Maël FORCIER

Promotion X2014

Research Internship Report

from April 10th to July 28th 2017

David L. Woodruff
University of California Davis
One shield avenue
Davis CA, 95 616 USA

Ecole Polytechnique
Département de Mathématiques appliquées
Optimisation stochastique
Stéphane Gaubert

Déclaration d'intégrité relative au plagiat

Je soussigné Maël Forcier certifie sur l'honneur :

1. Que les résultats décrits dans ce rapport sont l'aboutissement de mon travail.

2. Que je suis l'auteur de ce rapport.

3. Que je n'ai pas utilisé des sources ou résultats tiers sans clairement les citer et les référencer selon les règles bibliographiques préconisées.

Je déclare que ce travail ne peut être suspecté de plagiat.

Date ..... Signature

28/07/17

Abstract

In my internship at the University of California in Davis, I worked on a software that creates scenarios for the production of solar and wind renewable energy to use them in a stochastic optimization problem of unit commitment. This software is called `Prescient` *and it is used by researchers at Universities and the US Department of Energy and may eventually be used by grid operators. To do so, we need to analyze the actual powers and the errors to the forecast power that are produced and especially the dependence of these values between different hours of the day. I then coded in* `Prescient` *a complex tool that allow any user to work with multidimensional random variables and their distributions and especially to use copulas which are useful tools to capture the dependence between random variables without focusing on the univariate distributions of the coordonates called marginals. I explain in this report how this code is structured, the way it was designed and how to use it. Then, I used this code to make some experiments with real data of wind energy production. Inspired by the litterature and the previous interns, I created a method and an algorithm that permits to check how well a copula fits the datas. The mathematical aspects of this method and the algorithm are also presented here. I eventually present, interpret and comment the results I obtained. All copulas seem to fit pretty well the data except for the independence copula which does not fit them at all as we expected.*

Résumé

Lors de ce stage à l'Université de Californie de Davis, j'ai travaillé sur un projet de logiciel qui crée des scénarios de production d'énergies renouvelables solaires et éoliennes pour les utiliser dans un problème d'optimisation stochastique. Ce logiciel nommé `Prescient` *est utilisé par le US Department of Energy Pour y parvenir, nous devons analyser les données des puissances mesurées réellement ainsi que les erreurs ou différences par rapport aux prédictions et en particulier les corrélations de ces puissances à différentes heures de la journée. J'ai alors codé dans* `Prescient` *un outil complexe qui permet à n'importe quel utilisateur de manipuler des variables aléatoires à plusieurs dimensions et leurs distributions de probabilité et surtout d'utiliser des copules qui sont des outils utiles pour étudier la dépendance entre des variables aléatoires sans se préoccuper des distributions unidimensionnelles de leurs coordonnées appelées marges. J'explique dans ce rapport comment le code est structuré, la façon dont il a été pensé et comment l'utiliser. J'ai ensuite utilisé ce code pour effectuer des expériences avec des données réelles de production d'énergie éolienne. Inspiré par la littérature et par mes prédécesseurs stagiaires, j'ai imaginé une méthode et un algorithme qui permettent de vérifier à quel point une copule décrit bien les données. Les aspects mathématiques de cette méthode et cet algorithme sont aussi présentés ici. Enfin, je présente, commente et interprète les résultats obtenus. Toutes les copules semblent bien décrire les données sauf la copule d'indépendance qui est très mauvaise comme attendu.*

Acknowledgments

- I would like to give sincere thanks to my mentor Professor David L. Woodruff, who guided me a lot for this job and was always a great help during this internship.
- I also want to thank my teacher, Professor Stéphane Gaubert who made me like optimization and helped me find this internship.
- I eventually want to thank all my colleagues, Sebastian Angerhausen, Antoine Keller and Dominic Yang. It was a pleasure to work with them.

Contents

Introduction

Matching the supply and the demand of energy and in particular electricity is a complex problem for the grid operators. It is usually solved in part using what is called unit commitment. Unit commitment is an optimization problem where the grid operator has to decide whether to turn on or off each unit of electricity production so that the demand is matched. That means providing enough energy so that every client can use the energy he wants and not providing too much energy which will implies insecurity on the network as well as a loss of energy and money. Unit commitments were historically deterministic problems where the production and the demand are reliable forecast. Nevertheless, in the last decade, the production of electricity with renewable energies especially wind and solar energies has significantly increase in California and all over the world. These new sources are intermittent and less predictable, unit commitment problems then become stochastic. The more the part of solar and wind is in the electricity generation is, the harder it will be to forecast the production and to take the best unit commitment decisions.

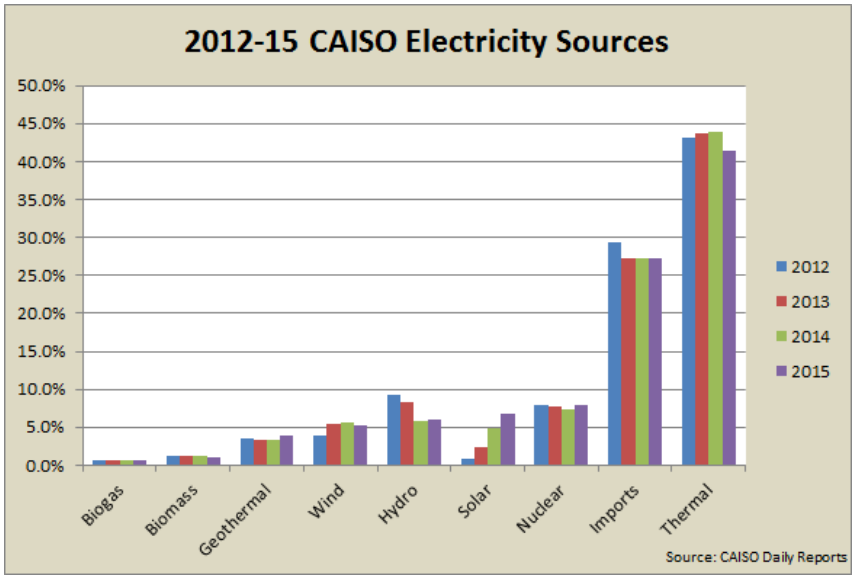

Figure 1: Electricity production from different sources between 2012 and 2015

Professor David L.Woodruff from the University of California at Davis and his team are working on a software in python called `Prescient` providing solutions for stochastic unit commitment problems for electricity grid management companies such as California Independent System Operator (CAISO) or power marketing agencies such as Bonneville Power Association (BPA). In order to have accurate and efficient solutions to stochastic optimization problems, instead of a deterministic forecast, we have to create different scenarios with their probabilities to happen. The project I was working in was called Grid Operation Scenario Maker (GOSM). Its goal was to provide relevant scenarios and probabilities that could be used by the stochastic optimization algorithm of prescient.

When I started my internship, GOSM was already able to create scenarios of wind and solar energy production by focusing on the errors between the forecast power and the actual power that was measured. The probabilities were calculated thanks to error distributions at different hours of the day called day part separators. However, the random variables representing the different sources at different dps were considered independent because it was first easier to code. But, it seems obvious that the energy production or the errors between two adjacent hours are highly correlated. Moreover, nothing tells us that the energy production of wind and solar are not correlated. Because GOSM had already integrated the distribution of this variable, the best way to modelize the dependance between variables without caring about the distribution of each variable which we call marginals is to use copulas. Copulas are useful mathematical tools which only describes the dependance between variables and which do not depend of marginals. My work in the GOSM team was to implement different kinds of copulas in the software and to test on various datas how well they modelize the dependence between the datas. In this report, I will first explain how I built this distribution and copula library for GOSM in python. Then, I will present the experiment I did to verify how copulas fit the datas. I will finally show and analyze the result I found.

1 Distribution and copula library

The first part of my job was to create copula classes that were easy and intuitive to use for other users such as the next people who will work on GOSM. At the beginning of my internship only 2 distribution classes(`UnivariateEmpiricalDistribution`, `UnivariateEpiSplineDistribution`) were implemented. To make verifications and control tests, I had to implement other classical distributions in addition to copulas. At the end, I had a great and pretty complete library of distributions and copulas with a relevant object oriented structure. I wrote a user manual to explain how this library works. This section sums up this user manual. If more details are needed, you can find the whole formulas in the Appendix A, B and C.

1.1 Mathematical Definitions

Definition 1. *The cumulative density function (cdf) of a random variable $X \in \mathbb{R}$ is the function $F : \mathbb{R} \to [0,1]$ where :*

$$F(x) = \mathbb{P}(X \leq x)$$

This definition also works when X is a multidimensional random variable i.e. $\in \mathbb{R}^d$ if we use the notation $(x_1, ..., x_d) \leq (y_1, ..., y_d) \Leftrightarrow \forall i, x_i \leq y_i$

Definition 2. *If the cdf of $X \in \mathbb{R}$ is differentiable, the probability density function (pdf) of X is the function $f : \mathbb{R} \to [0,1]$ where :*

$$f(x) = \frac{dF(x)}{dx} = \frac{d\mathbb{P}(X \leq x)}{dx}$$

If $X \in \mathbb{R}^d$ and F is regular enough, the probability density function pdf of X is the function $f : \mathbb{R} \to [0,1]$:

$$f(x) = \frac{d^d F(x)}{dx_1 ... dx_d}$$

Each one of these two functions gives all the informations about how the random variable X behaves. With them, we can compute different charasteristic values of the distribution.

Definition 3. *The mean of a random variable $X \in \mathbb{R}^d$ is :*

$$\mathbb{E}(X) = \int_{\mathbb{R}^d} x f(x) dx$$

Definition 4. *The variance of a random variable $X \in \mathbb{R}^d$ is :*

$$Var(X) = \mathbb{E}((X - \mathbb{E}(X))^2) = \int_{\mathbb{R}^d} (x - \mathbb{E}(X))^2 f(x) dx$$

Property 1. *If $X \in \mathbb{R}$ is a random variable with a continuous cdf F,*

then $U = F(X)$ is a uniform random variable in [0,1]

1.2 Distributions

To model the random properties of data (for example the production of wind power or wind forecast errors), we define distribution as a python object that contains several methods. This distribution python object does not correspond strictly to the mathematical definition of a distribution because it contains several different methods that describe many things linked to the probability law of the variable. To simplify, when we will talk about distributions, it will refer to the python objects or classes that will inherit from `BaseDistribution` class.

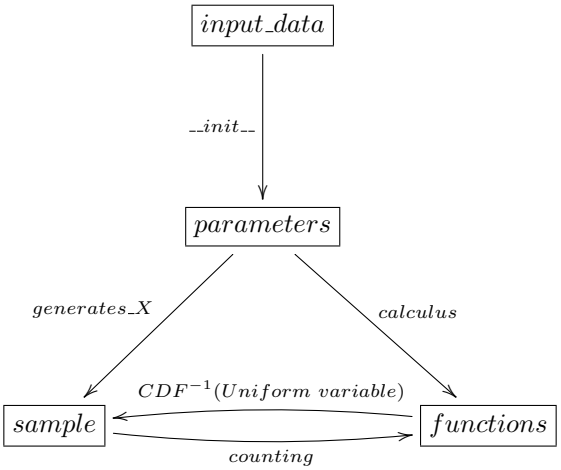

Figure 2: Simplified diagram of how a distribution object works

1.2.1 Methods and parameters of a distribution object

The cdf, pdf and other functions The methods cdf and pdf compute the two chararacteristic functions defined above. If the object `distrobject` represents a random variable X with cdf F and pdf f, `distrobject.cdf(x)` will return the value of $F(x)$ and `distrobject.pdf(x)` will return the value of $f(x)$. Most of the time, this value are calculated using analytical formulas.

For example, the pdf of a normal distribution is calculated using the formula

$$f(x) = \frac{1}{\sqrt{2\pi}} exp(\frac{(x-\mu)^2}{2\sigma^2})$$

To make others calculus and depending on the class of the object, one might need other functions linked to the cdf and the pdf. For example, the method `cdf_inverse` will compute the inverse of the cdf. `distrobject.cdf_inverse(x)` will return the value $F^{-1}(x)$.

The mean, var and other parameters One can see that these formulas depend on parameters. In the previous normal case, they are μ and σ.

Althought the mean and the variance can be interesting values that we calculate using the cdf or the pdf. They also can be considered as parameters. In the normal case, μ is also the mean and σ^2 the variance. If the object `distrobject` represents a random variable X, `distrobject.mean` will return the value of $\mathbb{E}(x)$ and `distrobject.var` will return the value of $Var(x)$.

