Algorithms for Handling Multivariate Poisson Distributions

Biko Pougala

4th Year Project Report
Computer Science
School of Informatics
University of Edinburgh

2020

Abstract

Multivariate Poisson distributions that allow dependencies can be used to model various types of data such as traffic accidents, RNA sequencing or insurance claims. However, no well-defined and practical formula has been studied to model these so we introduce a different approach that relies on a copula function. A valid multivariate distribution can be constructed by mapping a copula function to a set of marginals. This method allows to account for a large variety of dependencies between Poisson random variables. In this project we review some copula functions and develop algorithms to generate data, fit a multivariate Poisson distribution and select the optimal parameters of the copula function. We review their performance across various choices of ground truth parameters and conclude that fitting a multivariate distribution improves when the number of samples of each variable increases. We propose the Powell method as the optimal method for the convergence of the inference for margins protocol to optimise the parameters of the copula.

Note

Words in **bold** represent important concepts that have been introduced for the first time.

Table of Contents

Chapter 1

Introduction and background

Poisson distributions are widely used to model "the probability of a given number of events occurring in a fixed period of time if these events occur at a fixed rate and independently of the time the previous event occurred", according to the commonly accepted definition**[1]**.
Multivariate distributions designate probability distributions where more than two random variables are studied at the same time. In the case where the data is continuous, different models exist to describe such distributions:

1. multivariate normal models: these models are used to describe correlated continuous random variables where each random variable is clustered around a mean value,
2. multinomial models: a generalisation of the binomial distribution, these models study the number of occurrences of mutually exclusive events.

However, no such well-defined models exist for the discrete case.
The joint probability mass function of a multivariate distribution with independent discrete marginals is:

$$P_X(X_1,...,X_N) = \Pi_{i=1}^{N} P_X(X_i) \tag{1.1}$$

where $X_1,...,X_N$ are independently-distributed random variables. The obvious limitation of this expression is that it does not allow any form of dependence or correlation between the random variables. The use cases for this formula are therefore very limited as dependent Poisson random variables are very common**[1]**, such as:

1. *text analysis*: the probability that a word appears on a given line is dependent on the other words around. For instance, the word "team" is more likely to appear with the word "sport" than with the word "pancake".
2. *crime statistics*: if the crime rate in neighbouring postcodes is high, it is likely that criminals are travelling from one postcode to another.
3. *genomics*: the probability of seeing a certain sequence of nucleotides is more likely next to another given sequence of nucleotides.

In this project we focus on joint discrete probability distributions where the marginals are Poisson distributions. Although the use cases are many, as seen above, the literal, well-defined expression of a multivariate Poisson distribution has not been extensively studied**[1]**. As an example, one type of d-dimensional distribution proposed is defined as**[2]**:

$$\mathcal{P}_{MultiPoisson}(\mathbf{x};\lambda) = \exp(-\sum_{i=0}^{d}\lambda_i)(\Pi_{i=1}^{d}\frac{\lambda_i^{x_i}}{x_i!})\sum_{z=0}^{min_i x_i}(\Pi_{i=1}^{d}\binom{x_i}{z})z!(\frac{\lambda_0}{\Pi_{i=1}^{d}\lambda_i})^z \tag{1.2}$$

Where $\mathbf{x} \in \mathbb{Z}_+^d$ and λ are the d Poisson rates.
This formula is limited for two reasons: it only accounts for positive dependencies, which significantly limits the scope of applications this formula can be used for, and the formula gets really complex and difficult to compute and extrapolate for a high number of dimensions. We thus choose to take a different approach and introduce **copulas**.

1.1 Copulas

Coming from the Latin *copula* meaning "link", a copula is a multivariate cumulative distribution function defined on the unit hypercube such that its marginals are uniform. A copula function can be used to "glue" together marginal distributions to form a valid joint multivariate probability distribution. Skullbeck-Leiblerar's theorem states: Let F_X be a d-dimensional cumulative distribution function with marginals $F_{X_1},...,F_{X_d}$. Then there exists a d-copula C such that**[2]**:

$$F_X(x_1,...,x_d) = C(F_{X_1}(x_1),...,F_{X_d}(x_d)). \tag{1.3}$$

Copula functions can model both linear and non-linear dependencies. Because the marginals are discrete, C is defined on $[0,1]^d$ and unique on $Ran(F_{X_1}) \times ... \times Ran(F_{X_d})$ where *Ran* is the range.**[11]**

Let $X_1,...,X_n$ be random variables, $F_1,...,F_n$ their respective continuous cumulative distribution functions and H their joint distribution function, then

$$H(x_1,...,x_n) = \mathbb{P}(X_1 \leq x_1,...,X_n \leq x_n) = C(F_1(x_1),...,F_n(x_n)) \tag{1.4}$$

There exist different types of copulas that can each model different types of dependence structures but we have decided to restrain this work to only three of the most popular ones: the **Gaussian copula** and two **Archimedean copulas**: the **Clayton copula** and the **Gumbel copula**.

1.1.1 Concepts of dependence

Dependence, in the area of statistics, refers to a statistical relationship between two or more random variables. Studying the dependence between random variables is useful

because it can help predict the behaviour of a random variable based on the observation of other dependent random variables.
Linear correlations are one of the most popular ways to represent dependence between distinct random variables. If X and Y are random variables, then perfect linear dependence is defined as $Y = aX + b$ where $a \in \mathbb{R} \setminus 0, b \in \mathbb{R}$. One popular example is the Pearson correlation coefficient**[16]**. It is defined as:

$$S_{xx} = \sum_{i=1}^{N} (X_i - \bar{X})^2$$

$$S_{yy} = \sum_{i=1}^{N} (Y_i - \bar{Y})^2$$

$$S_{xy} = \sum_{i=1}^{N} (Y_i - \bar{Y})(X_i - \bar{X})$$

$$r = \frac{S_{xy}}{\sqrt{S_{xx} S_{yy}}}$$

Where X and Y are two N-dimensional random variables.
Correlations range between -1 and 1 : a perfect positive linear correlation between X and Y yields a correlation of $+1$ and a perfect negative linear correlation between X and Y yields a correlation of -1. No correlation at all yields a value of 0.
However, correlation between random variables does not have to be linear, its structure can be complex and intricate. For instance, the correlation can be stronger in some parts of the distribution than in others, or positive in some parts of the distribution and negative in others.
Accounting for various types of dependence structures is therefore essential to efficiently model multivariate probability distributions.

1.1.2 The Gaussian copula

The Gaussian copula is the copula that is associated to a joint Gaussian (normal) distribution. Using **Equation** (1.3), for a given correlation matrix $R \in [-1,1]^{d \times d}$ and with uniformly-distributed random variables $u_1, ..., u_d$ we get**[12]**:

$$C_R^{Gauss} = \Phi_R(\Phi^{-1}(u_1), ..., \Phi^{-1}(u_d)) \tag{1.5}$$

Where Φ_R is the joint cumulative distribution function of a Gaussian distribution with mean vector 0 and correlation matrix equal to R, Φ^{-1} is the inverse cumulative distribution function of a standard Gaussian and d is the dimension of the copula. Note that R must be a positive semi-definite matrix.

We can fix Poisson-distributed random variables $\rho_1, ..., \rho_d$ to the Gaussian copula defined by **Equation** (1.5) by setting $u_n = F_n(x_n)$ where F_i is the i-th marginal cumulative distribution function. Indeed, it is straightforward to see that a continuous cumulative distribution function is uniformly distributed.

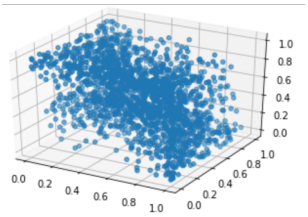

Figure 1.1: A 3-dimensional Gaussian copula with correlation $\sigma = 0.5$

The Gaussian copula is an entropy-maximising distribution[1] which is the reason why it is heavily used to model credit and default risk as this field often requires to work with partial, fragmented data**[4]**.

1.1.3 Archimedean copulas

Archimedean copulas are a special class of copulas. All Archimedean copulas are specified using a single strict generator function. Its theoretical definition is**[9]**:

$$C(u_1, ..., u_d; \theta) = \psi^{-1}[\psi(u_1; \theta) + ... + \psi(u_d; \theta)] \tag{1.6}$$

Where ψ is a continuous generator function such that $\psi(0) = 1$. Many Archimedean copulas choose the inverse of a Laplace-Stieltjes transform as a generator function as it satisfies this condition and is relatively straightforward to compute. $\theta \in \mathbb{R}$ is the unique parameter of this copula which governs the strength of the dependence between the marginals.

One important property Archimedean copulas have is they are flexible enough to accommodate for a wide variety of dependence structures between random variables, including asymmetry and non-linearity. As such, they have become widely popular in econometric applications, particularly in the field of insurance claims.
However, the use of only one parameter to describe the dependency between all random variables is more restrictive than for the Gaussian copula, which uses a correlation matrix specifying a different dependence value for every pair of random variables.**[12]**

1.1.4 Tail dependence

One particular type of dependence is called **tail dependence**. In econometrics applications, the risk-return correlation in a given investment portfolio tends to be higher in bear markets[2] than it is in bull markets[3]**[5]**. It is a clue for a stronger dependence on the tails — extreme ends — of the probability distribution. Other statistical tools have been used to measure this particularity such as the extreme value theory, which can be used to calculate the probability of an extreme, rare event**[8]**. This shows the importance of studying the extremes of a probability distribution and of properly describing this type of correlation. Failure to do so has been widely regarded as one of the

[1] The *maximum entropy principle* states that the most optimal probability distribution to fit to a dataset is the one least biased towards the provided information.

[2] A bear market describes a stock market in decline, where returns on investments are low.

[3] A bull market describes a stock market that is on the rise, where returns on investments are strong.

reasons behind the 2008 economic and financial collapse.**[6][7]** The tail dependence of two random variables describes their comovements at the extremums of the probability distribution, or, visually speaking, the clustering of "extreme" events. There are various ways to represent the tail dependence of bivariate random variables. An important one is called **lower tail dependence** and has been concretised by Joe [1990]:

$$\lambda_L := \lambda_L(C) = \lim_{u \to 0} \frac{\mathbb{P}(X \leq F_X^{-1}(u), Y \leq F_Y^{-1}(u))}{\mathbb{P}(X \leq F_X^{-1}(u))} = \lim_{u \to 0} \frac{C(u,u)}{u}. \tag{1.7}$$

Where C is a copula function $[0,1]^d \to [0,1]$, X and Y are two bivariate random variables, F^{-1} is an inverse cumulative distribution function and u is a number defined in the domain of X and Y. A non-negative value for λ_L translates in a lower tail dependence between X and Y — the dependence will be stronger in the lower (negative) tail of the distribution than in the upper (positive) tail.

