bibliography

[1] F. Salmon. *Recipe for Disaster: The Formula that Killed Wall Street*. 2009. URL: https://www.wired.com/2009/02/wp-quant/ (visited on 06/16/2019).

[2] D. X. Li. "On Default Correlation". In: *The Journal of Fixed Income* 9.4 (2000), pp. 43–54.

[3] S. Reid. *A Recipe for the 2008 Financial Crisis*. 2015. URL: http://www.turingfinance.com/recipe-for-the-financial-crisis/ (visited on 06/16/2019).

[4] S. Watts. "The Gaussian Copula and the Financial Crisis: A Recipe for Disaster or Cooking the Books?" In: (June 2016).

[5] *Copula (linguistics)*. 2019. URL: https://en.wikipedia.org/wiki/Copula_(linguistics) (visited on 06/16/2019).

[6] A. J. McNeil, R. Frey, and P. Embrechts. *Quantitative Risk Management: Concepts, Techniques and Tools*. Princeton, NJ, USA: Princeton University Press, 2015.

[7] J.-F. Mai and M. Scherer. *Simulating Copulas*. 2nd Edition. Series in Quantitative Finance 6. Singapore: World Scientific, 2017.

[8] J. Mai and M. Scherer. *Financial Engineering with Copulas Explained*. Financial Engineering Explained. Palgrave Macmillan UK, 2014.

[9] H. Joe. *Dependence Modeling with Copulas*. Chapman & Hall/CRC Monographs on Statistics & Applied Probability. Taylor & Francis, 2014.

[10] G. S. Fishman. *Monte Carlo: Concepts, Algorithms and Applications*. New York, NY, USA: Springer Verlag, 1996.

[11] M. J. Wichura. "Algorithm AS 241: The Percentage Points of the Normal Distribution". In: *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 37.3 (1988), pp. 477–484. ISSN: 00359254, 14679876. URL: http://www.jstor.org/stable/2347330.

[12] D. J. Best and D. E. Roberts. "Algorithm AS 91: The Percentage Points of the Chi-Squared Distribution". In: *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 24.3 (1975), pp. 385–388. ISSN: 00359254, 14679876. URL: http://www.jstor.org/stable/2347113.

[13] B. L. Shea. "Algorithm AS R85: A Remark on AS 91: The Percentage Points of the Chi-Squared Distribution". In: *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 40.1 (1991), pp. 233–235. ISSN: 00359254, 14679876. URL: http://www.jstor.org/stable/2347937.

[14] R. Martin and C. Liu. "Simulating from a gamma distribution with small shape parameter". In: *Computational Statistics* (Feb. 2013).

[15] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. 3rd ed. New York, NY, USA: Cambridge University Press, 2007.

[16] W. J. Cody. "Rational Chebyshev approximations for the error function". In: *Math. Comp.* 23.107 (1969), pp. 631–637.

[17] R. B. Nelsen. *An Introduction to Copulas (Springer Series in Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[18] W. Dos Passos. *Numerical methods, algorithms and tools in C#*. Apr. 2016, pp. 1–575.