Depending on the class of the object, it could have other parameters that would be explained in detail while presenting the different classes. Examples include the degree of freedom (`df`) for student distributions, the covariance matrix (`cov`) for multivariate distribution or the theta parameters (`theta`) for archimedian copulas.

`__init__` and `input_data` The method __init__ creates a new object of a distribution class. It can be called either by the speficic parameters of the distribution or with a set of data called input_data. For instance, $distrobject = UnivariateNormalDistribution(mean = mu, var = sigma**2)$ will return an object with the parameters μ for mean and σ^2 for var both stored as the attributes `distrobject.mean` and `distrobject.var`. The kinds and the names of the parameters change according to the class.

But, in practice, it is very usefull to fit a distribution to a data set. By default, the parameters (such as mean and var in the normal case) are set as `None`. Then, if the parameters are not specified they will be calculated using a data set. For instance, `UnivariateNormalDistribution(input_data = Y)` will return an object with the paremeters
$\frac{1}{n}\sum_{k=1}^{n} Y_k$ for mean and $\frac{1}{n}\sum_{k=1}^{n} Y_k^2 - (\frac{1}{n}\sum_{k=1}^{n} Y_k)^2$ for var.

So `__init__` fits the best distribution possible considering the class called and `input_data`.

`generates_X` and sample A distribution object modelizing a random variable X also has a method that can generates realisations of X following the probability law.

For instance, in the normal case we call the numpy function `numpy.random.randn` with the correct parameters to generates such variables. In the case of univariate distribution it can also be obtained by generating a uniform variable in [0,1] and composing by the inverse of the cdf. $F^{-1}(U)$, where U is uniformly distributed on [0,1], follows the same probability law as X.

Because sometimes generating a lot of variables can be time consuming, they are stored automatically in `distrobject.X`. Then, `distrobject.generates_X(n)` returns a vector of n independent realisations of X and store them in the attribute `mydistrobject.X` by appending this n new values by appending it to the old attribute.

1.2.2 Files and classes

In order to make comparisons and find the best distribution model for a variable, I implemented several different distributions. Even if they follow the same global pattern described above, their parameters, their functions and the way they are implemented can be very different because of a need to speed up the computation or simply because the mathetical objects are really different. All the distributions are coded with classes that one can find in a certain file. I explain in this section how the files are organized and how the classes work. One can find in appendix A the whole list of files and classes.

base_distribution.py This file contains 2 abstract classes that define what all distributions have in common :

`BaseDistribution` is the global class from which all the distributions classes will inherit. It contains different methods such as the cdf and the pdf which can be overwritten according to the subclass.

`MultiDistr` inherits from `BaseDistribution`. It is the abstract class from which all the distributions with more than one variable will inherit. The particularity of multivariate distribution is that it works with dictionnaries. So the argument of its methods should always be dictionnaries unlike univarate distribution whose input datas can be list or vectors. This helps avoid confusion concerning the different coordonates of the distribution and facilitates meaning for error messages. To keep the same keys in the same order, a MultiDistr object has a parameters an attribute `dimkeys` that specify the list of the names of the coordonates. The user must always call a `MultiDistr` object by inputing a list of dimkeys in its arguments. This abstract class also contains the method `rect_prob` that computes the probability for the random variable to be in a n dimension rectangle defined by two opposed points `lowerdict` and `upperdict`. Even thaugh this calculus requires a non trivial recursion, it only needs the cdf to be computed and it is the same calculus for all multivariate distribution.

distributions.py This file contains all the concrete classes of the classical distributions that are not built with copulas. If they are called Univariate, they directly inherit from `BaseDistribution`, if they are called Multi they inherit form `MultiDistr`. Most of them are classical distributions such as gaussian or uniform distribution but some are special as we use them in particular purposes. One can find a detailed explanation of each of these class on appendix B.

copula.py This file contains the abstract class `CopulaBase` which inherits from `MultiDistr` and from which all the concrete copula classes inherit. The mathematical details of copulas are explained in the next sections. As was the case with our distribution objects and classes in python, our copula objects and classes in python are different from the strict mathematical definition of a copula which is just what we call a C function. The inheritance from `MultiDistr` is very convenient in our applications, but is non-standard for copula classes.

`CopulaBase` is the abstract class that contains the attributes and methods that all the copula classes have in common. As an instance of MultiDistr, a copula object is called with a `dimkeys` argument and an `input_data` argument and both are stored as attributes with the same names. A copula object has another argument called `marginals` which is a dictionary of univariate distribution with `dimkeys` as keys. The `marginals` are needed because of the inheritance from `MultiDistr`. The marginals dictionary is directly stored in an attribute called `marginals`. Having this univariate distribution dictionary permits separation of the distribution of each coordonates and the dependence. The method is to use what we

presented as property 1 on each marginal. Instead of working on the dependence between X_1, X_2 ... and X_n, we prefer to focus on the dependence between $U_1 = F_1(X_1)$, $U_2 = F_2(X_2)$ ... and $U_n = F_n(X_n)$. It is now much easier to compare the dependence between coordonates that all have the same distribution which is uniform in [0,1]. We work in what we call the U-space, $[0,1]^d$, as opposed to the X-space, $\mathbb{R}^d$. One can easily pass from one to the other by composing coordonate by coordonate with the F_i or by using the formulas presented in the next sections. Like all the children of `MultiDistr` a copula object has methods and attributes in the X-space, but it contains also its twins in the U-space, just like if a copula object contained the distribution of X and the distribution of U. Considering this analogy, C (as the cdf of U) plays in the U-space the role of cdf in the X-space. c (as the pdf of U) plays in the U-space the role of pdf in the X-space. Finally, `generates_U` permits to return samples of U with the good dependence, it plays in the U-space the same role as `generates_X` in the X-space.

vine.py Vine copulas are copulas that are built using other bivariate copulas. I explain them with more details below and in the Appendix C.

distribution_factory.py In all the previous files, each definition of a new concrete class begins with `@register_distribution(name="name-of-the-class")`. This permits giving a string name to each class. The file distribution_factory.py contains the code that allow us to write :

```
distr_class = distribution_factory(copula_string)
distr_object = distr_class(input)
```

instead of

```
if copula_string =="univariate-uniform":
   distr_object = UnivariateUniformDistribution(input)
if copula_string =="univariate-normal":
   distr_object = UnivariateNormalDistribution(input)
```

etc, for all the cases.

This pratical way of selecting the classes is also used to build distributions that are created using others such as combined copulas or vine copulas.

tester.py This file contains several unittests to check the distribution classes. Some are just quick tests to check if the code runs. But most of them attempt to verify if the implemented formulas are correct. In order to do that, the method tries to obtain the same result by different ways or by making a loop on diagram that should be commutative like the one in figure 1. For instance, `test_pdf_cdf` integrates the pdf to compare it to the cdf and differentiate the cdf to verify if it is equal to the pdf. `test_c_with_C_2_dim` integrates twice `c` and verify if the value is equal to `C`. `test_C_with_sample` checks the low loop of diagram in figure 1 in the U-space : with a big data set created by `generates_U`, it creates an empirical cdf of U (called `C_from_sample`) and compares it to the actual cdf of U which is `C`. Another example is `test_with_gaussian_copula_3_dim` which checks if making a distribution with normal (or gaussian) marginals and a gaussian copula give the same result than a creating directly a multinormal (or multigaussian) distribution.

1.3 Copulas properties

In order to study the dependance of several correlated random variables, we need to focus on a tool called copula. It is more practical to consider our different variables together as a multidimensional variable. The distribution univariate random variables are then called marginals. Copulas permit expression of the dependance separated from the marginals. In our object oriented program, it is really practical because one can work on the dependance with the copula while the marginals are just inputs. To introduce and define the copulas, let us quote Sklar's theorem :

Theorem 1. *Sklar, 1959. Let us consider a random vector $X = (X_1, ..., X_d)$ with a cumulative distibution function $F(x_1, ..., x_d) = \mathbb{P}(X_1 \leq x_1, .., X_d \leq x_d)$.*
Then, exists a function C called copula $C : [0,1]^d \to [0,1]$ so that :

$$F(x_1, ...x_d) = C(F_1(x_1), ..., F_d(x_d))$$

where F_i is the cumulative density function of the ith marginal.
The copula C is unique if the marginals are continuous.

We will also use the density of the copula :

Definition 5. *The copula density of a copula C is the function $c : [0,1]^d \to \mathbb{R}^+$:*

$$c(u,v) = \frac{\partial^d C}{\partial u_1 ... \partial u_d}(u,v)$$

By differentiating, we obtain the equation :

$$f(x_1,...,x_d) = c(F_1(x_1),...,F_d(x_d))f_1(x_1)...f_d(x_d)$$

Definition 6. *The Kendall distribution function $\mathcal{K}_F : [0,1] \to [0,1]$ of a distribution of cumulative density function F is :*

$$\mathcal{K}_F(u) = \mathbb{P}(F(X) \leq u)$$

where X is a random variable following the distribution defined by F.

$$\begin{aligned}
\mathcal{K}_F(u) &= \mathbb{P}(F(X) \leq u) \\
&= \int_{\mathbb{R}^d} \mathbb{1}_{F(x) \leq u} f(x) dx \\
&= \int_{\mathbb{R}^d} \mathbb{1}_{C(F_1(x_1),...,F_d(x_d)) \leq u} c(F_1(x_1),...,F_d(x_d)) f_1(x_1)...f_d(x_d) dx \\
&= \int_{[0,1]^d} \mathbb{1}_{C(y) \leq u} c(y) dy \\
&= \int_{K_u} c(y) dy \text{ , where } K_u = \{y \in [0,1]^d, C(y) \leq u\} \\
&= \mathbb{P}(C(U) \leq u)
\end{aligned}$$

Where U is a random variable on $[0,1]^d$ with cdf C (and uniform marginals).

Intuitively, when we do the changing of variable $y_i = F_i(x_i)$:
We have $dy_i = F_i'(x_i)dx_i = f_i(x_i)dx_i$. So, indeed $dy = f_1(x_1)...f_d(x_d)dx$.
The meticulous proof with the Jacobian determinant gives the same result.

$\mathcal{K}_F$ only depends on the copula C of the distribution function.
Thus, we will use the notation $\mathcal{K}_C$.
If we write $\Gamma_{u_2,...,u_d}(x) = C(x, u_2, ..., u_d)$, we also have :

$$\mathcal{K}_C(u) = \int_0^1 ... \int_0^1 \left(\int_0^{\Gamma^{-1}_{y_2,...,y_d}(u)} c(y) dy_1 \right) dy_2 ... dy_d$$

We want to use Vine copulas to study multidimensional dependance. To compute such vine copulas, we need to have the partial derivative of C function and its inverse. For each of the following copulas in 2 dimensions, I calculated the partial derivate $h(u,v,\theta) = \frac{\partial C_\theta}{\partial v}(u,v)$. I also calculated its inverse h^{-1} considering its first variable : $h(h^{-1}(u,v,\theta),v,\theta) = u$

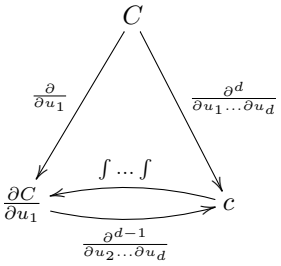

Figure 3: Relations between copula functions

The following subsections present briefly the different kinds of copulas I dealt with. To have more detail about the formulae, one can refer to appendix C.

1.4 Elliptical Copulas

Elliptical copulas are copulas of multidimensional vectors with a elliptical distribution.

Definition 7. *According to [5], a random vector* $\mathbf{X} \in \mathbb{R}^d$ *has an elliptical distribution if it can be written :*

$$\mathbf{X} = \mu + \rho A \mathbf{U}$$

where $\mu \in \mathbb{R}^d$
ρ *is a random positive univariate variable*
$A \in \mathbb{R}^{d \times d}$
$\mathbf{U}$ *is uniformly distributed on the unit sphere of* $\mathbb{R}^d$
ρ *and* $\mathbf{U}$ *are independent*

Definition 8. *The covariance matrix of a random vector* $\mathbf{X} = (X_1, ..., X_d) \in \mathbb{R}^d$ *is the matrix* K *where :*

$$K_{i,j} = Cov(X_i, X_j) = \mathbb{E}((X_i - \mathbb{E}(X_i))(X_j - \mathbb{E}(X_j)))$$

The correlation matrix of a random vector $\mathbf{X} = (X_1, ..., X_d) \in \mathbb{R}^d$ *is the matrix* $\tilde{K}$ *where :*

$$\tilde{K}_{i,j} = \frac{Cov(X_i, X_j)}{\sqrt{Var(X_i)Var(X_j)}}$$

Confusing this two matrix can easily lead to mistakes. Unlike multivariate distribution, when using copulas, we do not need to take care of the variance of each marginal, we prefer to let this information in the `marginals` dictionary so we will prefer the correlation matrix.