1.1.5 The Clayton copula

Named after British statistician David George Clayton who first introduced it in 1978, it is a very popular type of Archimedean copulas because of its asymmetry and its ability to capture lower tail dependencies (as per Figure 1.2). Its theoretical definition**[9]** is:

$$C^{Clayton}(u_1, ..., u_d; \theta) = (max\{1 - d + \sum_{i=1}^{d} u_i^{-\theta}, 0\})^{-1/\theta} \tag{1.8}$$

Where $\theta \in (-1, \infty)/\{0\}$. Note that as $\theta \to 0$, it converges to the independence copula. We can also simplify this definition further in the bivariate case as:

$$C^{BivClayton}(u_1, u_2; \theta) = (\max\{u_1^{-\theta} + u_2^{-\theta} - 1, 0\})^{-1/\theta} \tag{1.9}$$

Its generator function is defined as:

$$\psi(t) = (1+t)^{-1/\theta} \tag{1.10}$$

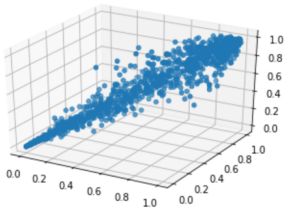

Figure 1.2: A 3-dimensional Clayton copula with parameter $\theta = 7.5$

1.1.6 The Gumbel copula

The **Gumbel** copula (or Gumbel-Hougard copula) is an asymmetric copula that has the particularity of accounting for upper-tail dependence (as per Figure 1.3).

Let $u_1, ..., u_d$ be uniform random variables, $\theta \geq 1$ and C the corresponding copula function, then we get**[9]**:

$$C^{Gumbel}(u_1, ..., u_d; \theta) = \exp(-[\sum_{i=1}^{d}(-\ln u_i)^{\theta}]^{1/\theta}) \tag{1.11}$$

Its generator function is:

$$\psi(t) = \exp(-t^{1/\theta}) \tag{1.12}$$

Note that when $\theta = 1$ we obtain the independence copula.

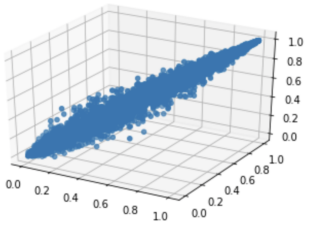

Figure 1.3: A 3-dimensional Gumbel copula with parameter $\theta = 8.5$

Because the Gumbel copula is only defined when its parameter θ is a positive real number, this copula cannot be used to model negative dependencies contrarily to the other two.

There is an open-source package `Copula` written in R to fit multivariate distributions, but it has only been conceived for bivariate continuous distributions. In this project instead we shall work with a higher number of dimensions and with discrete count data.

1.2 The data matrix model

For the purposes of this project, we choose to adopt a unified model for the data which will be called the *data matrix model*. Let X be an $N \times M$ matrix as:

$$\begin{pmatrix} x_{1,1} & x_{1,2} & x_{1,3} & \dots & x_{1,M} \\ x_{2,1} & x_{2,2} & x_{2,3} & \dots & x_{2,M} \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ x_{N,1} & x_{N,2} & x_{N,3} & \dots & x_{N,M} \end{pmatrix}$$

Where each row is an observation and each column is a variable collected from this observation. Each item $x_{i,j}$ is a non-negative integer. The number of columns is fixed and the same for every collection item, but the number of rows is unbounded. The more observations there are, the more complex the structure of the data and its underlining dependence is.

As an example, RNA sequencing can be modelled using this data matrix model: RNA-seq is a process that can be used to study various forms of gene expression and aspects of the RNA, such as RNA translation or structure**[10]**. Let $x_{i,j}$ be the count of reads in a particular DNA location, where i is the gene in question and j is an area of the gene studied. The data is discrete as only counts of reads are being measured, strongly dependent because gene expression works in intricate and complex combinations of different genes and the counts are Poisson distributed.

1.3 The Kullbeck-Leibler Divergence method

The Kullbeck-Leibler divergence number χ, also called *entropy divergence*, is a non-symmetric and non-negative number that measures the similarity between two probability distributions $p(x)$ and $q(x)$**[14]**. When $\chi(p,q) = 0$, the two distributions are identical. Let $p(x)$ and $q(x)$ be two discrete probability distribution functions. Then their Kullbeck-Leibler divergence number χ is defined as:

$$\chi_{KL}(p(x)||q(x)) = \sum_{u \in X} p(u) \ln \frac{p(u)}{q(u)} \tag{1.13}$$

It is not a proper measure of distance because it is not symmetrical and χ_{KL} does not satisfy the triangle inequality, however it can be interpreted as the amount of increase of relative entropy of one probability distribution over another.
This measure will be used to compare the multivariate probability distribution. In this project it will be assumed χ_{KL} is measured in bits.

1.4 Optimisation methods

Two methods to find the optimal values for θ, R and λ have been selected because they have been found effective in similar applications while not relying on unnecessarily complex or cumbersome techniques, like computing gradients. The first one is the **Nelder-Mead method** and the second one is the **Powell's method**.

1.4.1 Nelder-Mead method

It is a commonly-used numerical method for finding global extremums in a multi-dimensional space. It is a relatively straight-forward method to compute because it only requires function evaluations to find the minimum**[18]**.
It starts by randomly initialising a d-dimensional simplex[4] of $d+1$ vertices at x_0, the initial guess. At every step of the algorithm, the simplex will be reshaped, one vertex at a time, and the algorithm will select the alteration that moves the simplex into a "better" part of the search space — the definition of "better" depends on the objective function that is evaluated at every iteration —. Nelder-Mead can be adapted to find the minimum or maximum of an objective function of multiple parameters. One advantage

[4] A simplex is a high-dimensional triangle.

of this method is that, being gradient-free, the Nelder-Mead method does not get fooled by statistical noise, which happens a lot with experimental data, and therefore performs well with objective functions that yield jagged curves. However it is slow and often does not converge due to the tendency of the simplex to irreversibly collapse in a subset of the search space**[18]**.

1.4.2 Powell's method

Powell's method is another optimisation technique to find the extremums of an objective function without computing derivatives. It is initialised by an initial guess x_0 and a set of n independent search vectors $s_1, ..., s_n$. At every step of the algorithm, it will change the solution vector $\mathbf{x}$ to be the minimum value along every search vector from s_1 to s_n. Powell's method is believed to converge faster in well-conditioned problems but suffers from the same search space collapse issues the Nelson-Mead method faces.

Chapter 2

Description of the package

2.1 Overview

The entire project was written in Python 3. Related Python packages that we used include SciPy, Numpy for representing and calculating the probability distributions, Itertools for implementing the inclusion-exclusion principle mentioned in **Section 2.2.4** and MatplotLib to generate some of the plots. We implemented two classes, `MultivariatePoisson` which is used to define and construct the multivariate Poisson distribution and its marginals, and `Copula`, which is used to define the underlying copula function. We designed both classes in an object-oriented way using features such as inheritance and encapsulation so they could be easily extended in the future with additional features. It also helps in the development phase: breaking down code into distinct parts makes debugging easier and allows one to switch on and off various versions of each function and see which one works best. Starting from the `MultivariatePoisson` class, here is a list of its functions and how they work:

1. `init(string family, int theta*, tuple cov*, int seed)`
   Initialises a multivariate Poisson distribution with the copula family `family`, chosen between Gaussian, Clayton or Gumbel. `theta` is a real number representing the parameter θ for the two Archimedean copulas (*cf:* **Section 1.1.3**). Similarly, `cov` is the Gaussian copula parameter R (*cf:* **Section 1.1.2**). Either one, but not both, of `theta` or `cov` must be null, depending on the `family` chosen as these two parameters are mutually exclusive. Acceptable values for `cov` are a scalar, in which case the covariance matrix will be the identity matrix times this number, or a multidimensional array. The optional argument `seed` is used to generate predictable results. The value set to `seed` will be used to create a `RandomState` object from the NumPy package which will then be passed to the functions that use randomly generated data, such as `rvs()`. Using the copula parameters specified as arguments, this function also initialises a `Copula` object. Returns a handle.
   The reason behind creating an `init()` function and having all function calls go through this handle is that the copula and distribution hyperparameters are shared and used by all functions in this class. Therefore using global variables makes

sure that these values are always consistent; the user can access and update them via regular getters and setters for each of these properties and the changes will be propagated everywhere else in the code. These getters and setters are not included in the API since in Python they are not explicit.

2. `rvs(tuple`[1] `mean, tuple size`[2]`, RandomState random_state*)`
   Generates artificial data using the `Copula` object provided by the handle and Poisson rates `mean`, covariance matrix `cov` if the copula is Gaussian or copula parameter `theta` for the Archimedean case, and size `size`, using the procedure detailed in **Algorithm 1**. The data generated is a tuple that follows the data matrix model mentioned in **Section 1.2**.

3. `pmf(tuple data, tuple mean)`
   Calculates the probability mass function of a dataset `data` with Poisson rates `mean` using the `Copula` object provided by the handle. For that it first calculates the joint CDF using function `Copula.cdf()` with the `Copula` object initialised in `init()`. From this result it calculates the PMF using the **inclusion-exclusion principle** of Poincaré and Sylvester, which is further detailed in **Section 2.2.4**. Similarly to `rvs()`, the tuple `data` must comply with the data matrix model. If not, an error is thrown.

4. `marginal_cdf(tuple data, tuple mean)`
   Fits a marginal Poisson distribution to every dimension of tuple `data` using frequencies and SciPy's `poisson.cdf()` function. Let $\mathbf{x_i}$ be a row vector representing a dimension of the data matrix, this function returns a vector $\mathbf{y_i} = (p_{i,1}, ..., p_{i,n})$ where $p(i, j) = P_{x_i}(x_i \leq j)$.

5. `optimise_params(tuple data, tuple mean)`
   Calculates the optimal copula parameters and fits a joint probability mass function as well as marginal distributions for each dimension. It infers the optimal Poisson rates directly from `data` and the optimal copula parameters using the *inference for margins* method through repeated computations of `pmf()` until its likelihood is maximised, a protocol further detailed in **Section 2.2.5**.

The arguments marked with an asterisk are either optional or conditionally optional — optional on condition that the other parameter is not null —. I made the choice to split the functions between these two classes based on the premises of the project: constructing a valid joint multivariate Poisson distribution by mapping marginal distributions, on one side to a copula function, on the other.

We shall now delve into the implementation of the `Copula` class:

1. `init(int theta*, tuple cov*, string family, int seed*)`
   Initialises a new copula of the family `family` among `'gaussian'`, `'clayton'` or `'gumbel'` and with parameter `theta` in case of an Archimedean copula or

[1] The pseudocode notation "tuple" is the equivalent of either an integer or a NumPy array in Python. Either one of the two or both are accepted as a valid input, depending on the context.

[2] If `size` is an integer, then the function generates a `size`-dimensional array with 100 data points in each dimension. If `size` if a Python tuple, then the first element is the number of dimensions, the second element is the number of data points for each dimension.

with parameter `cov` in case of the Gaussian copula. Either one of `theta` and `cov` must be null. As the range for `theta` is different for each family of copulas, an error is thrown if the value provided is out of range for this family. The full protocol for creating a copula is defined in **Algorithm 2**. It returns a handle.

2. `cdf(tuple data, tuple mean)`
   This function constructs a joint multivariate cumulative distribution function by mapping the copula function of the copula provided by the handle to the marginal distributions, as per the protocol specified in **Algorithm 3**. The marginal distributions are calculated independently by calling `marginal_cdf()` from `MultivariatePoisson`. The copula functions used are the ones defined in Equations (1.5), (1.8) and (1.11).

3. `generate(tuple size, tuple mean*)`
   It implements the generator function for each copula family as defined in Equations (1.10) and (1.12) for the Archimedean copulas and (1.5) for the Gaussian copula. This function is required to generate artificial data from the current copula of size `size`. The argument `mean` is only used with the Gaussian copula.

4. `create_data(tuple size, tuple mean*)`
   Samples data from the current copula using the `generate()` function above, the inverse of a Laplace-Stieltjes transform for the copula family, the provided `mean` array for the Poisson rates and the inverse cumulative distribution function of a Poisson. The full protocol is laid out in **Algorithm 1**.

2.2 Analysis of algorithms used

For the various parts of the API, I designed a series of algorithms with guidance from my project supervisor and using formulae and theorems that I have referenced accordingly. These algorithms delve in more details into the theoretical concepts used by the aforementioned API.

2.2.1 Generating artificial data

Algorithm 1: Generating artificial samples from a copula **[16]**

Data: A number of dimensions D, data samples S, an optional mean array μ and a copula `cop`
Result: A multi-dimensional array of data samples `arr`
if $\mu = \varnothing$ **then**
  $\mu \leftarrow$ a random vector drawn from a uniform distribution
end
$X \leftarrow$ `cop.generate((D, S))`
for $i \leftarrow$ *1 to D* **do**
  data $\leftarrow F^{-1}(X_i, \mu_i)$
  `arr`.append(data)
end

Here, F^{-1} is the inverse cumulative distribution function of the Poisson random variable X. The reason why I let μ be optional is that I wanted my package to be as straightforward to use as possible, allowing a user to play around with a dataset without knowing or caring about its nitty-gritty details. In that case, means are randomly generated, drawn from a uniform distribution. The API makes use of a `seed` parameter to return predictable results. To compute the percent-point function I use the function `poisson.ppf` from the SciPy Python library. We can use copulas to generate artificial data that adopts an arbitrary dependence structure governed by the copula parameter. The `cop.generate()` function makes a call to a subroutine defined in **Algorithm 2**. Recall that a copula function is a cumulative distribution function, a mapping $[0,1]^d \rightarrow [0,1]$. A percent-point function is a function $[0,1]^d \rightarrow \mathbb{N}$ that takes a cumulative distribution function as input and maps it to the corresponding integer values**[16]**. It is therefore possible to generate random Poisson variates from a copula function.

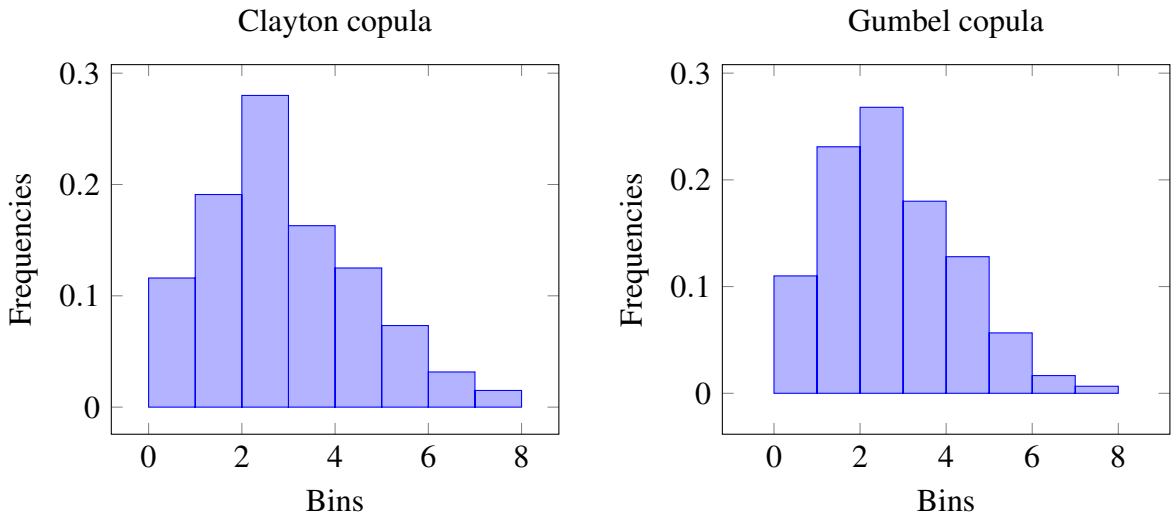

Figure 2.1: Marginal univariate distributions generated from Archimedean copulas with copula parameter $\theta = 8.6$ and Poisson rate $\lambda = 2.2$

The two above marginal univariate distributions are Poisson distributions, with frequencies clustered around the mean value λ and probabilities decaying on either sides of this value.
One thing to notice is that data generated with the same parameters (dependency strength governed by the copula parameter θ and Poisson rate λ) has very similar shape and structure, which means that any copula family can be used to generate data. It shows that all three copulas can generate valid Poisson marginal distributions.

2.2.2 Generating copulas

Algorithm 2: Generating copulas **[9]**

Data: A copula family `family` as a string, a number of dimensions D, data samples S, a covariance matrix R or real number θ and a mean array μ
Result: A multi-dimensional copula cop
if `family` = *Gaussian* **then**
  $X \leftarrow$ draw S samples from $\mathcal{N}^D(\mu, R)$
end
if `family` = *Clayton* **then** $X \leftarrow$ draw S samples from $\Gamma^D(1/\theta, 1)$;
else *// Gumbel case*
  $\beta \leftarrow \cos(\pi/2\theta)^\theta$
  $X \leftarrow$ draw S samples from $LS^D(1/\theta, 1, 0, \beta)$;
for $i \leftarrow$ *1 to* D **do**
  $Y \leftarrow F_X(X)$
  cop.append(Y)
end

Where $N^D(\mu, R)$ is a D-dimensional multivariate Gaussian distribution with mean μ and covariance R. Likewise $\Gamma^D(1/\theta, 1)$ is a D-dimensional Gamma distribution with shape parameter $1/\theta$ and scale parameter 1 and $LS^D(1/\theta, 1, 0, \beta)$ is a Levy-Stable distribution with stability parameter $1/\theta$, skewness parameter 1, scale parameter 0 and location parameter $\cos(\pi/2\theta)^\theta$.
The definition of the Gaussian copula as defined in **Section 1.1.2** gives a straightforward method to sample from a Gaussian copula: draw random samples from a multivariate Gaussian distribution using the same parameters μ and R as the copula.
However, sampling from Archimedean copulas is slightly more complicated. Marshall and Onkin (1988) have devised a protocol to sample a d-dimensional Archimedean copula in 3 steps**[9]**:

1. Sample a vector $\mathbf{V}$ from $LS^{-1}(\psi)$ where $LS^{-1}(\psi)$ is the Levy-Stieltjes transform of the generator function ψ of the Archimedean copula
2. Sample a d-dimensional vector $\mathbf{U}$ from a uniform distribution
3. Return vector $\mathbf{S} = (S_1, ..., S_n)$, where $S_i = \psi(-\log(U_i)/\mathbf{V}), \quad i \in \{1, ..., d\}$

Some Archimedean copulas have a well-defined inverse Laplace-Stieltjes transform, like tha Clayton and Gumbel (*cf:* Equations (1.10) and (1.12)), however some others do not. My original plans for this project included having the Frank copula as one of the accepted copulas. The Frank copula is interesting as its dependence is weaker in both tails of the distribution and stronger in its center. However, the Frank copula lacks an explicit inverse Laplace-Stieltjes transform, which is replaced by a step function. In this case, Step 1 of Marshall-Onkin's procedure is replaced by an infinite logarithmic series expansion to find an approximation of the inverse Laplace-Stieltjes transform. We made various attempts at constructing such expansions which were inconclusive and the results we obtained were not robust enough compared to the other copulas so we made the decision not to go with this copula.