1.4.1 Gaussian Copula

The gaussian copula of correlation matrix K is the copula of a gaussian vector $\mathcal{N}(0, K)$.

Because one can change the variance of each marginal, we will only consider covariance matrix where there are only 1 in the diagonal.

We will note $\mathcal{N}(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{t^2}{2}} dt$, the cumulative density function of the standard normal distribution and $\mathcal{N}^{-1}$ its inverse.

C function in dimension d

$$C_K(u_1, ...u_d) = \int_{-\infty}^{\mathcal{N}^{-1}(u_1)} \dots \int_{-\infty}^{\mathcal{N}^{-1}(u_d)} \frac{1}{(2\pi)^{\frac{d}{2}} \sqrt{det(K)}} e^{-\frac{x^\top K^{-1} x}{2}} dx$$

With a change of variable, we have also

$$C_K(u_1,...u_d) = \int_0^{u_1} ... \int_0^{u_d} \frac{1}{\sqrt{det(K)}} e^{-\frac{\mathcal{N}^{-1}(x)^\top (K^{-1}-I_d)\mathcal{N}^{-1}(x)}{2}} dx$$

where $\mathcal{N}^{-1}(x) = \begin{pmatrix} \mathcal{N}^{-1}(x_1) \\ \vdots \\ \mathcal{N}^{-1}(x_d) \end{pmatrix}$

1.4.2 Student Copula

The student copula of correlation matrix K and of degree of freedom ν is the copula of a student random vector $t_\nu(0, K)$.

Because one can change the variance of each marginal, we will only consider covariance matrix where there are only 1 in the diagonal.

C function in dimension d

$$C_K(u_1,...u_d) = \int_{-\infty}^{t_\nu^{-1}(u_1)} ... \int_{-\infty}^{t_\nu^{-1}(u_d)} \frac{\Gamma(\frac{\nu+d}{2})}{\Gamma(\frac{\nu}{2})(\nu\pi)^{\frac{d}{2}}\sqrt{det(K)}} \left(1 + \frac{1}{\nu} x^\top K^{-1} x\right)^{-\frac{\nu+d}{2}} dx$$

With a change of variable, we have also

$$C_K(u_1,...u_d) = \int_0^{u_1} ... \int_0^{u_d} \frac{\Gamma(\frac{\nu+d}{2})\Gamma(\frac{\nu}{2})^{d-1}}{\sqrt{det(K)}\Gamma(\frac{\nu+1}{2})^d \Pi_{j=1}^d 1 + \frac{t_\nu^{-1}(x_j)^2}{2})^{-\frac{\nu+1}{2}}} \left(1 + \frac{1}{\nu} t_\nu^{-1}(x)^\top K^{-1} t_\nu^{-1}(x)\right)^{-\frac{\nu+d}{2}} dx$$

where $t_\nu^{-1}(x) = \begin{pmatrix} t_\nu^{-1}(x_1) \\ \vdots \\ t_\nu^{-1}(x_d) \end{pmatrix}$

1.5 Archimedean Copulas

Archimedean copulas take the form :

$C(u_1, ..., u_d) = \Phi(\Phi^{-1}(u_1) + ... + \Phi^{-1}(u_d))$

where Φ is the Laplace transform of a positive nonzero random variable Y : $\Phi(u) = \mathbb{E}(e^{-uY})$

Φ is called the generator function.
It generally depends on a parameter called θ.
In some papers, the definition of the generator function and its inverse can be inverted. We will use the definition above which gives us simpler formulas.

All of the next formulas can be expressed with the generator function.
For example, if we refer to [1], we can write the Kendall distribution function thanks to the generator function :

$$\mathcal{K}_C(x) = \begin{cases} \frac{(-1)^{d-1}(\Phi^{-1}(0))^{d-1}}{(d-1)!}\Phi_-^{(d-1)}(\Phi^{-1}(0)) & \text{if } x = 0 \\ \sum_{k=0}^{d-2} \frac{(-1)^k(\Phi^{-1}(x))^k \Phi^{(k)}(\Phi^{-1}(x))}{k!} + \frac{(-1)^{d-1}(\Phi^{-1}(x))^{d-1}\Phi_-^{(d-1)}(\Phi^{-1}(x))}{(d-1)!} & \text{if } x \in]0,1] \end{cases}$$

1.5.1 Frank Copula

C function in dimension 2 $C_\theta(u,v) = -\frac{1}{\theta} log(1 + \frac{(e^{\theta u}-1)(e^{\theta v}-1)}{e^{-\theta}-1})$

1.5.2 Gumbel Copula

C function in dimension 2 $C_\theta(u,v) = e^{-((-log(u))^\theta+(-log(v))^\theta)^{\frac{1}{\theta}}}$

1.5.3 Clayton Copula

C function in dimension 2 $C_\theta(u,v) = (max\{u^{-\theta}+v^{-\theta}-1;0\})^{-\frac{1}{\theta}}$

1.6 Empirical Copula

Having a sample of multivariate random variables, we can define the finite empirical distribution of this sample where each realisation of the sample is equiprobable. The empirical copula is the copula of this empirical distribution. Contrary to the other ones, the empirical copula depends on its marginals.

$$C(u_1,...,u_d) = \sum_{k=1}^{n} \mathbb{1}_{\forall i, F_i(x_k)\leq u_i}$$

where, F_i is the cdf of the marginal i.

1.7 Vine Copulas

Vine copulas are copulas that are built using multiple bivariate copulas. One can intuitively understand them considering random variables are just correlated two by two like they were the node of a tree or the parts of a chain. Everything is explained in details in [2].

1.7.1 Canonical Vine Copula

Global probability density function

$$f(x_1,...,x_d) = \prod_{k=1}^{d} f(x_k) \prod_{j=1}^{d-1} \prod_{i=1}^{d-j} c_{j,j+i|1,...,j-1}(F(x_j|x_1,...,x_{j-1}), F(x_{j+i}|x_1,...,x_{j-1}))$$

1.7.2 D Vine Copula

Global probability density function

$$f(x_1,...,x_d) = \prod_{k=1}^{d} f(x_k) \prod_{j=1}^{d-1} \prod_{i=1}^{d-j} c_{i,i+j|i+1,...,i+j-1}(F(x_i|x_{i+1},...,x_{i+j-1}), F(x_{i+j}|x_{i+1},...,x_{i+j-1}))$$

2 Experimental method

Once I had enough copulas and distributions to test, I wanted to compare them to see how they fit the data. Because we use these data with a particular purpose of creating stochastic scenarios, using the usual likelihood was not relevant. We needed a tool that was focusing on the dependence on the tails and that was able to compute take into account the problem of unreprodicibility explained below. Using ideas from [4] and by the previous intern in Davis, Ambroise Idoine, I built a method to compare the accuracy of copulas. In this section, I explain what it consists in and how I implemented it.

2.1 Solving the problem of unreproducibility

Each day j, we look at a set of data including for example a description of the state of the system early in the morning and the observations of the 90 previous days. With this set of data, we are able to choose a distribution represented by its cumulative density function F_j with a set of parameters θ_j that we hope will predict the best what will happen on day j. Some techniques are presented in [2] or [3] but we will not focus on them in this article. We now want to verify if this distribution was well chosen.

Unfortunately, we only observe what happens on the day j once. Let us note O_j the day j observation. O_j is not sufficient to check if this random variable follow the distribution F_j. Moreover, each day is different and for instance another day $i \neq j$ will give a different distribution F_i with different parameters θ_i. Thus, it is impossible to verify if each distribution is correct each day. Nevertheless, their are techniques to verify if our procedure is valid and if the estimation of distributions makes sense.

Let us define $U_j = F_j(O_j)$. Property 1 tells us that U_j has a uniform distribution. As all U_j are computed independantly, all the U_j must be independant and distributed uniformly.

We now have a set of observation $\mathbf{U} = (U_1, ..U_n)$ that should be independant and uniformly distributed on [0,1]. We can now compute the extent to which it follows a uniform distribution for example by computing the Earth Mover Distance between the empirical distribution of U and the uniform distribution or by using histograms.

2.2 Histograms inspired by rank histograms

When we have a big set of data that are realisation of random variables that should be uniformly distributed between 0 and 1. The histograms of their data should be flat between 0 and 1 and equal to 0 elsewhere. This gives a method for verifying if a dataset is well fitted by a distribution using histograms which is well described in [7]. But as [7] says it, the biggest problem with rank histograms is that they are primarly useful only in one dimension. So, we adapted the methods by making projection in order to plot our histograms. Projecting on the marginals is useless because what we care about is the dependance. Thus, we will project on the diagonals. This way we can measure the fit of the copulas in the corner that we are interested in.

2.3 Earth Mover Distance

Definition 9. *The **Earth Mover Distance** (EMD) between two histograms $P = ((x_i, p_i))_i$ and $Q = ((y_j, q_j))_j$ is :*

$$EMD(P,Q) = \frac{\min_{(f_{i,j}) \in F} \sum_{i,j} f_{i,j} d_{i,j}}{\min(\sum_i p_i, \sum_i q_i)}$$

where $F = \{(f_{i,j}) | f_{i,j} \geq 0, \sum_i f_{i,j} \leq P_i, \sum_j f_{i,j} \leq Q_j, \sum_{i,j} f_{i,j} = \min(\sum_i p_i, \sum_i q_i)\}$
and $d_{i,j}$ is the distance between x_i and y_j.

In our problem, we look at histograms of distribution with real density function. So, with probability 1 we will not have the same result twice. Thus, the weight of our histograms will just be 1 for each value : $\forall i, q_i = 1, \forall j, p_j = 1$.
Moreover, we will only consider vectors with same dimension d. We can now simplify the notation and define the EMD between two vectors :

Definition 10. *The **Earth Mover Distance** (EMD) between two vectors* $\boldsymbol{u} = (u_1, ..., u_d)$ *and* $\boldsymbol{v} = (v_1, ..., v_d)$ *of dimension d is :*

$$EMD(\boldsymbol{u}, \boldsymbol{v}) = \frac{1}{d} \min_{(f_{i,j}) \in F} \sum_{i=1}^{d} \sum_{j=1}^{d} f_{i,j} d_{i,j}$$

where $F = \{(f_{i,j}) | f_{i,j} \geq 0, \sum_{i=1}^{d} f_{i,j} \leq 1, \sum_{j=1}^{d} f_{i,j} \leq 1, \sum_{i=1}^{d} \sum_{j=1}^{d} f_{i,j} = d\}$
and $d_{i,j} = |u_i - v_j|$

Solving this linear program is possible but there is a faster way to compute the EMD thanks to the following property :

Property 2. *For any vectors* $\boldsymbol{u}$ *and* $\boldsymbol{v}$ *of dimension n :*

$$EMD(\boldsymbol{u}, \boldsymbol{v}) = \frac{1}{d} \sum_{i=1}^{d} |\tilde{u}_i - \tilde{v}_i| = \frac{1}{d} ||\tilde{\boldsymbol{u}} - \tilde{\boldsymbol{v}}||_1$$

where $\tilde{\boldsymbol{x}}$ *is the sorted vector of* $\boldsymbol{x}$ *:*
$\{x_1, ..., x_d\} = \{\tilde{x}_1, ..., \tilde{x}_d\}$ *and* $\forall i \leq j, \tilde{x}_i \leq \tilde{x}_j$

2.4 Projection on diagonal

Because we are very interested in extreme events, we will focus on tails of the multivariate distribution. To study their dependance, it is interesting to consider the corners of the space of copulas which is an hypercube.

2.4.1 Corner

Definition 11. *A **corner** of an hypercube* $[0,1]^d$ *is a point* $\boldsymbol{a} = (a_1, ..., a_d) \in \{0,1\}^d$ *:*

$$\forall i \in [\![1, d]\!], a_i = 0 \text{ or } a_i = 1$$

So there are 2^d corners in a hypercube of dimension d.