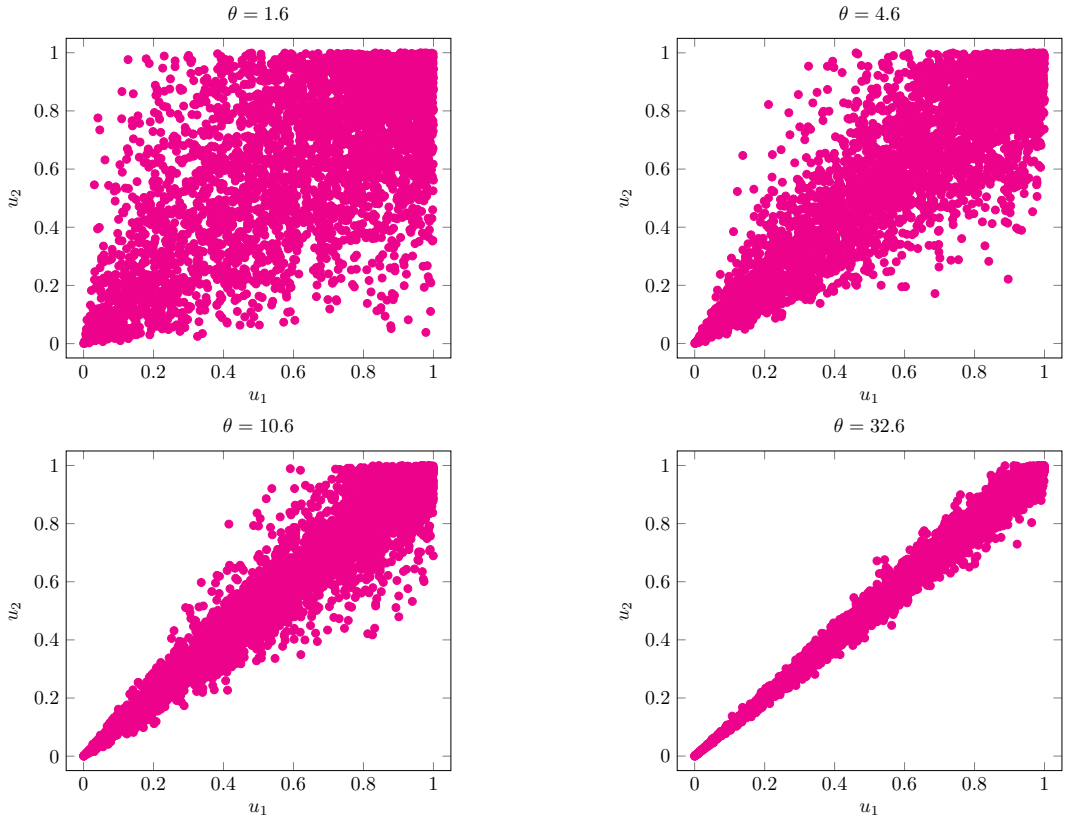

Figure 2.2: Two-dimensional Clayton copula with 4,000 samples generated with a varying value for parameter θ.

The four plots in **Figure 2.2** were obtained by running **Algorithm 2**. u_1 and u_2 are two uniform random variables. The first thing to note is that the probabilities are more concentrated in the negative tail of the distribution than elsewhere. The second thing is that as the value of θ increases, the width of the "beam" of data points shrinks, until the relationship between u_1 and u_2 becomes almost linear. The same two variables u_1 and u_2 are used in all four cases, but a different value for θ makes it possible to express varying levels of co-dependence. The same principle can be generalised to a higher number of dimensions. Note that the copula function is always only defined in the unit hypercube.

In **Figure 2.3**, u_1 and u_2 are two uniform random variables defined on range $[0,1]$ to illustrate the bivariate case of the Gumbel copula. Similarly to the Clayton case, the copula parameter θ acts as a knob to regulate the level of correlation between u_1 and u_2, always with a stronger correlation in the positive tail of the distribution than elsewhere: from a very low level of dependence ($\theta = 1.6$) which results in seemingly uncorrelated random variables, to a strict, almost linear relationship between the two variables ($\theta = 18.6$). The narrower the beam of points is, the more correlated u_1 and u_2 are. One thing to notice is the larger concentration of probabilities in the tails of the bivariate distribution, particularly on the right rail, which is very evident in the case $\theta = 1.6$.

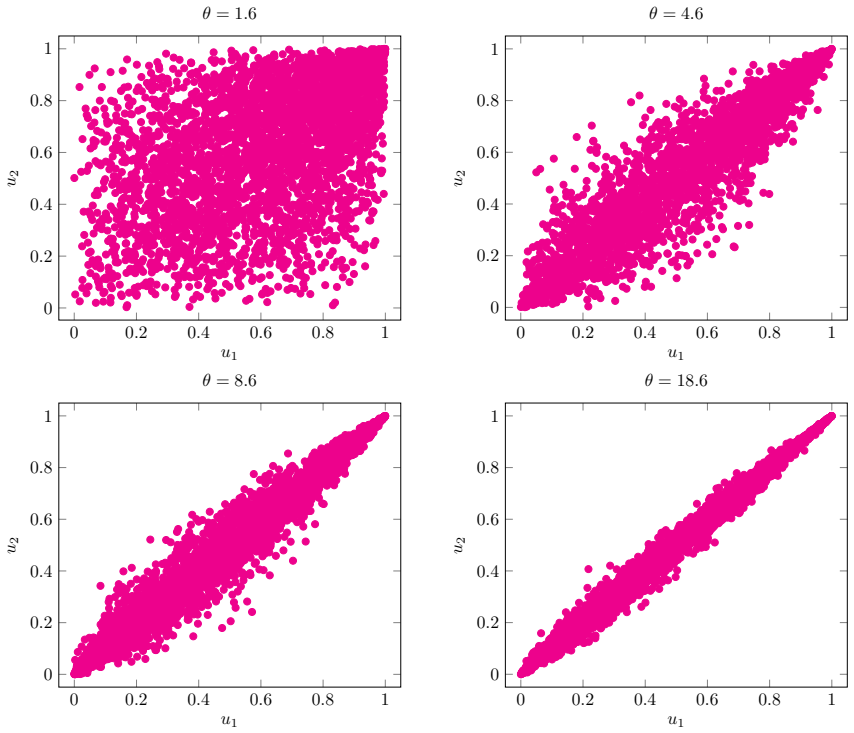

Figure 2.3: Two-dimensional Gumbel copula with 4,000 samples generated with a varying value for parameter θ.

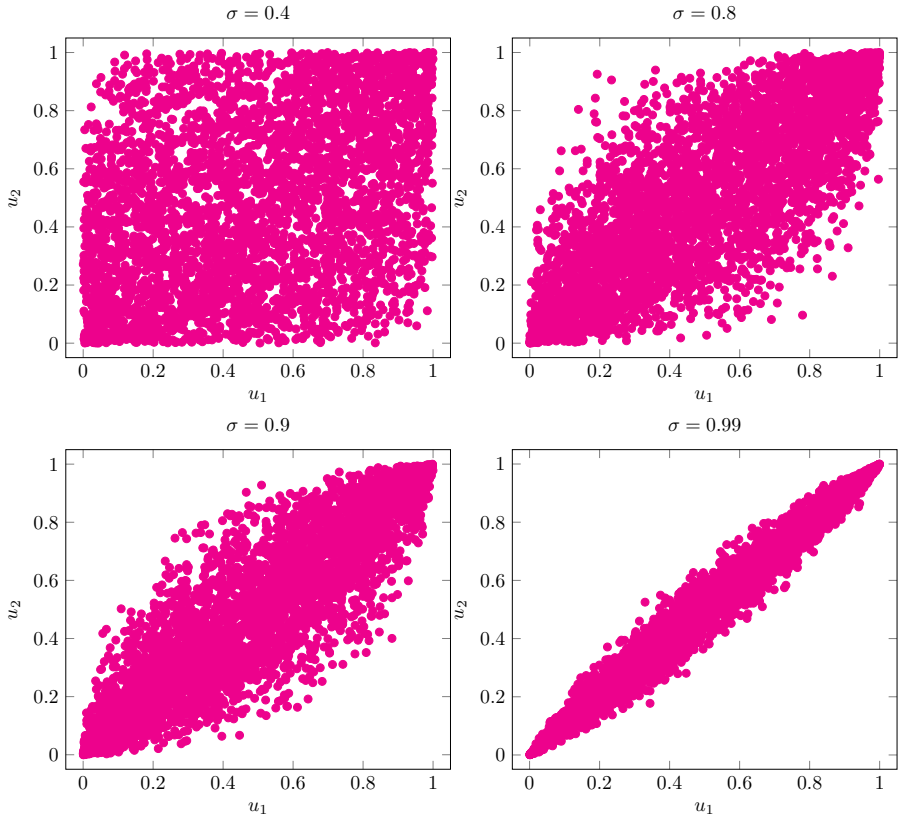

Figure 2.4: Two-dimensional Gaussian copula with 4,000 samples generated with a varying value for correlation σ.

Similarly to the Gumbel copula case, the Gaussian copula also tends towards a linear relationship as correlation σ tends to 1. In the two-dimensional case, the correlation matrix is of the form: $R = \begin{pmatrix} 1 & \sigma \\ \sigma & 1 \end{pmatrix}$ where σ is the correlation coefficient which is defined on range $[0,1]$. Here u_1 and u_2 are two uniform random variables defined on range $[0,1]$ to illustrate the bivariate case. One of the particularities of the Gaussian copula is the fact that every pair of random variables is characterised by its own correlation coefficient, which is a more relevant difference when the number of random variables is greater than 2. On the other hand, Archimedean copulas only have one parameter value to describe the entire dependence structure of the data, regardless of the number of random variables studied, which means that in the Archimedean case, some expressive power is lost when describing a complex dependence relationship between a high number of random variables. We shall run this to tests in **Sections 3.1** and **3.2**.

2.2.3 Generating the joint cumulative distribution function

Algorithm 3: Constructing the joint cumulative distribution function **[11] [12]**

Data: A set of cumulative distribution functions u_i of univariate random variables x_i, and parameter θ for Archimedean copulas or correlation matrix R for the Gaussian

Result: A joint multivariate cumulative distribution function U

$d \leftarrow$ number of dimensions

1 $u \leftarrow u \setminus 0$

switch `family` **do**

case `clayton`:

2 $U \leftarrow 1 - \mathrm{d} + (\sum_{i=1}^{d} u_i^{-\theta})^{-1/\theta}$.

case `gaussian`:

3 $x \leftarrow \sum_{i=1}^{d} F_i^{-1}(u_i)$

4 $U \leftarrow F_{N(0,R)}(x)$

case `gumbel`:

5 $U \leftarrow \exp(-[\sum_{i=1}^{d}(-\ln u_i)^{\theta}]^{1/\theta})$

Where F^{-1} is the inverse cumulative distribution of a standard ($N(0,1)$) Gaussian distribution and $F_{N(0,R)}$ is the cumulative distribution function of a multivariate Gaussian distribution with mean vector 0 and covariance matrix R. From **Section 1.1**, a joint cumulative distribution function can be constructed by mapping a copula function to a set of marginal uniform distributions. d-dimensional copulas are of the form $[0,1]^d \rightarrow [0,1]$. Hence we need to transform the discrete count dataset into values between 0 and 1. Choosing cumulative distribution functions for this purpose is the most appropriate choice, as they are straightforward to compute for marginals and are always continuous on $[0,1]$.

If μ is high, the first few values of the marginal cumulative distribution function will be zero. But the definitions of copulas state that $C(\mathbf{u}) = 0$ if at least one coordinate of $\mathbf{u}$ is 0. Because the whole procedure is an approximate — albeit a close one, we made the decision to discard from computations, in line 1, all null values from $\mathbf{u}$ for

simplification purposes.
The process of mapping a copula function to a set of marginal distributions is different for each copula family;

1. with the Clayton family, on line 2, all marginals u_i are raised to the power of $-\theta$ and summed, before being raised to the power $-1/\theta$;
2. with the Gaussian family, the inverse cumulative distribution of a standard normal distribution mapped to every marginal u_i is summed, which is then plugged into a Gaussian distribution $F_{N(O,R)}$.
3. with the Gumbel family, the sum of the negative natural logarithm of every marginal is summed, raised to the power of θ and subsequently raised to the power of $^{1/\theta}$.

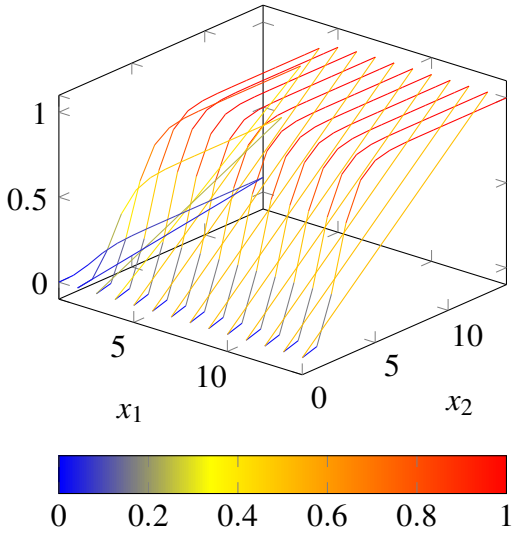

Figure 2.5: Two-dimensional joint cumulative distribution function of two discrete random variables x_1 and x_2 issued by a Clayton copula.

The joint cumulative distribution function in **Figure 2.5** joins two Poisson random variables x_1 and x_2 whose respective rates are $\lambda_1 = 2.1$ and $\lambda_2 = 4.3$. When $x_1 = 0$ and $x_2 = 0$, the joint distribution F_X is close to 0 (the blue area). Because of its low rate, the joint distribution quickly increases to 1 as $x_1 \to \infty$. On the other hand, the joint probability increases more steadily to 1 as $x_2 \to \infty$. A higher value for λ_1 would shift the curves to the right as Poisson variates are clustered around the mean value and rapidly decay on either side of λ. Applying the same logic for x_2, a higher value for λ_2 would make the curves less steep as the point where the curves hit $F_X = 1$ would come when $x_2 \gg \lambda_2$.

2.2.4 Generating the joint probability mass function

Algorithm 4: Constructing the joint probability mass function **[11]**

Data: A joint multivariate cumulative distribution function U and a set of d counts $\mathbf{x} = (x_1, ..., x_d)$.
Result: A joint probability mass function M

```
d ← number of dimensions
s ← number of values per dimension
m ← combinations of substrings from {0,1} of size s.
M ← {}
for i ← 0 to d do
    κ ← all substrings in m summing to i.
    for k ∈ κ do
        c ← (−1)^i · U(x_1 − k_1, ..., x_d − k_d)
        M ← M + c
    end
end
```

This protocol uses the **inclusion-exclusion principle** which is an important procedure in combinatorial mathematics first introduced by Poincaré and Sylvester. The formula to compute the probability mass function P_X of a joint cumulative distribution function X is given by **[11]**:

$$
\begin{aligned}
P_X(\mathbf{x}) &= P\left(A \setminus \bigcup_{i=1}^{d} A_i\right) \\
&= P(A) - \sum_{k=1}^{d} (-1)^{k-1} \sum_{I \subseteq \{1,\ldots,d\}, |I|=k} P\left(\cap_{i \in I} A_i\right) \\
&= \sum_{k=0}^{d} (-1)^{k} \sum_{\substack{m \in \{0,1\}^d, \\ \sum m_i = k}} F_X(x_1 - m_1, \ldots, x_d - m_d).
\end{aligned}
$$

Where A and A_i are defined as the sets $A = \{X_1 \leq x_1, ..., X_d \leq x_d\}$ and $A_i = \{X_1 \leq x_1, ..., X_d \leq x_d, X_i \leq x_i - 1\}$.
In other terms, to get the value of $P(X_1 = x_1, ..., X_d = x_d)$, which is the conjunction of every random variate x_i, the cumulative distribution function is calculated at these points, namely $P(X_1 \leq x_1, ..., X_d \leq x_d)$, and for each random variable X_i, the cumulative sum of all previous values of x from 1 up to $x - 1$ is truncated.
This method is an approximation and a slow one for the matter as it requires for every random variable X_i to compute the cumulative distribution function at every point from 1 to $X_i - 1$ and sequentially remove each. Potential performance improvements could be obtained by revising this algorithm with the use of dynamic programming (see the **Future Work** section for more).
However, as discussed in **Chapter 3**, this sacrifice in performance does not translate into a sacrifice in the accuracy of the fitting of a probability distribution.

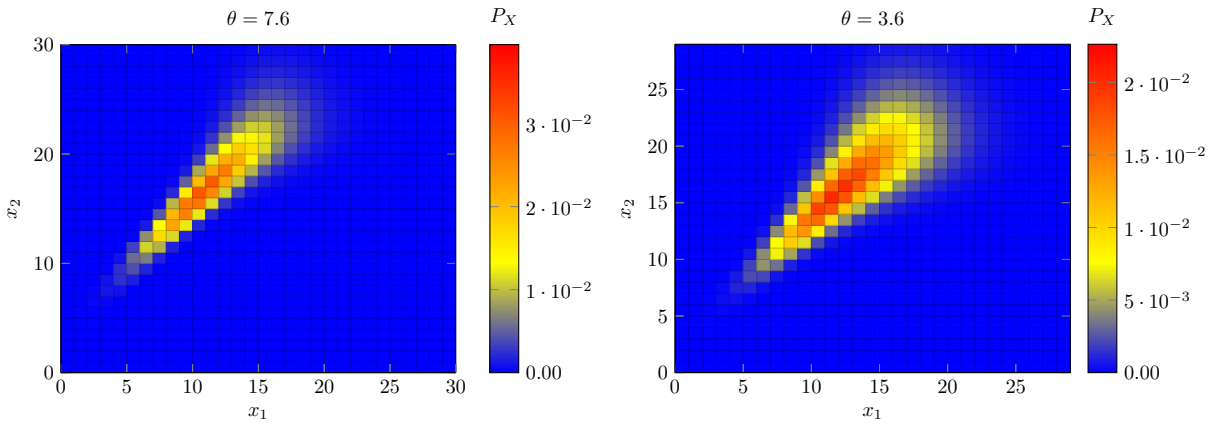

Figure 2.6: Two-dimensional joint probability mass function P_X generated from a Clayton copula with varying parameter θ and of two Poisson random variables x_1 and x_2 and their respective rates λ_1 and λ_2. In both cases, $\lambda_1 = 12.2$ and $\lambda_2 = 15.6$, and $Ran(x_1) = Ran(x_2) = [0, 30]$.

The fragmented aspect of the two figures is due to the data being discrete. For both probability mass functions in **Figure 2.6**, the sum over all samples is approximately 1, which is a good sanity check to run. The probability function P_X is considerably higher ($P_X \approx 0.03$) around the means of both random variables and P_X decays as $x_1 \to \pm\infty$ and $x_2 \to \pm\infty$, similarly to the phenomenon already discussed in **Section 2.2.1**. Furthermore, an interesting property of these joint probability mass functions is that their shape is similar to the one of the Clayton copulas in **Figure 2.2**: the higher value of θ has a narrower beam of points whereas the lower value of θ has a wider beam of points. This phenomenon further confirms the copulas' ability to regulate the level of correlation between random variables with its parameter value.
Once the joint probability mass function P_X has been computed, its log likelihood can be derived and then maximised in order to obtain the optimal copula parameters.

2.2.5 Optimal copula parameters estimation

First let:

$$\ell_i(\lambda_i) = \sum_{t=1}^{T} \log P_{X_i}(x_{i,t}; \lambda_i) \tag{2.1}$$

be the sum of log likelihoods of the marginal distributions $P_{X_i}(x_{i,t}; \lambda_i)$, with λ_i being the parameter of the Poisson distribution, and:

$$\ell(\theta, \lambda_1, ..., \lambda_d) = \sum_{t=1}^{T} \log P_X(x_t; \theta, \lambda_1, ..., \lambda_d) \tag{2.2}$$

be the log likelihood of the joint probability mass function $P_X(x_t; \theta, \lambda_1, ..., \lambda_d)$ where θ is the parameter of the chosen copula family.
Then we can apply the procedure called **inference for margins[11]** which proceeds in

two steps:
First, we separately maximise the Poisson rates λ_i:

$$\hat{\lambda}_i = \arg\max_{\lambda_i}\{\ell_i(\lambda_i)\} \tag{2.3}$$

Then we can estimate the copula parameter $\hat{\theta}$ as such:

$$\hat{\theta} = \arg\max_{\theta}\{\ell(\theta, \hat{\lambda}_1, ..., \hat{\lambda}_d)\} \tag{2.4}$$

Given a sample of data of size n with observed random vectors $y_1, ..., y_n$, the d log-likelihood functions for the univariate marginal distributions are defined as**[17]**:

$$L_j(\lambda_j) = \sum_{i=1}^{n} \log f_j(y_{i,j}; \lambda_j), \qquad j = 1, ..., d \tag{2.5}$$

where $\lambda_1, ..., \lambda_j$ are the parameters for the j Poisson random variables and $f_j(\cdot)$ is the j-th marginal cumulative distribution function.

Algorithm 5: Finding optimal parameters for the copula **[18]**

Data: A d-dimensional dataset D, an initial guess for the copula parameter θ^0 for Archimedean copulas or an initial guess for the covariance matrix σ^0 for the Gaussian, a patience value ω and a learning rate ψ.
Result: A predicted mean array $\hat{\lambda}$ and a predicted copula parameter $\hat{\theta}$ for Archimedean copulas or a predicted covariance matrix $\hat{\sigma}$ for the Gaussian.