2.4.2 Diagonal

Definition 12. *A **diagonal** Δ is a segment which links to opposite corner* $\boldsymbol{a}$ *and* $\boldsymbol{b}$ *:*

$$\Delta = [\boldsymbol{a}, \boldsymbol{b}] \text{ where } \forall i \in [\![1, d]\!], a_i = 0 \iff b_i = 1$$

Alternatively :

$$\Delta = \{(1 - \lambda)\boldsymbol{a} + \lambda \boldsymbol{b}, \lambda \in [0,1]\}, \text{ where } \forall i \in [\![1, d]\!], a_i = b_i + 1 \text{ mod } 2$$

*Because one diagonal can be written [**a**,**b**] or [**b**,**a**], we will always consider* $a_1 = 0$ *so that each diagonal has a unique way notation.*

*We can also define the **direction** of a diagonal as the vector :*

$$U_\Delta = \frac{1}{\sqrt{d}} (\boldsymbol{b} - \boldsymbol{a})$$

2.4.3 Projection

Definition 13. *The **matrix of projection** on the linear space will be :*

$$M_\Delta = U_\Delta U_\Delta^\top$$

*Finally, the **projection on the diagonal** which is an affine space is the function* P_Δ *such that :*

$$P_\Delta(X) = M_\Delta(X - C) + C \text{ where } C = (\frac{1}{2}, ..., \frac{1}{2}) \quad (1)$$

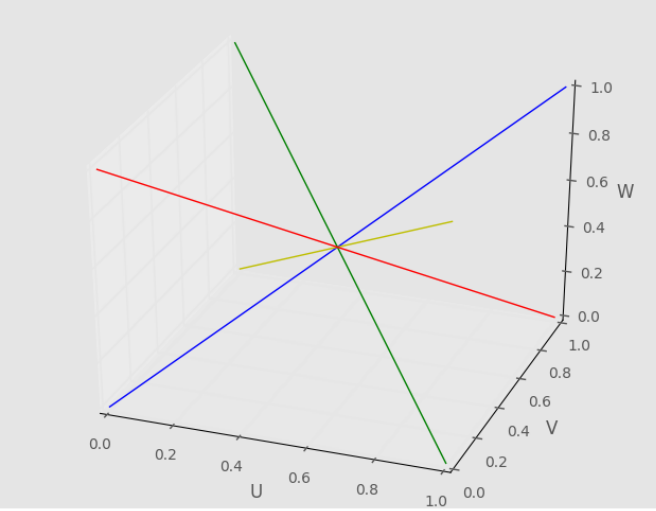

Figure 4: Examples of diagonals, *The blue one is [(0,0,0),(1,1,1)], the yellow one is [(0,1,0),(1,0,1)], the green one is [(0,1,1),(1,0,0)], and the red one is [(0,0,1),(1,1,0)]*

So there are 2^{d-1} diagonals, directions and matrix of projection in an hypercube of dimension d.

The division by $\sqrt{d}$ in the definition of direction permits to have a unit vector.

M is indeed a matrix thanks to the order of the factors (and not a scalar product as $U^\top U$).

One should not confuse the matrix of projection on the linear space and the traditional affine projection on the diagonal. That is why we need to translate everything with the center of the hypercube C.

2.4.4 Distribution on the diagonal

We now want to study the distribution of the points projected on the diagonal to compare it to a uniform distribution. Since the diagonal is a segment, each point x of the diagonal can be described by only one scalar number λ : $x = (1-\lambda)a + \lambda b$ (cf Definition of the diagonal).
λ can be understood as the normalised distance between a and x :

$$\|x - a\| = \|(1-\lambda)a + \lambda b\| = \lambda\|a - b\| = \lambda\sqrt{d}$$

Where $\|.\|$ is a norm in our space.

But λ can be easily evaluated by taking the first coordonates of x :

$$x_1 = (1-\lambda)a_1 + \lambda b_1 = (1-\lambda) * 0 + \lambda * 1 = \lambda$$

This equality is possible thanks to our useful convention $(a_1, b_1) = (0, 1)$.

We now have a unique number that should be uniformly distributed on [0,1].

2.5 Our algorithm

For each day j do :

- Using all day $i \leq j-1$, fit a parametric distribution model with copula C_j
- Generate n realizations U of the random variable with the copula dependance and uniform marginals :

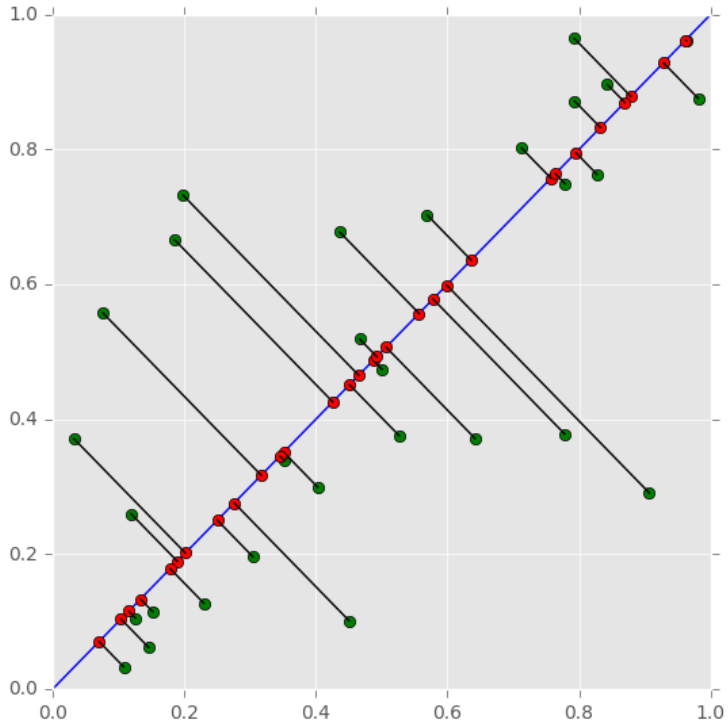

Figure 5: Projection of 30 points on the [(0,0),(1,1)] diagonal *The diagonal is in blue, the initial points are green and their projections are red.*

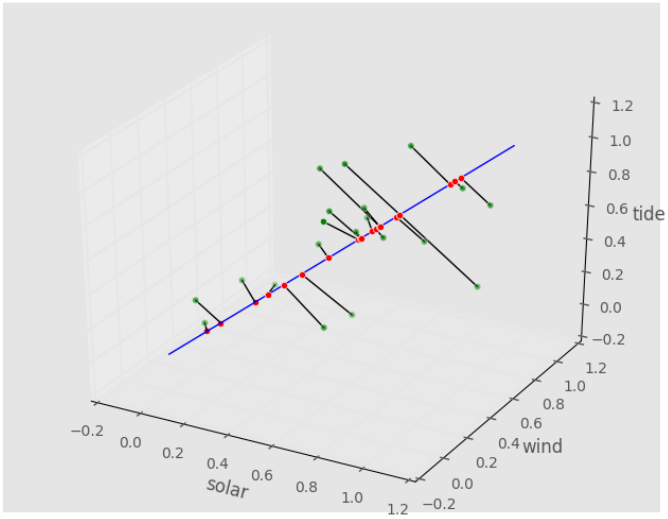

Figure 6: Projection of 20 points on the [(0,0,0),(1,1,1)] diagonal *The diagonal is in blue, the initial points are green and their projections are red.*

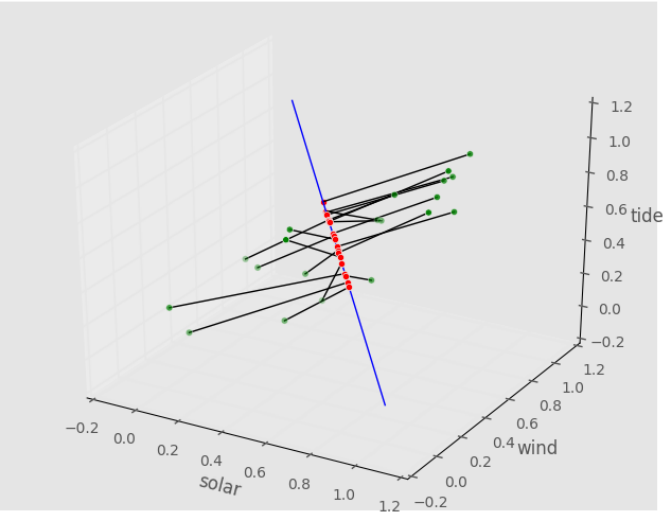

Figure 7: Projection of 20 points on the [(1,0,0),(0,1,1)] diagonal *The diagonal is in blue, the initial points are green and their projections are red.*

Generate $\mathbf{U} = (U_1, ..., U_n)$ with each $U_i = (U_{i,1}, ..., U_{i,d}) \in [0,1]^d$

where $\forall i \in [\![1, n]\!]$,
$\mathbb{P}(U_{i,1} \leq u_1, ..., U_{i,d} \leq u_d) = C(u_1, ..., u_d)$ and
$(\forall j \in [\![1, d]\!], U_{i,j}$ is uniformly distributed on [0,1].)
For example, n=10000

- For each diagonal Δ :
- Project all the U_i on the diagonal
  Define $\mathbf{V}_\Delta = (V_{\Delta,1}, ..., V_{\Delta,n}) = (P_\Delta(U_1), ..., P_\Delta(U_n))$
- Define the empirical distribution on this diagonal :

  $F_\Delta(X) = \frac{1}{n} \sum_{k=1}^{n} \mathbb{1}_{V_{\Delta,k} \leq X}$

  where $a \leq b \iff \forall i \in [\![1, d]\!], a_i \leq b_i$
- Observe with the data what happened on day j
  Call this observation $O_j = (O_{j,1}, ..., O_{j,d})$
- Pass it in the copula space
  Define $Q_j = (F_1(O_{j,1}), ..., F_d(O_{j,d}))$
  where the F_i are the cumulative density functions of the marginals estimated with another method.
- Project Q_j on the diagonal
  Define $R_{\Delta,j} = P_\Delta(Q_j)$
- Define $S_{\Delta,j} = F_\Delta(R_{\Delta,j})$

- Either compute the distance between the empirical distribution of the $S_\Delta = (S_{\Delta,i})_{i \in days}$ and the uniform distribution on [0,1].
- Or make a histogram with the $S_\Delta = (S_{\Delta,i})_{i \in days}$ $F_\Delta^{-1}(P_j)$

2.6 Test code

In this section, I will explain how we excecuted this algorithm with different parameters. They are arguments of many test functions I wrote and are just strings defining options :

`source` : the type of power source we want it can be 'solar' or 'wind'

`datatype` : if the data are power ('actuals'), errors ('errors), a normal distributed sample ('normal-sample') or a uniformly distributed sample ('uniform-sample'), the two last ones are options to make verifications.

`segment_marginals` : to segment the data to fit the marginals, you can either take only the date at the hour of your dps ('hour') or fit this marginal with the data of the whole day ('anytime'). Note : this is not the way the data are segmented to fit the copula.

`kind` : which projection you want it can be on a diagonal (`"diagonal"`), a marginal (`"marginal"`) or can even compose with the kendall function (`"kendal"`).

`index` : the index of the diagonal or the marginal you want to project on. Diagonals are indexed in the order of diag. list of diags. Marginals are index as the coordonates. index does not matter for kendall function.

`method` : way you choose the data to fit the distributions, you can either fit copulas and marginals with the datas of the whole year and check the observation then ('wholeyear'), or you can decide to just use the days before the one you are computed ('daytoday')

3 Results and analysis

I executed the experiment described previously on different sets of datas from CAISO and then from BPA. Because CAISO datasets for the years 2014 and 2015 were not big enough to be representative, I then did all my experiments with the datas from BPA from May 2012 to April 2017. All the figures presented below are obtained from the BPA datas thanks to the algorithm described above. Because I did not want to make the tests more complex, I only worked on wind power, it will nearly be the same with solar power except that I could not consider the hours during the night.

3.1 Main experiment with actuals

The simplest experiment we can imagine is to test if our different copulas fit well the correlation between wind power production at two adjacent hours. So I computed the algorithm of section 2 with the actuals power at noon and at 1 PM. The obtained histograms are presented below.