$\hat{\lambda} \leftarrow \frac{\sum_{i=1}^{d} D_i}{d}$
$\hat{\theta} \leftarrow \theta^0$
$s \leftarrow$ number of data samples for each dimension
switch `family` **do**
  case `clayton` or `gumbel`:
    for $i \leftarrow 1$ to ω **do**
      $a \leftarrow$ `Nelder-Mead(log_likelihood, D,` $\hat{\theta}$, $\hat{\lambda}$`)` || `Powell(log-likelihood, D,` $\hat{\theta}, \hat{\lambda}$`)`
      if $|\hat{\theta} - a| < \psi$ **break**
      $\hat{\theta} \leftarrow a$
    end
  case `gaussian`:
    for $i, j \leftarrow 1$ to d **do**
      $\hat{\sigma}(i,j) \leftarrow \frac{\sum_{i=1}^{s}(D_x - \hat{\mu}_x)(D_y - \hat{\mu}_y)}{s-1}$

The optimal values for the rate $\hat{\lambda}$ and the covariance matrix $\hat{\sigma}$ in the case of the Gaussian copula are directly obtained from the data without going through an optimisation algorithm. For the covariance matrix, this is a rough estimate and is not the optimal solution as the marginal distributions have an effect on the correlations. However, for

the sake of conciseness, no alternative method will be discussed here.
The `log-likelihood` subroutine is the one described in Equation (2.2).
For Archimedean copulas, the algorithm starts with an initial guess θ_0 passed to either the Nelder-Mead or Powell's method alongside the `log-likelihood` function to be maximised, the data and the optimised $\hat{\lambda}$. The patience value ω represents the maximum number of iterations one is willing to wait until convergence. This is important because the Nelder-Mead method does not guarantee convergence, nor does it guarantee that a global minimum will be found. Indeed, runs of this procedure with large, complex and high-dimensional data will seldom fail; finding an optimal value for the patience, and with that balancing possibly scarce computing resources with giving time for a slow process to converge is a difficult task to undertake and would most likely resort to trial-and-error.

Powell's method is believed to be faster to converge**[18]** on average compared to Nelson-Mead, so a series of tests will be run to determine if this claim is true.

Chapter 3

Discussion of results

We want to measure how effective these algorithms are at fitting a joint Poisson distribution to a never-before-seen dataset. This fitted probability distribution would have a range of applications, such as predicting missing data in a large dataset. In some areas such as electronic commerce and other commercial applications, this would considerably lower the cost of acquiring a larger dataset to account for the inevitable missing values.
When working with artificially-generated data samples, we are in control of the ground truth variables and can therefore measure how far off the optimised parameters fitted to this data are compared to these ground truths parameters. Guiding these tests are two questions:

1. Does the number of dimensions impact the accuracy of the fitting?
2. Does the number of data items per dimension impact the accuracy of the fitting?

We shall run a series of tests to answer these questions. All tests are run in the School of Informatics' DICE environment. We shall generate artificial data using the methods mentioned in **Algorithm 1** with varying the number of data samples generated, the number of dimensions of the data and the copula parameter. To allow for reproducibility of tests, the same Poisson rates are used across copula families.

3.1 Results

Here is a table summarising the results. Each line represents a particular experience with set parameters. Each experience is run 50 times with different data as input then the mean squared error value is averaged to make sure the results are as representative as possible. The Kullbeck-Leibler divergence method is used to measure the accuracy of the fitting. The closer this value is to 0, the better. The convergence algorithm is the Nelder-Mead method, defined in **Section 1.4.1**.

3.1.1 Convergence analysis

N of dimensions	N of samples	$\lvert\hat{\theta} - \theta\rvert$	Kullbeck-Leibler Divergence Number
2	100	0.22	1.016229
2	200	0.19	0.752879
2	400	0.31	0.960898
2	800	0.17	0.782587
3	100	1.35	0.418339
3	200	1.12	0.311738
3	400	1.19	0.352515
3	800	1.09	0.254837
4	100	1.94	1.758551
4	200	1.78	1.496730
4	400	1.77	1.488376
4	800	1.77	1.466654

Table 3.1: Convergence values for the Gumbel copula

Both the Kullbeck-Leibler divergence number and $|\hat{\theta} - \theta|$ have really low values in all the situations tested, although there is to note that some extreme outliers have been washed out from the table of results — in some occasions, the algorithm does not convergence and the predicted value $\hat{\theta}$ becomes either a very high number or a negative number. This is due to the search space collapse problem that the convergence algorithm suffers from, as mentioned in **Section 1.4.2**.
Running the algorithm a large number of times and taking the median $\hat{\theta}$ can offset the issue. Another thing to note is the difference between the predicted $\hat{\theta}$ and θ decreases as the number of samples increases, particularly in the bivariate case: the predicted θ is 0.22 off with 100 samples compared to 0.19 with 200 samples and 0.17 off with 800 samples. However, it's more nuanced as the number of dimensions increases; the Kullbeck-Leibler divergence is always greater than 1 with both 3 and 4 dimensions, even though it decreases in both cases; the decrease is very slow: in the bivariate case there is a 22% decrease between the value of $|\hat{\theta} - \theta|$ with 100 samples and with 800 samples. This is compared to an 8% decrease between 100 samples and 800 samples in the 4-dimensional case. Similarly, in the bivariate case, there is a 22% decrease in the Kullbeck-Leibler divergence number compared to a 16% decrease in the 4-dimensional case. For the Gumbel copula, the fitness of the probability distribution seems to improve as the number of data samples increases but increase to stagnate as the number of dimensions increases.

N of dimensions	N of samples	$\lvert\hat{\theta}-\theta\rvert$	Kullbeck-Leibler Divergence Number
2	100	1.51	1.307709
2	200	0.99	0.233417
2	400	0.99	0.228676
2	800	0.30	0.263918
3	100	1.26	0.609518
3	200	1.15	0.312819
3	400	1.08	0.394686
3	800	1.09	0.394933
4	100	1.83	1.715789
4	200	1.65	1.561232
4	400	1.44	1.504178
4	800	1.44	1.506017

Table 3.2: Convergence values for the Clayton copula

In the Clayton case, similar results are obtained as for the Gumbel copula. The value for $|\hat{\theta}-\theta|$ decreases as the number of samples increases, from 1.51 for $n = 100$ samples to 0.30 for $n = 800$ — an 80% decline. Similarly, the Kullbeck-Leibler Divergence number decreases as the number of samples increases. This means that the fitted probability distribution with the optimised value for θ becomes closer to the original probability distribution. However, when the number of dimensions increases, both $|\hat{\theta}-\theta|$ and the Kullbeck-Leibler Divergence number increase and hover above 1.

N of dimensions	N of samples	Kullbeck-Leibler Divergence Number
2	100	0.960898
2	200	1.016229
2	400	0.782879
2	800	0.535411
3	100	1.045194
3	200	1.198687
3	400	0.752158
3	800	0.490546
4	100	1.371759
4	200	0.890719
4	400	0.796573
4	800	0.732853

Table 3.3: Convergence values for the Gaussian copula

The Gaussian case is slightly different; the Kullbeck-Leibler divergence number decreases gradually as the number of samples increases just like the Clayton and Gumbel

cases. However, the divergence number does not generally increase when the number of dimensions increases, contrarily to the two Archimedean copulas. When $d = 2$ and $n = 200$, $\chi_{Kullbeck-Leibler} \approx 1.01$ whereas when $d = 4$ and $n = 200$, $\chi_{Kullbeck-Leibler} \approx 0.89$, thus a 12% decline. This suggests that with high-dimensional data, the Gaussian copula is better at fitting multivariate distributions than the two Archimedean copulas. I reckon this is due to the high-dimensional correlation matrix σ, which represents the relationship between any two dimensions with a distinct number, therefore better accounting for complex, high-dimensional dependence structures. In the bivariate case, both Archimedean copulas and the Gaussian copula only use one number to represent the correlation between random variables, respectively θ and σ so the difference in the fitting is not obvious. However, for higher dimensions, although the Archimedean copula parameter is still one-dimensional, the covariance matrix of the Gaussian copula uses 3 numbers, σ_1, σ_2 and σ_3 to describe the relationship between the three random variables. The covariance matrix will look like this: $R = \begin{pmatrix} 1 & \sigma_1 & \sigma_2 \\ \sigma_1 & 1 & \sigma_3 \\ \sigma_2 & \sigma_3 & 1 \end{pmatrix}$ The same principle can be applied with an increasingly higher number of dimensions.

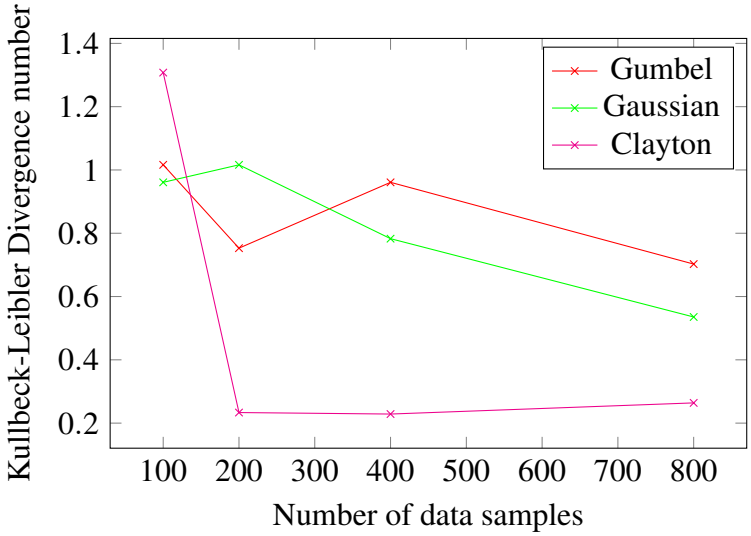

Figure 3.1: Evolution of the Kullbeck-Leibler Divergence number with respect to the number of data samples in the bivariate case. All copula families see a downward trend as the number of samples increases, however not all at the same rate.

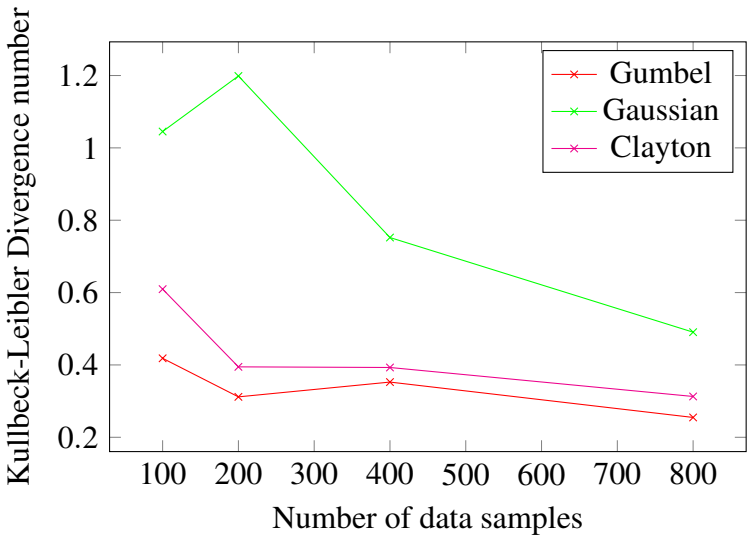

Figure 3.2: Evolution of the Kullbeck-Leibler Divergence number with respect to the number of data samples in the trivariate case. The Gumbel copula stagnates as the number of samples increases, when the other two see a downward trend.

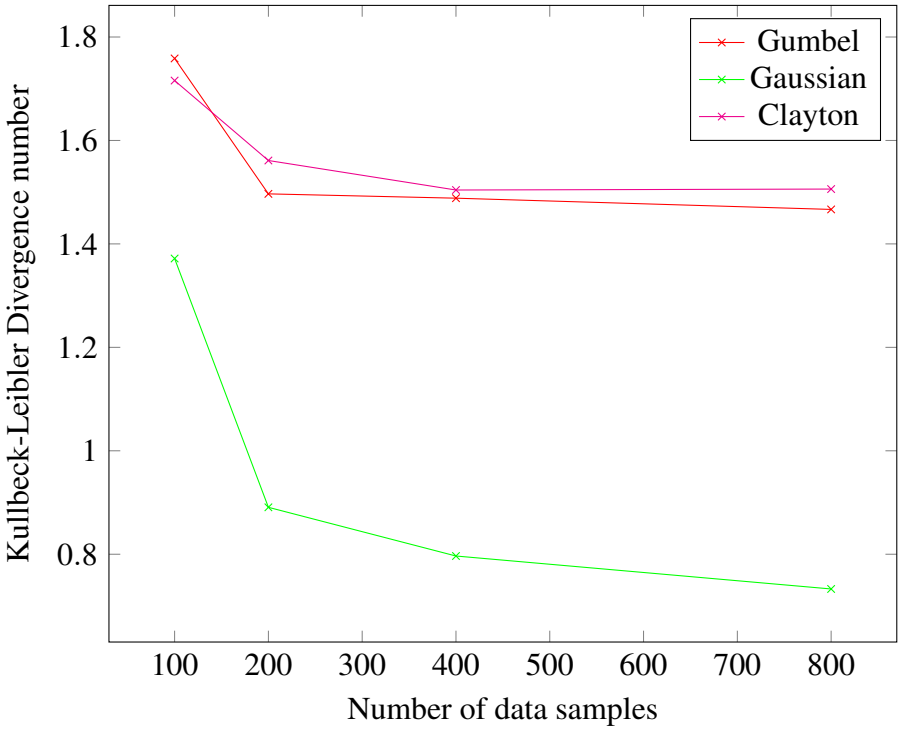

Figure 3.3: Evolution of the Kullbeck-Leibler Divergence number with respect to the number of data samples in the 4-dimensional case. The two Archimedean copulas have a consistently high Kullbeck-Leibler number, constrasting with the Gaussian copula case, which still sees a downward trend.

When plotting the data as line plots, it's clearer to understand and compare how all three copulas fare in these different conditions: the Gaussian copula fits a better prob-

ability distribution than the two Archimedean copulas when the number of dimensions is higher, with an average of 50% lower Kullbeck-Leibler divergence number in the 4-dimensional case. When the number of dimensions is low, all three copulas fit a better multivariate distribution when the number of samples is higher, with an average of 15% lower Kullbeck-Leibler number on all three copulas with 100 samples compared to 800. This is not surprising as a high number of data samples decreases the chances of overfitting.

3.1.2 Running time analysis

The method used to fit the probability distribution to the data is a slow one, as explained in **Section 2.2.4**, therefore optimising the time complexity of the method used for convergence is paramount in a presumably resource-scarce environment. On top of the two methods for computing and maximising the log likelihood detailed in **Section 2.2.5** — Nelder-Mead and Powell —, we introduce an additional method for a baseline model, which we shall call "Naive": given a learning rate ψ and starting value for $\hat{\theta}$ called θ_0 both provided by the user, the Naive method will increase the temporary value for $\hat{\theta}$ by ψ repeatedly until either the likelihood stops improving or the rate of increase is lower than a threshold called τ. This method is cumbersome for two reasons: first because this method, similarly to the previous two, does not use gradients and therefore has no way to learn optimal initial values for ψ, θ_0 and τ. Hence the user needs to resort to trial-and-error to guess these parameters which can take a long time to get right. For instance, a low value for τ will fool the Naive method into converging at a local maxima whereas a value too high will increase the chances of the method not converging. Finding the right balance would therefore be difficult to achieve. The second reason is that convergence is expected to be very slow as this method has no strategy to optimally select a maxima.
However this method should not pose any problem accuracy-wise in the controlled environment of this project, where all parameters for the marginals and the copula are known and well-defined. This method can therefore be a good estimate in whether either Powell or Nelder-Mead can improve the time it takes to convergence to optimal copula parameters.

Method used	N of samples	Time to convergence (s)
Nelder-Mead	200	21.03
Powell	200	19.65
Naive (baseline)	200	29.64
Nelder-Mead	400	44.16
Powell	400	40.38
Naive	400	98.72
Nelder-Mead	1000	291.30
Powell	1000	267.81
Naive	1000	811.08
Nelder-Mead	1400	1411.53
Powell	1400	1288.03
Naive	1400	5012.01

Table 3.4: Convergence running times for the Clayton copula

Method used	N of samples	Time to convergence (s)
Nelder-Mead	200	16.71
Powell	200	16.70
Naive	200	22.31
Nelder-Mead	400	50.31
Powell	400	53.64
Naive	400	86.22
Nelder-Mead	1000	231.11
Powell	1000	232.28
Naive	1000	722.52
Nelder-Mead	1400	1254.03
Powell	1400	1159.23
Naive	1400	4773.46

Table 3.5: Convergence running times for the Gumbel copula

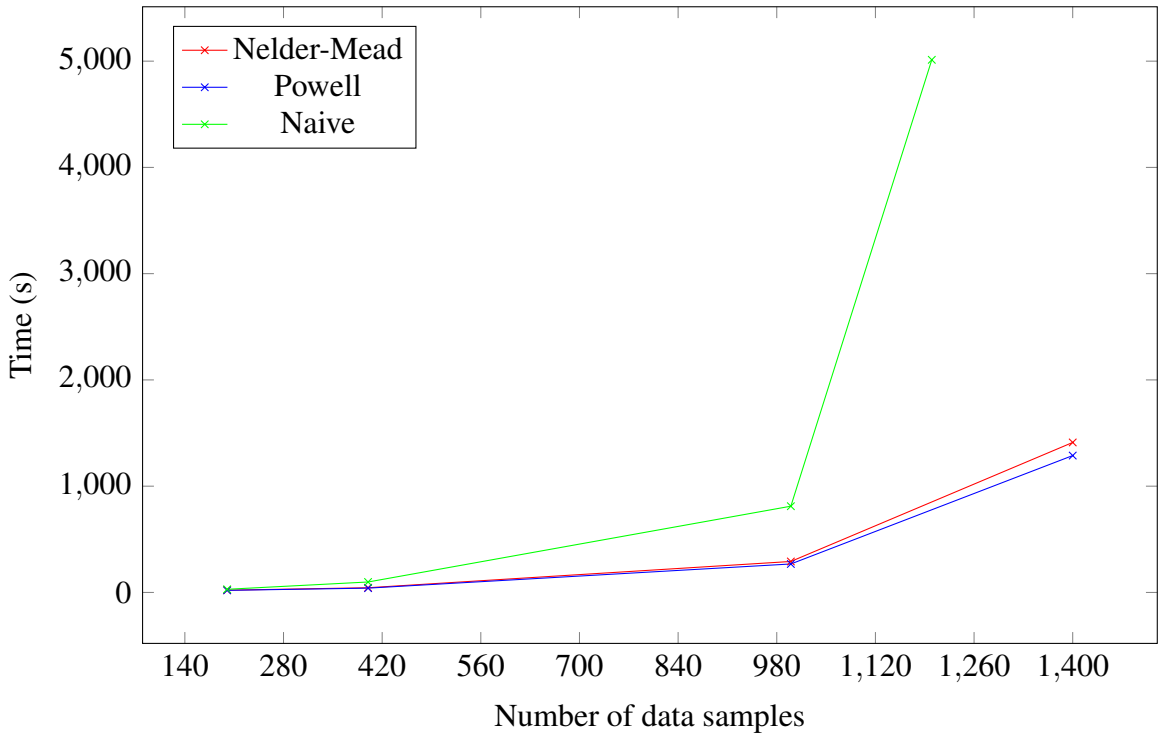

Figure 3.4: Evolution of the running time with respect to the number of data samples in the trivariate Clayton copula case. Powell's method takes slightly less time than Nelder-Mead, but the difference is not obvious without a very large number of samples. Both methods are bounded above by the Naive method.

The running time for all three methods grows fast as the number of data samples, and therefore the dimensionality of the data increases. Indeed, in the case where there are 200 data samples, the probability mass function is a 200x200 matrix with therefore 40,000 numbers to compute. However, with 1,400 data samples, the probability mass function is a 1400x1400 matrix with thus 1.96 million numbers to compute. On top of that, either method has to compute this large matrix at every iteration of the algorithm and without manual garbage collection control in Python, this can have an important impact on the computer's memory. Despite all that, both Nelder-Mead and Powell's method seem to fare relatively well in terms of running time as it doesn't grow exponentially, contrarily to the Naive method.