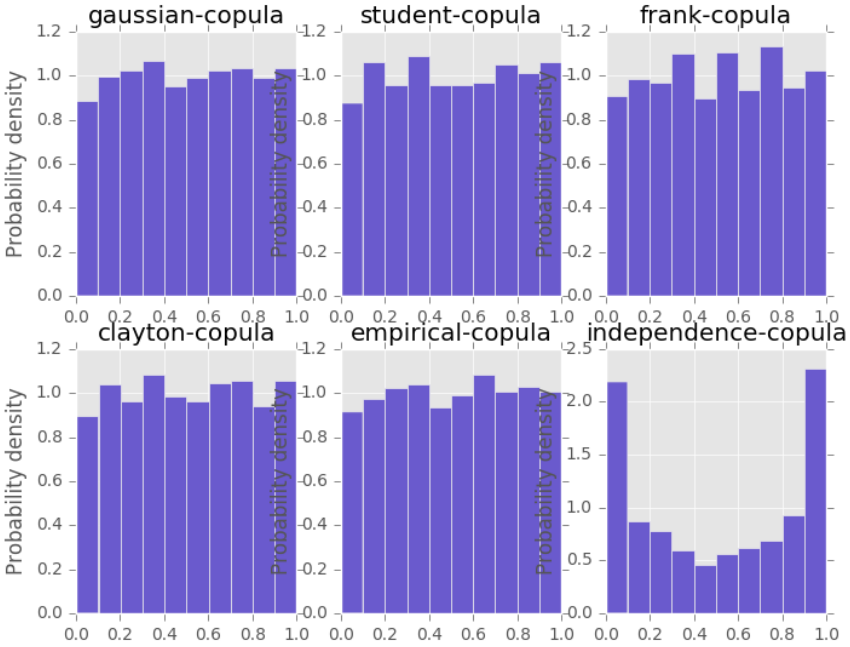

Figure 8: Results of the test on the dependance between the actuals at 12 and 13 when projected on diagonal [(0,0),(1,1)] and without segmentation

One can see that, except for the independance copula, these histograms look very flat. (The gumbel copula is not represented on this graph for presentation issues but has also been calculated as one can see in the EMD results below). These results show that our copulas fit well highly correlated datas on the direct diagonal.

Diagonal	Copula	(0, 1)	(0, 0.1)	(0.9, 1)
[[0, 0], [1, 1]]	gaussian-copula	0.0061	0.0075	0.0065
	student-copula	0.0066	0.0073	0.0045
	frank-copula	0.0072	0.0079	0.0073
	clayton-copula	0.0064	0.0087	0.0128
	gumbel-copula	0.0067	0.0065	0.0058
	empirical-copula	0.0065	0.0062	0.0069
	independence-copula	0.0830	0.0412	0.0449

Figure 9: The EMD results for wind actuals at 12 and 13 with univariate empirical marginals on diagonal [[0, 0], [1, 1]]

When we look at the earth mover distances between the S vectors we computed and a uniformly and regularly distributed vector on [0,1] with the same length, no copula is way better than the others.

Indeed, the differences between these copulas are so little that we cannot tell if those differences are simply due to the randomness of the experiment. The only exception is the independence copula which fits worse the data than all other copulas. This looks reasonable and reassuring considering this data are highly correlated. It means that using copulas is not useless.

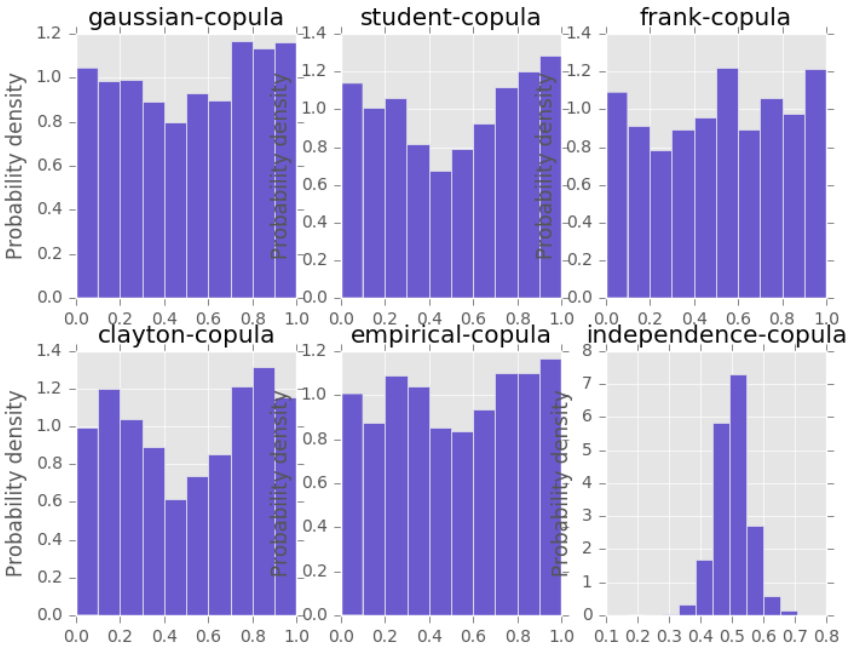

Figure 10: Results of the test on the dependance between the actuals at 12 and 13 when projected on diagonal [(0,1),(1,0)] and without segmented the datas

When we look at the projection on the other diagonal, we see that our copulas are less precised to fit the anticorrelation. Nevertheless, they are not U-shaped or Λ-shaped, so we can assume that these imprecison is just noise due to the lack of data in the anticorrelation areas next to the corners. The method might converge slowlier on this diagonal than on the other one, so this diagonal might need more data.. When we look at the EMD results, we have the same conclusion as the direct diagonal : all copulas fits pretty well and equivalently except the independence copula which is very bad.

Diagonal	Copula	(0, 1)	(0, 0.1)	(0.9, 1)
[[0, 1], [1, 0]]	gaussian-copula	0.0187	0.0097	0.0113
	student-copula	0.0275	0.0087	0.0113
	frank-copula	0.0184	0.0122	0.0130
	clayton-copula	0.0265	0.0019	0.0032
	gumbel-copula	0.0206	0.0024	0.0043
	empirical-copula	0.0144	0.0065	0.0054
	independence-copula	0.2090	0.3494	0.3527

Figure 11: The EMD results for wind actuals at 12 and 13 with univariate empirical marginals

3.2 Main experiment with errors

Even if it is easier to understand power production actuals, we need to do the experiments with the errors (which is the difference forecast minus actuals) because GOSM always works with errors and its distributions are calculated using the errors and not the actuals. In order to have more precised distributions, GOSM also segments its datas which means we choose cleverly which data we will put in the arguement input_data of our distribution class. In the case below, we segment by forecasts and with a window of 40 %.

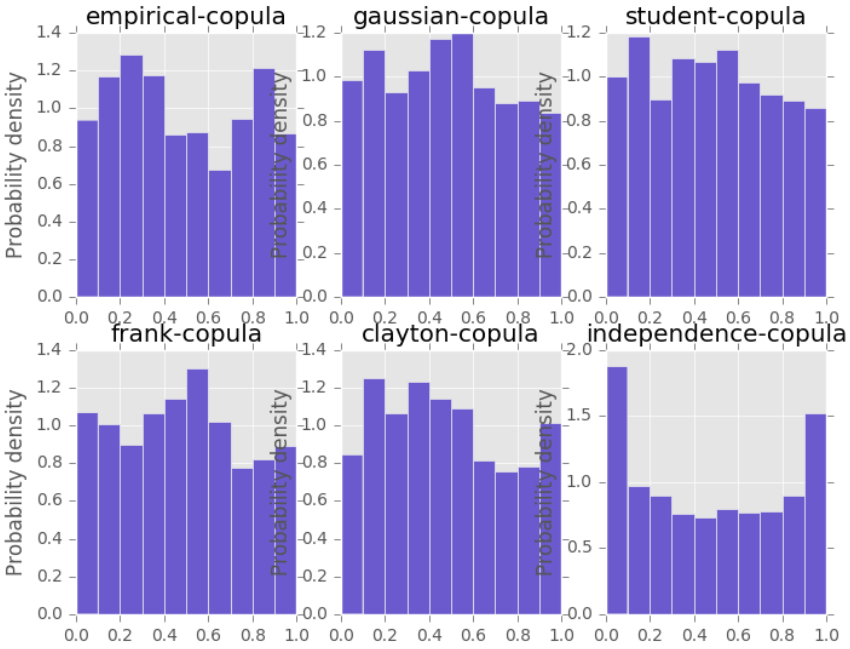

Figure 12: Results of the test on the dependance between the errors at 12 and 13 when projected on diagonal [(0,0),(1,1)] and by segmented the datas with the forecast

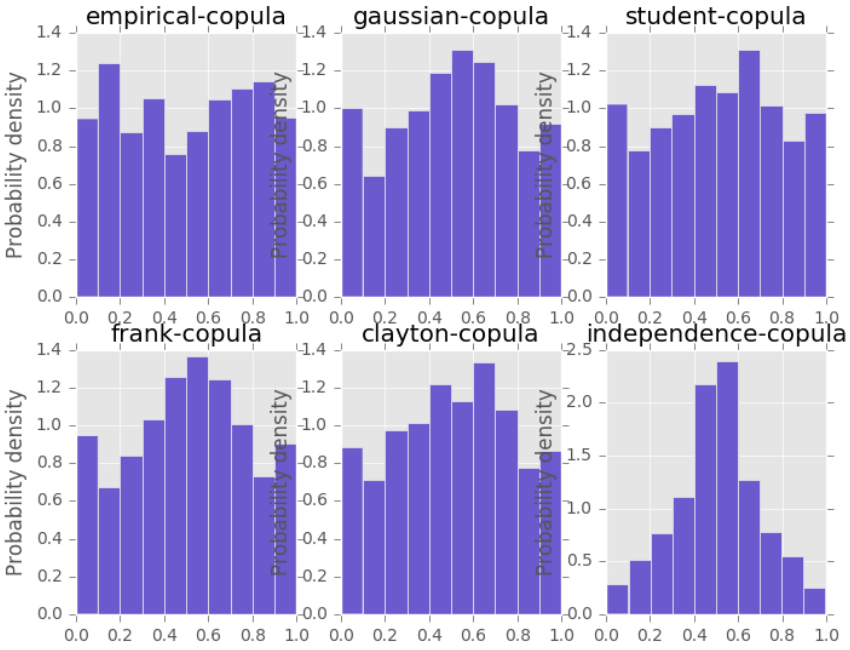

Figure 13: Results of the test on the dependance between the errors at 12 and 13 when projected on diagonal [(0,1),(1,0)] and by segmented the datas with the forecast

The obtained results are here similar to the results for the actuals. What we learn is that copulas are not really different for fitting the the data except for the independence copula that fits it badly as we expect.

Diagonal	Copula	(0, 1)	(0, 0.1)	(0.9, 1)
[[0, 0], [1, 1]]	empirical-copula	0.0168	0.0144	0.0062
	gaussian-copula	0.0203	0.0189	0.0079
	student-copula	0.0178	0.0165	0.0058
	frank-copula	0.0231	0.0265	0.0141
	clayton-copula	0.0242	0.0081	0.0236
	gumbel-copula	0.0216	0.0258	0.0036
	independence-copula	0.0496	0.0414	0.0364
[[0, 1], [1, 0]]	empirical-copula	0.0114	0.0084	0.0043
	gaussian-copula	0.0418	0.0095	0.0130
	student-copula	0.0311	0.0064	0.0091
	frank-copula	0.0463	0.0074	0.0102
	clayton-copula	0.0397	0.0088	0.0084
	gumbel-copula	0.0400	0.0074	0.0098
	independence-copula	0.1053	0.0764	0.0769

Figure 14: The EMD results for wind errors at 12 and 13 with univariate empirical marginals

3.3 Verification and marginals

In order to verify if our method was working, I coded different other tools linked to the tests. I first needed to clean the datas so that the software won't fail. I suppressed all the datas that had no sense such as strings instead of float and converted it to `NaN`. Then each time I would call a a new distribution object, I would drop the `NaN` in the columns i am using. I also delt with the problem of datetime objects. Because I was studying the dependence between data at different hours, I wanted that the dates keep being the indexes of my dataframe and the hours become the columns while keeping the different datatypes.