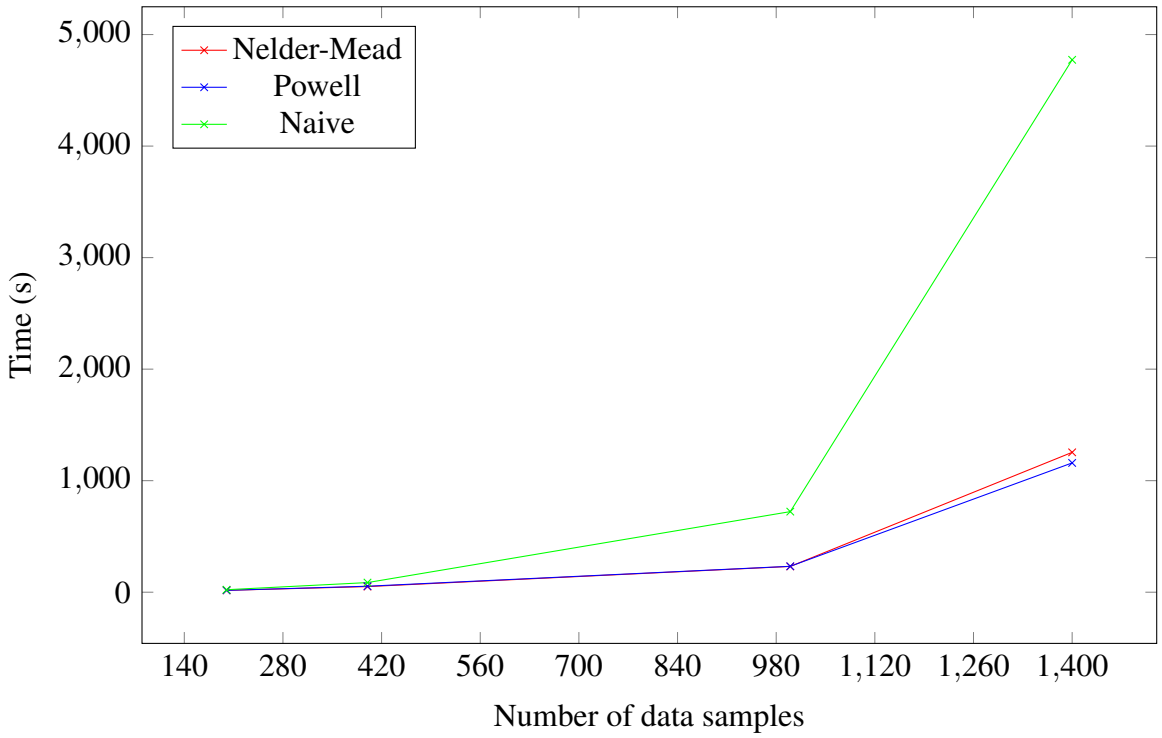

Figure 3.5: Evolution of the running time based on the method chosen for 3 dimensions and the Gumbel copula

With the Gumbel copula as well, we can make the case that Powell's method has a slightly lower running time that the Nelder-Mead method when the number of data samples is very high: from a 0.6% edge in favour of Powell's method for 200 samples up to a 7.6% edge when there are 1,000 samples. The gap between Powell's method and the baseline model is even wider — at around 75.7% —.
To put it all in a nutshell, Powell's method looks to be the most optimal convergence method for very high-dimensional data but the two methods are tied when the number of samples is low. In all cases, these two convergence methods fare much better than the Naive method.

Chapter 4

Conclusion

It is possible to construct a valid multivariate Poisson distribution by mapping a copula function to marginal univariate Poisson distributions, which are both individually straightforward to represent. We have implemented a Python package that contains functions to represent, manipulate and extrapolate multivariate Poisson distributions from count data:

1. generate copulas from the Gaussian, Clayton and Gumbel families with varying parameter value and dimensionality, and represent each graphically (**Section 2.2.2**),
2. construct a joint cumulative distribution function by computing the marginal distributions from a dataset that follows the data matrix model (**Section 1.2**), and applies the formula of each copula family (**Equations** (1.5), (1.8) and (1.11) and **Section 2.2.3**),
3. construct a joint probability mass function from the joint cumulative distribution function by applying Poincaré and Sylvester's inclusion-exclusion principle (**Section 2.2.4**),
4. find the copula parameter value that fits the most optimal multivariate probability distribution function by applying the inference for margins method (**Section 2.2.5**),
5. generate artificial data from a copula by using its generator function and the Poisson inverse cumulative distribution function (**Section 2.2.1**).

Each of these steps has been empirically tested, and in the light of convergence testings we have concluded that the distribution fitting works optimally with datasets with a large number of samples (over 800) and a low number of dimensions (less than 4). Furthermore, although both Nelder-Mead's and Powell's method are valid methods to optimise the parameters of the marginal distributions and the copula, Powell's method is the algorithm with the lowest running time for very large datasets.

Chapter 5

Future Work

Although much has been already done, there is still a considerable amount of possibilities to extend this project further and make it more robust, more adaptable and account for even richer and more complex dependence structures.
A first possible area to explore would be to extend the `Copula` class to account for the Frank copula. As was mentioned in **Section 2.2.2**, the Frank copula does not account for either positive or negative tail dependence. It is as such the only Archimedean copula displaying radial symmetry**[20]**, an interesting property worth exploring with some particular datasets. Alternative solutions to finding an approximation to a step function could be found.

We mentioned in **Section 2.2.4** that the algorithm constructing the joint probability mass function of several random variables was a slow process because of the application of the inclusion-exclusion principle. However, we believe there are ways to improve its performance by exploring some ways to avoid unnecessary calculations, a problem that has been solved before using dynamic programming**[21]**. This technique has been determined to have a much better time and space complexity, which would be especially beneficial with very high-dimensional datasets.

Finally, although the tests that have been run were thorough and robust, it would likely benefit from testing it with data-sets of lesser quality — with missing values and outliers and measure how well the optimised multivariate probability distributions fit this data and whether any of the three copulas reacts better at noisy data. An idea would be to extend **Algorithm 1** with the option to add a quantified amount of Gaussian noise to the data.

Chapter 6

Appendix

N dimensions	N of samples	Copula family	Kullbeck-Leibler Divergence
2	20	Clayton	0.82347
2	40	Clayton	0.65677
2	100	Clayton	0.25028
2	200	Clayton	1.56265
2	400	Clayton	0.53337
2	20	Gumbel	0.30529
2	40	Gumbel	0.30529
2	100	Gumbel	1.42251
2	200	Gumbel	1.56265
2	400	Gumbel	0.53337
2	20	Gaussian	0.30529
2	40	Gaussian	0.30529
2	100	Gaussian	1.42251
2	200	Gaussian	1.56265
2	400	Gaussian	0.53337
3	20	Clayton	0.39551
3	40	Clayton	0.90037
3	100	Clayton	0.31323
4	20	Clayton	0.36816
4	40	Clayton	0.71495
6	20	Clayton	0.56729
6	40	Clayton	0.78542
6	100	Clayton	0.32871

Table 6.1: ests run on datasets with a smaller number of samples

Bibliography

[1] David I. Inouye, Eunho Yang, Genevera I. ALlen, Pradeep Ravikumar, *A Review of Multivariate Distributions for Count Data Derived from the Poisson Distribution*, December 2016

[2] Pravin K. Trivedi and David M. Zimmer *Copula Modeling: An Introduction for Practitioners*, Foundations and Trends in Econometrics, 2007

[3] Roger B. Nelsen, *Properties and applications of copulas: A brief survey*, Department of Mathematical Sciences, Lewis Clark College

[4] Vassilis Hajivassiliou, Daniel McFadden and Paul Ruud, *Simulation of Multivariate Normal Rectangle Probabilities and their Derivatives: Theoretical and Computational Results*, Revised October 1994

[5] Giovanni De Luca and Giorgia Rivieccio, *Multivariate Tail Dependence Coefficients for Archimedean Copulae*, Department of Statistics and Mathematics for Economic Research, Parthenope University of Naples, December 2012

[6] *The Gaussian Copula and the Financial Crisis: A Recipe for Disaster or Cooking the Books?*, University of Oxford, June 2016

[7] Donald Mackenzie and Taylor Spears, *'The formula that killed Wall Street': The Gaussian copula and modelling practices in investment banking*, Social Studies of Science Vol. 44, No. 3 (June 2014)

[8] Coles, Stuart, *An Introduction to Statistical Modeling of Extreme Values*, Springer, 2001

[9] Marius Hofert, *Sampling Archimedean copulas*, Fakultät für Mathematik und Wirtschaftswissenschaften, Universität Ulm, 2008

[10] Rory Stark, Marta Grzelak and James Hadfield, *RNA sequencing: the teenage years*, Nature Reviews Genetics, 2019

[11] Arno Onken, Steffen Grünewälder, Matthias H. J. Munk, Kullbeck-Leibleraus Obermayer, *Analyzing Short-Term Noise Dependencies of Spike-Counts in Macaque Prefrontal Cortex Using Copulas and the Flashlight Transformation*, PLoS Computational Biology, November 2009

[12] Martin Haugh, *An Introduction to Copulas*, IEOR Quantitative Risk Management, 2016

[13] Andreas Svensson, Frederik Lindsten and Thomas B. Schön, *Learning Nonlinear State-Space Models Using Smooth Particle-Filter-Based Likelihood Approximations*, Department of Information Technology, Uppsala University, November 2017

[14] Jiawei Han, *Data Mining: Principles and Algorithms*, Department of Computer Science, University of Illinois

[15] M. J. D. Powell, *An efficient method for finding the minimum of a function of several variables without calculating derivates*, The Computer Journal, 7, 2, 1964.

[16] National Institute of Standards and Technology, *e-Handbook of Statistical Methods*, https://www.itl.nist.gov/div898/handbook/, Accessed on 28 March 2020

[17] Harry Joe, James Jianmeng Xu, *The Estimation Method of Inference Functions for Margins for Multivariate Models*, The University of British Columbia Library, October 1996

[18] Marco A. Luersen, Rodolphe Le Riche, *Globalized Nelder-Mead method for engineering optimization*, Elsevier, March 2004

[19] Gary G. Venter, *Tails of copulas*, Casuatly Actuarial Society, January 2002

[20] Nader Naifar, *Modelling dependence structure with Archimedean copulas and applications to the iTraxx CDS index*, Journal of Computational and Applied Mathematics, 35, 8, 2011.

[21] Richard M. Karp, *Dynamic programming meets the principle of inclusion and exclusion*, Operations Research Letters, 1, 2, 1982