Then, I needed to verify if the experiment was working with data for which we know the properties. So, I added to my dataframe, random samples that I generated with python `numpy.random` functions. All these work on data made by in the function `creates_data` of copula.py.

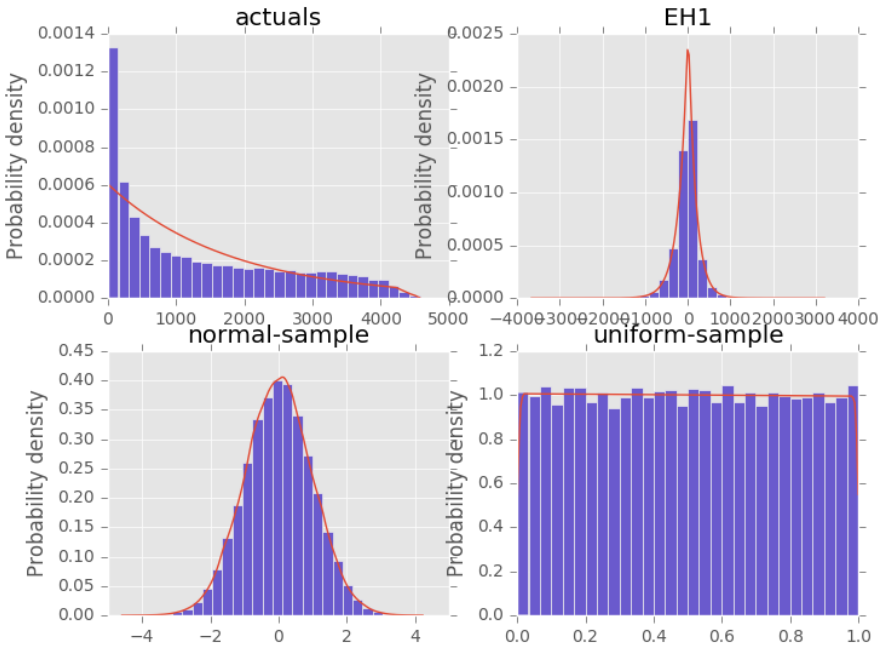

Figure 15: Histograms of our data and their fitted univariate epispline pdf

I coded functions that draw plots to understand our data more. `csv_to_histogram` permits to draw an histogram from a dataframe stored in a csv. One can then easily see the distribution of actuals, errors to one hour forecast (EH1) or to verify if our random samples have the good properties.

I also had to check if the marginals fitted the data well, otherwise the whole experiment on copulas

was useless. That is why I added the possibility to project on a marginal instead of a diagonal in the big code. In a simplier way, I added the possibility in `csv_to_histogram` to draw the pdf of a fitted univariate distribution of any class of the user's choice. I found that in some cases, the univariate epispline distribution can fit badly the datas. For example, it was not able to model the high probability to have a production of 0 MW. This caused me a lot of trouble for my experiments because univariate epispline were the default distribution in GOSM. But we finally decided to work with the univariate empirical distribution in our tests to temporarly avoid this problem and focus on copulas. All the experiments presented above are then computed with univariate empirical marginals.

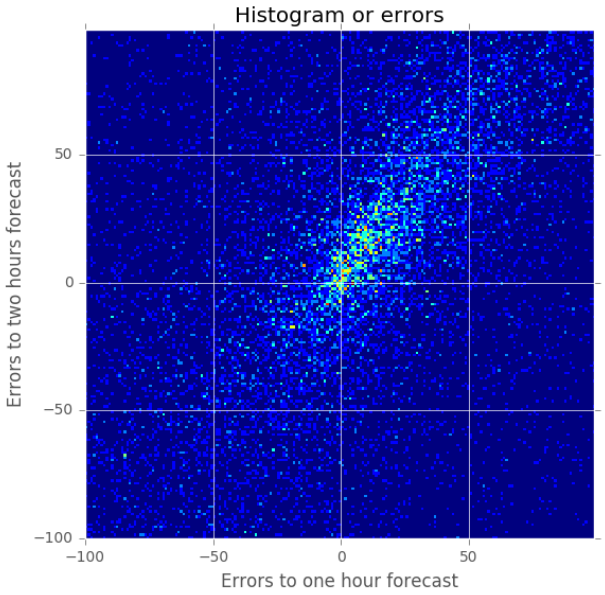

Figure 16: Histograms of errors

As well as we verified our data and the fitted distribution linked with them in one dimension, I coded a function called `print_contours` that draw a 2d histograms of our data thanks to a color code and the contours of the pdf of the fitted multivariate distribution with copula and marginals of the user choice. It also permits to compare visually if the copula seems to fit well the data. In our case, the contours plots are coherent with the datas histograms for all copula except for the independence one.

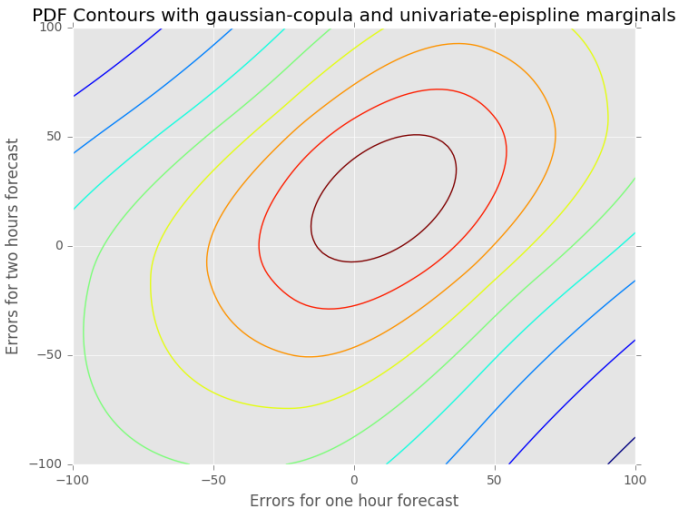

Figure 17: Contours of the error-fitted multivariate distribution, with gaussian copula and univariate epispline marginal, pdf

Conclusion

I coded a complex tool which is a small part of a larger project. With my experiments, I was able to confirm that copulas are really useful to model the dependence between our different data. We now know that all copulas fit the data that we tested with roughly the same accuracy except the independence one. However, my work consisted in the first experiments focused on specific datas. So it needs to be continued to have a larger vision of the problem and more precise conclusions with higher degrees of likelihood. But the main code is now written and permits much more experiences than the simplest ones. I give ideas of directions for future work in appendix D. All this code will be very useful in GOSM, every user can now choose which copula he wants and creates more precise scenario and especially more precise probability. With these more relevant inputs, a GOSM user can now run the stochastic optimization problem and test if adding copulas give better results.

Concerning this internship, I found really interesting to build a coherent and complex structure such as this distribution and copula code. I realized the importance of verifying of the procedure which can become completely wrong even if we have made a small mistake. Once the code is running, it can look like a black box which gives results we have to believe. We should always avoid that and prefer to have a precised vision of our data and our algorithmn so that we really understand what happens so we would be able to interpret correctly the results. That is why I coded many unittests which are really good verifications on the one hand and which make us understand more precisely our problem on the other hand. I also learned that dealing with real data is way more complex than simulated ones. You have many small bugs you need to fix and you do not know many things about this data. This is why I added the possibility to check and compare with simulated samples which are very easy to deal with.

I enjoyed doing some research. Dealing with interesting and clever mathematical problems always insterested me. I also liked the idea that this project is useful in the field of energy. Since I am interesting in both optimization and energy, I would like to continue working on such projects. They are not closed fields since my internship was eventually more about statistics and probability theory than optimization even if it was a part of a bigger stochastic optimization problem. As I did in this internship, I would like to still be opened and also learn from other subjects. Next year, I will study a master in Université Pierre et Marie Curie in Paris to continue learning about optimization while integrating the Corps des Ingénieurs des Ponts Eaux et Forêts to work on energy. My goal is to make a thesis in optimization applied to the energy and the smart grids field. This internship was a first experience for me that taught me a lot.

References

[1] Alexander J. McNeil, Johanna Neslehova *Multivariate Archimedean Copulas, d-monotone Functions and l1-norm Symmetric Distributions*,Ann. Stat., 37:3059-3097, 2009.

[2] Kjersti Aas, Claudia Czado, Arnoldo Frigessi, Henrik Bakken, *Pair-copula construction of multiple dependence*, 2007.

[3] Kjersti Aas, *Modeling the dependance structure of financial assets : A survey of four copulas* 2004.

[4] Johanna F. Ziegel, Tilmann Gneiting, *Copula Calibration*, Electronic Journal of Statistics, 2014.

[5] Johan Segers, *Copulas: An Introduction Part II: Models*, Columbia University, New York City, 9_11 Oct 2013.

[6] Sabrina Nitsche, Scott Winner, C/'esar A. Silva-Monroy, Andrea Staid, Jean-Paul Watson, David L. Woodruff *Improving Wind Power Prediction Intervals Using Vendor-Supplied Probabilistic Forecast Information.*

[7] Thomas M. Hamill, *Interpretation of Rank Histograms for Verifying Ensemble Forecasts*, 2000.

[8] Stéphane Gaubert, Frédéric Bonnans, *Recherche opérationnelle : aspects mathématiques et applications*, Editions de l'École polytechnique, 2016

[9] Johannes O.Royset, Roger J-B Wets, *Nonparametric Density Estimation via Exponential Epi-Splines: Fusion of Soft and Hard Information*, 2013.

Appendix

A List of files and classes

base_distribution.py
BaseDistribution
MultiDistr

distributions.py
UnivariateBasicEmpiricalDistribution
UnivariateEmpiricalDistribution
UnivariateEpiSplineDistribution
UnivariateUniformDistribution
UnivariateNormalDistribution
UnivariateStudentDistribution
MultiNormalDistribution
MultiStudentDistribution

copula.py
CopulaBase
GaussianCopula
StudentCopula
FrankCopula
ClaytonCopula
GumbelCopula
WeigthedCombinedCopula
IndependenceCopula
EmpiricalCopula

vine.py
CVineCopula
DVineCopula

distribution_factory.py

tester.py
MultiNormalDistributionTester
UnivariateNormalDistributionTester
MultiStudentDistributionTester
UnivariateStudentDistributionTester
CopulaTester
VineCopulaTester

B Distribution formulas

`UnivariateBasicEmpiricalDistribution` is the simplest distribution you can build without making an hypothesis on the model. It assumes that all the results in the `input_data` are the only possible ones and that they are equiprobable. This gives us a method to generate a sample, we simply pick randomly one element of `input_data`. If we note $Y = (Y_1, ..., Y_n) =$ `input_data`, the functions would be :

$$F(x) = \frac{1}{n}\sum_{k=1}^{n} \mathbb{1}_{Y_k \leq x}$$

$$\mathbb{P}(X = x) = \frac{1}{n}\sum_{k=1}^{n} \mathbb{1}_{Y_k = x}$$

One can notice that F is not continuous, that implies that f is not well defined.

`UnivariateEmpiricalDistribution` works almost like `UnivariateBasicEmpirical` but avoids the problem of continuity by cleverly interpolating lines between the points we have. When the set of data is big enough it gives the same result than `UnivariateBasicEmpricalDistribution`.

`UnivariateEpiSplineDistribution` is a more clever object obtained thanks to the resolution of an optimization problem where the cdf as a particular shape (for more information see [9]) :

$$F(x) = e^{-g(x)}, \text{ where } g \text{ is polynomial by pieces}$$

To use this class, it needs a `gosm_options` object to be defined "globally". For example, one can choose the number N which indicates how much subsegments we want to divide our initial segment to create each piece of the polynomial function. Plus, this class needs the installation of the language pyomo and of the solver ipopt. To learn more about these installation, check `GOSM` user manual. This class is unusual because it was extracted from an earlier version of software called `Prescient`.

`UnivariateUniformDistribution` contains only two parameters called `a` and `b` who represent respectively the minimum and the maximum of the distribution. When an object of this class is called with `input_data`, the `__init__` method, assign the minimum of `input_data` to a and its maximum to b.

$$a = \texttt{a} = \texttt{min(input\_data)}$$

$$b = \texttt{b} = \texttt{max(input\_data)}$$

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{if } x \in [a, b] \\ 0 & \text{else} \end{cases}$$

$$F(x) = \begin{cases} 0 & \text{if } x \leq a \\ 1 & \text{if } x \geq b \\ \frac{x-a}{b-a} & \text{else} \end{cases}$$

`UnivariateNormalDistribution` has two parameters `mean` and `var` which are stored as attributes. We present below the main formulas that define this class. Each time we represent parameters with their names in the python code and with its usual mathematical notation so that the formulas are not too hard to read.

$$\mu = \texttt{mean} = \texttt{mean(input\_data)}$$

$$\sigma^2 = \texttt{var} = \texttt{var(input\_data)}$$

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{\sigma^2}}$$

$$F(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y-\mu)^2}{\sigma^2}} dy$$

`UnivariateStudentDistribution`

$$\mu = \texttt{mean} = \texttt{mean(input\_data)}$$

$$\nu = \texttt{df} = \texttt{2var(input\_data)/(var(input\_data)-1)}$$

$$f(x) = \frac{1}{\sqrt{\nu\pi}} \frac{\Gamma(\frac{\nu+1}{2})}{\Gamma(\frac{\nu}{2})} \left(1 + \frac{(x-\mu)^2}{\nu}\right)^{-\frac{\nu+1}{2}}$$

$$F(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{\nu\pi}} \frac{\Gamma(\frac{\nu+1}{2})}{\Gamma(\frac{\nu}{2})} \left(1 + \frac{(y-\mu)^2}{\nu}\right)^{-\frac{\nu+1}{2}} dy$$

`MultiNormalDistribution`

$$\mu = \texttt{mean} = \texttt{mean(input\_data)}$$

$$K = \texttt{cov} = \texttt{cov(input\_data)}$$

$$f(x) = \frac{1}{(2\pi)^{\frac{d}{2}}\sqrt{det(K)}} e^{-\frac{(x-\mu)^\top K^{-1}(x-\mu)x}{2}}$$

$$F(x) = \int_{-\infty}^{x} \frac{1}{(2\pi)^{\frac{d}{2}}\sqrt{det(K)}} e^{-\frac{(y-\mu)^\top K^{-1}(y-\mu)}{2}} dy$$

`MultiStudentDistribution`

$$\mu = \texttt{mean} = \texttt{mean(input\_data)}$$

$$\nu = \texttt{df}$$

$$K = \texttt{cov} = \texttt{cov(input\_data)}$$

$$f(x) = \frac{\Gamma(\frac{\nu+d}{2})}{\Gamma(\frac{\nu}{2})(\nu\pi)^{\frac{d}{2}}\sqrt{det(K)}} \left(1 + \frac{1}{\nu}(x-\mu)^\top K^{-1}(x-\mu)\right)^{-\frac{\nu+d}{2}}$$

$$F(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{\nu\pi}} \frac{\Gamma(\frac{\nu+1}{2})}{\Gamma(\frac{\nu}{2})} \left(1 + \frac{(y-\mu)^2}{\nu}\right)^{-\frac{\nu+1}{2}} dy$$

C Copula formulas

C.1 Elliptical copulas

C.1.1 Gaussian Copula

$\mathcal{N}(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{t^2}{2}} dt$

C function in dimension d $C_K(u_1, ...u_d) = \int_{-\infty}^{\mathcal{N}^{-1}(u_1)} ... \int_{-\infty}^{\mathcal{N}^{-1}(u_d)} \frac{1}{(2\pi)^{\frac{d}{2}}\sqrt{det(K)}} e^{-\frac{x^\top K^{-1} x}{2}} dx$

With a change of variable, we have also

$$C_K(u_1, ...u_d) = \int_0^{u_1} ... \int_0^{u_d} \frac{1}{\sqrt{det(K)}} e^{-\frac{\mathcal{N}^{-1}(x)^\top (K^{-1} - I_d)\mathcal{N}^{-1}(x)}{2}} dx$$

where $\mathcal{N}^{-1}(x) = \begin{pmatrix} \mathcal{N}^{-1}(x_1) \\ \vdots \\ \mathcal{N}^{-1}(x_d) \end{pmatrix}$

C function in dimension 2 In dimension 2, all matrix of correlation can be written $K = \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}$. So we will only consider one parameter ρ.

$$C_\rho(u, v) = \int_{-\infty}^{\mathcal{N}^{-1}(u)} \int_{-\infty}^{\mathcal{N}^{-1}(v)} \frac{1}{2\pi\sqrt{1-\rho^2}} e^{-\frac{x^2 - 2\rho xy + y^2}{2(1-\rho^2)}} dxdy$$

With the same change of variable, we have also

$$C_\rho(u, v) = \int_0^u \int_0^v \frac{1}{\sqrt{1-\rho^2}} e^{-\frac{\rho^2\mathcal{N}^{-1}(x)^2 - 2\rho\mathcal{N}^{-1}(x)\mathcal{N}^{-1}(y) + \rho^2\mathcal{N}^{-1}(y)^2}{2(1-\rho^2)}} dxdy$$

c function in dimension n

$$c_K(u_1, ..., u_d) = \frac{1}{\sqrt{det(K)}} e^{-\frac{\mathcal{N}^{-1}(u)^\top (K^{-1} - I_d)\mathcal{N}^{-1}(u)}{2}}$$

where $\mathcal{N}^{-1}(u) = \begin{pmatrix} \mathcal{N}^{-1}(u_1) \\ \vdots \\ \mathcal{N}^{-1}(u_d) \end{pmatrix}$

C.1.2 Student Copula

We will note $t_\nu(x) = \frac{\Gamma(\frac{\nu+1}{2})}{\sqrt{\nu\pi}\Gamma(\frac{\nu}{2})} \int_{-\infty}^{x} \left(1 + \frac{t^2}{\nu}\right)^{-\frac{\nu+1}{2}} dt$, the cumulative density function of the student distribution with degree of freedom ν and t_ν^{-1} its inverse.
We could write both with the regularized incomplete Beta function
$I_{a,b}(x) = \frac{1}{B(a,b)} \int_0^x t^{a-1}(1-t)^{b-1} dt$
$= \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} \int_0^x t^{a-1}(1-t)^{b-1} dt$

and its inverse $I_{a,b}^{-1}$:

$$t_\nu(x) = \begin{cases} I_{\frac{\nu}{2},\frac{1}{2}}(\frac{\nu}{x^2+\nu}) & \text{if } x \leq 0 \\ 1 - I_{\frac{\nu}{2},\frac{1}{2}}(\frac{\nu}{x^2+\nu}) & \text{if } x \geq 0 \end{cases}$$

and

$$t_\nu^{-1}(x) = \begin{cases} -\sqrt{\nu(\frac{1}{I^{-1}_{\frac{\nu}{2},\frac{1}{2}}(2y)} - 1)} & \text{if } x \leq \frac{1}{2} \\ \sqrt{\nu(\frac{1}{I^{-1}_{\frac{\nu}{2},\frac{1}{2}}(2(1-y))} - 1)} & \text{if } x \geq \frac{1}{2} \end{cases}$$

Generate X To generates our U vector in the $[0,1]^d$ space, we need to first generates a X vector in the $\mathbb{R}^d$ space following a Multivariate Student Distribution. Thus, we use the equality in law :

$$\sqrt{\frac{\nu}{S}} Z \sim t_\nu(0, K)$$

where $s \sim \chi^2_\nu$ and $Z \sim \mathcal{N}(0, K)$

C function in dimension d

$$C_K(u_1, ...u_d) = \int_{-\infty}^{t_\nu^{-1}(u_1)} ... \int_{-\infty}^{t_\nu^{-1}(u_d)} \frac{\Gamma(\frac{\nu+d}{2})}{\Gamma(\frac{\nu}{2})(\nu\pi)^{\frac{d}{2}}\sqrt{det(K)}} \left(1 + \frac{1}{\nu} x^\top K^{-1} x\right)^{-\frac{\nu+d}{2}} dx$$

With a change of variable, we have also

$$C_K(u_1, ...u_d) = \int_0^{u_1} ... \int_0^{u_d} \frac{\Gamma(\frac{\nu+d}{2})\Gamma(\frac{\nu}{2})^{d-1}}{\sqrt{det(K)}\Gamma(\frac{\nu+1}{2})^d \Pi_{j=1}^d 1 + \frac{t_\nu^{-1}(x_j)^2}{2})^{-\frac{\nu+1}{2}}} \left(1 + \frac{1}{\nu} t_\nu^{-1}(x)^\top K^{-1} t_\nu^{-1}(x)\right)^{-\frac{\nu+d}{2}} dx$$

where $t_\nu^{-1}(x) = \begin{pmatrix} t_\nu^{-1}(x_1) \\ \vdots \\ t_\nu^{-1}(x_d) \end{pmatrix}$

C function in dimension 2 In dimension 2, all matrix of correlation can be written $K = \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}$. So we will only consider one parameter ρ.

$$C_{\rho,\nu}(u, v) = \int_{-\infty}^{t_\nu^{-1}(u)} \int_{-\infty}^{t_\nu^{-1}(v)} \frac{1}{2\pi\sqrt{1-\rho^2}} \left(1 + \frac{x^2 - \rho xy + y^2}{\nu(1-\rho^2)}\right)^{-\frac{\nu+2}{2}} dxdy$$

Partial derivative of C If we derivate the formula above, we obtain (cf Pair copula constructions of muliple dependance) :

$h(u,v,\rho,\nu) = \frac{\partial C_{\rho,\nu}}{\partial v}(u,v) = t_\nu\left(\frac{t_\nu^{-1}(u)-\rho t_\nu^{-1}(v)}{\sqrt{\frac{(\nu+t_\nu^{-1}(v)^2)(1-\rho^2)}{\nu+1}}}\right)$

If we inverse this function, we obtain

$h^{-1}(u,v,\rho,\nu) = t_\nu(\sqrt{\frac{(\nu+t_\nu^{-1}(v)^2)(1-\rho)}{\nu+1}}t_\nu^{-1}(u) + \rho t_\nu^{-1}(v))$

c function in dimension d

$$c_{K,\nu}(u_1,...,u_d) = \frac{\Gamma(\frac{\nu+d}{2})\Gamma(\frac{\nu}{2})^{d-1}}{\sqrt{det(K)}\Gamma(\frac{\nu+1}{2})^d\Pi_{j=1}^d(1+\frac{t_\nu^{-1}(u_j)^2}{2})^{-\frac{\nu+1}{2}}}\left(1+\frac{1}{\nu}t_\nu^{-1}(u)^\top K^{-1}t_\nu^{-1}(u)\right)^{-\frac{\nu+d}{2}}$$

where $t_\nu^{-1}(u) = \begin{pmatrix} t_\nu^{-1}(u_1) \\ \vdots \\ t_\nu^{-1}(u_d)\end{pmatrix}$

C.2 Archimedean Copulas

Archimedean copulas take the form :

$C(u_1,...,u_d) = \Phi(\Phi^{-1}(u_1) + ... + \Phi^{-1}(u_d))$

where Φ is the Laplace transform of a positive nonzero random variable Y : $\Phi(u) = \mathbb{E}(e^{-uY})$

Φ is called the generator function.
It generally depends on a parameter called θ.
In some papers, the definition of the generator function and its inverse can be inverted. We will use the definition above which gives us simpler formulas.

All of the next formulas can be expressed with the generator function.
For example, if we refer to [1], we can write the Kendall distribution function thanks to the generator function :

$$\mathcal{K}_C(x) = \begin{cases} \frac{(-1)^{d-1}(\Phi^{-1}(0))^{d-1}}{(d-1)!}\Phi_-^{(d-1)}(\Phi^{-1}(0)) & \text{if } x = 0 \\ \sum_{k=0}^{d-2}\frac{(-1)^k(\Phi^{-1}(x))^k\Phi^{(k)}(\Phi^{-1}(x))}{k!} + \frac{(-1)^{d-1}(\Phi^{-1}(x))^{d-1}\Phi_-^{(d-1)}(\Phi^{-1}(x))}{(d-1)!} & \text{if } x \in]0,1] \end{cases}$$

C.2.1 Frank Copula

Generator, generator inverse and derivatives $\theta \in \mathbb{R}\backslash\{0\}$

$\Phi_\theta(t) = -\frac{1}{\theta}log(1+e^{-t}(e^{-\theta}-1))$

$\Phi_\theta^{-1}(t) = -log(\frac{e^{-\theta t}-1}{e^{-\theta}-1})$

$\Phi'_\theta(t) = \frac{e^{-t}}{\theta(\frac{1}{e^{-\theta}-1}+e^{-t})}$

$\Phi''_\theta(t) = \frac{-e^{-t}}{\theta(e^{-\theta}-1)(\frac{1}{e^{-\theta}-1}+e^{-t})^2}$

C function in dimension 2 $C_\theta(u,v) = -\frac{1}{\theta}log(1+\frac{(e^{\theta u}-1)(e^{\theta v}-1)}{e^{-\theta}-1})$

Partial derivative of C and inverse $h(u,v,\theta) = \frac{\partial C_\theta}{\partial v}(u,v) = \frac{e^{\theta u}-1}{e^{\theta u}+e^{\theta v}-e^{\theta(u+v-1)}-1}$

We first solve $\frac{aX+b}{cX+d} = y$, where $a = 1, b = -1, c = 1-e^{\theta(v-1)}, d = e^{\theta v}-1$

We have then $X = \frac{dy-b}{a-cy}$

We solve $e^{\theta u} = X : u = \frac{1}{\theta} log(X)$

So, we now have
$h^{-1}(u, v, \theta) = \frac{1}{\theta} log(\frac{(e^{\theta v}-1)u+1}{1-(1-e^{\theta(v-1)})u})$

c function in dimension 2 $c_\theta(u, v) = \frac{\theta e^{\theta(u+v)}(1-e^{-\theta})}{(e^{\theta u}+e^{\theta v}-e^{\theta(u+v-1)}-1)^2}$

C.2.2 Gumbel Copula

Generator and generator inverse $\theta \in [1, \infty[$

$\Phi_\theta(t) = e^{-t^{\frac{1}{\theta}}}$
$\Phi_\theta^{-1}(t) = (-log(t))^\theta$
$\Phi'_\theta(t) = -\frac{1}{\theta} t^{\frac{1-\theta}{\theta}} e^{-t^{\frac{1}{\theta}}}$
$\Phi''_\theta(t) = ((-\frac{1}{\theta} t^{\frac{1-\theta}{\theta}})^2 - \frac{1-\theta}{\theta^2} t^{\frac{1-2\theta}{\theta}}) e^{-t^{\frac{1}{\theta}}}$

C function in dimension 2 $C_\theta(u, v) = e^{-((-log(u))^\theta+(-log(v))^\theta)^{\frac{1}{\theta}}}$

Partial derivative of C and inverse $h(u, v, \theta) = \frac{\partial C_\theta}{\partial v}(u, v) = \frac{1}{v}(-log(v))^{\theta-1}((-log(u))^\theta+(-log(v))^\theta)^{\frac{1}{\theta}-1} e^{-((-log(u)^\theta+}$

We have to introduce the lambert W function which is the inverse of $x \to xe^x$:
$W(x)e^{W(x)} = x$

For $x \geq 0, ye^y = x$ has only one solution y.
So, there is no ambiguity to define $W(x)$ if $x \geq 0$.

We first solve $\lambda X^\alpha e^{-X} = y$ where $\lambda = \frac{(-log(v))^{\theta-1}}{v}, \alpha = 1 - \theta$

We obtain $X = \alpha W(\frac{1}{\alpha}(\frac{y}{\lambda})^{\frac{-1}{\theta-1}})$

Then, we solve $X = ((-log(u))^\theta + (-log(v))^\theta)^{\frac{1}{\theta}}$

which gives us $u = e^{-(X^\theta-(-log(v))^\theta)^{\frac{1}{\theta}}}$

Finally, we have $h^{-1}(u, v, \theta) = e^{-(((\theta-1)W(\frac{-log(v)}{\theta-1}(vu)^{\frac{-1}{\theta-1}}))^\theta-(-log(v))^\theta)^{\frac{1}{\theta}}}$

c function in dimension 2 $c_\theta(u, v) = \frac{C_\theta(u,v)}{uv}((-log(u)^\theta + (-log(v))^\theta)^{-2+\frac{2}{\theta}}[1 + (\theta - 1)((-log(u)^\theta + (-log(v))^\theta)^{-\frac{1}{\theta}}]$

C.2.3 Clayton Copula

Generator, generator inverse and derivatives $\theta \in [-1, \infty[\backslash\{0\}$

$\Phi_\theta(t) = (1 + \theta t)_+^{-\frac{1}{\theta}}$
$\Phi_\theta^{-1}(t) = \frac{1}{\theta}(t^{-\theta} - 1)$

$\Phi_\theta^{(n)}(t) = (-1)^n(1 + \theta t)^{-\frac{1+n\theta}{\theta}} \prod_{k=0}^{n-1}(1 + k\theta)$

C function in dimension 2 $C_\theta(u, v) = (max\{u^{-\theta} + v^{-\theta} - 1; 0\})^{-\frac{1}{\theta}}$

Partial derivative of C and inverse

$$h(u, v, \theta) = \frac{\partial C_\theta}{\partial v}(u, v) = \begin{cases} v^{-\theta-1}(u^{-\theta} + v^{-\theta} - 1)^{-\frac{1}{\theta}-1} & \text{if } u^{-\theta} + v^{-\theta} \geq 1 \\ 0 & \text{else} \end{cases}$$

$h^{-1}(u,v,\theta) = (u^{-\frac{\theta}{\theta+1}}v^{-\theta} - v^{\theta} + 1)^{-\frac{1}{\theta}}$ where u must be nonzero.

c function in dimension 2 $c_\theta(u,v) = (1+\theta)(uv)^{-1-\theta}(u^{-\theta} + v^{-\theta} - 1)^{-\frac{1}{\theta}-2}$

C.3 Empirical Copula

$$C(u_1,...,u_d) = \sum_{k=1}^{n} \mathbb{1}_{\forall i, F_i(x_k) \leq u_i}$$

where, F_i is the cdf of the marginal i.

C.4 Vine Copulas

C.4.1 Canonical Vine Copula

Global probability density function

$$f(x_1,...,x_d) = \prod_{k=1}^{d} f(x_k) \prod_{j=1}^{d-1} \prod_{i=1}^{d-j} c_{j,j+i|1,...,j-1}(F(x_j|x_1,...,x_{j-1}), F(x_{j+i}|x_1,...,x_{j-1}))$$

c function in dimension d By identifying c in the previous equation, we have :

$$c_{1,...,d}(F_1(x_1),...,F_d(x_d)) = \prod_{j=1}^{d-1} \prod_{i=1}^{d-j} c_{j,j+i|1,...,j-1}(F(x_j|x_1,...,x_{j-1}), F(x_{j+i}|x_1,...,x_{j-1}))$$

We now do an approximation of conditional independance :
$c_{j,j+i|1,...,j-1} = 1$ when $(1,...,j-1) \neq \emptyset \Leftrightarrow j \neq 1$

We obtain :

$$c_{1,...,d}(F_1(x_1),...,F_d(x_d)) = \prod_{i=1}^{d-1} c_{1,i+1}(F_1(x_1), F_{i+1}(x_{i+1}))$$

Then,

$$c_{1,...,d}(u_1,...,u_d) = \prod_{i=1}^{d-1} c_{1,i+1}(u_1, u_d)$$

C.4.2 D Vine Copula

Global probability density function

$$f(x_1,...,x_d) = \prod_{k=1}^{d} f(x_k) \prod_{j=1}^{d-1} \prod_{i=1}^{d-j} c_{i,i+j|i+1,...,i+j-1}(F(x_i|x_{i+1},...,x_{i+j-1}), F(x_{i+j}|x_{i+1},...,x_{i+j-1}))$$

c function in dimension d By identifying c in the previous equation, we have :

$$c_{1,...,d}(F_1(x_1),...,F_d(x_d)) = \prod_{j=1}^{d-1} \prod_{i=1}^{d-j} c_{i,i+j|i+1,...,i+j-1}(F(x_i|x_{i+1},...,x_{i+j-1}), F(x_{i+j}|x_{i+1},...,x_{i+j-1}))$$

We now do an approximation of conditional independance :
$c_{i,i+j|i+1,...,i+j-1} = 1$ when $(i+1,...,i+j-1) \neq \emptyset \Leftrightarrow j \neq 1$
We obtain :

$$c_{1,...,d}(F_1(x_1),...,F_d(x_d)) = \prod_{i=1}^{d-1} c_{i,i+1}(F_i(x_i), F_{i+1}(x_{i+1}))$$

Then,

$$c_{1,...,d}(u_1,...,u_d) = \prod_{i=1}^{d-1} c_{i,i+1}(u_i, u_{i+1})$$

D Directions for future work

- Testing other copulas with the weighted combined copulas, the code already permits it but as all copulas look the same it seems like there is no real interest for the moment.
- Testing more than two dimension and with vine copulas, the code already permit but it was enough difficult to deal with bidimensional copulas at first.
- Implementing the Joe copula which is a classical archemedian copula and maybe other relevant copulas.
- Making abstract classes ArchimedianCopula and EllipticalCopula which would inherit from CopulaBase and from which will inherit the appropriate copulas.
- Trying the experiment with kendall function. The problem is the slowliness of the algorithm but the code already permits it. It can be tried with an approximate empirical kendall function, (a bit like a MonteCarlo method). See [4]
- Thinking about segmentation. For example try different segmentations for input of marginals and input of copulas.
- Understanding why univariate epispline cannot fit small actuals

E EMD Demonstration

Demonstration :

First we have :

$$\begin{aligned} F &= \{(f_{i,j}) | f_{i,j} \geq 0, \sum_{i=1}^{n} f_{i,j} \leq 1, \sum_{j=1}^{n} f_{i,j} \leq 1, \sum_{i=1}^{n}\sum_{j=1}^{n} f_{i,j} = n\} \\ &= \{(f_{i,j}) | f_{i,j} \geq 0, \sum_{i=1}^{n} f_{i,j} = 1, \sum_{j=1}^{n} f_{i,j} = 1\} \end{aligned}$$

The way $\supset$ is trivial.

Let be $(f_{i,j})_{i,j} \in F$ and let us suppose that $\exists i_0, \sum_{i=1}^{n} f_{i_0,j} < 1 : \sum_{i=1}^{n} f_{i_0,j} = 1 - \epsilon$.
Thus, $\sum_{i=1}^{n}\sum_{j=1}^{n} f_{i,j} \leq 1 - \epsilon + \sum_{i=1, i\neq i_0}^{n} 1 = n - \epsilon < n$
$(f_{i,j})_{i,j} \notin F$: Contradiction

We will now demonstrate that it exists $f_{i,j}$ integers that solve the minimum problem. This a classical demonstration using several theorems. For any precision, see the chapter 2 of [8].

Let us define $M = (M_{k,(i,j)}) \in \mathcal{M}_{2n,n^2}(\mathbb{R})$ with :

$$M_{k,(i,j)} = \begin{cases} 1 & \text{if } k = i \\ -1 & \text{if } k = j + n \\ 0 & \text{else} \end{cases}$$

Because $1 \leq i, j \leq n$ the cases are incompatible.

We now have :

$$F = \{(f_{i,j}) | f_{i,j} \geq 0, Mf = B\}$$

where $B = \begin{pmatrix} 1 \\ \vdots \\ 1 \\ -1 \\ \vdots \\ -1 \end{pmatrix}$

The coefficient of M are just -1,0 and 1 and M has just once 1 and once -1 on each of its column.
Thanks to the Poincaré lemma (see [8] chapter 2), we can say that M is totally unimodulary. Because B has integer coefficient, we can say that all extreme points of F have integer coefficients.
So, we finally have it exists $f_{i,j}$ integers that solve the minimization problem.

Let $(f_{i,j})_{i,j}$ solve the mnimization problem with integer coefficients.
$\forall i, \forall j, 0 \leq f_{i,j} \leq 1$ and $f_{i,j} \in \mathbb{N} \Rightarrow f_{i,j} = 0$ or $f_{i,j} = 1$
$\forall i, \sum_{j=1}^{n} f_{i,j} = 1 \Rightarrow \exists! j_0, f_{i,j_0} = 1$
$\forall j, \sum_{i=1}^{n} f_{i,j} = 1 \Rightarrow \exists! i_0, f_{i_0,j} = 1$

We now have :

$$\exists \sigma \in \mathfrak{S}_n, f_{i,j} = \begin{cases} 1 & \text{if } j = \sigma(i) \\ 0 & \text{else} \end{cases}$$

So,

$$EMD(\mathbf{u}, \mathbf{v}) = \min_{\sigma \in \mathfrak{S}_n} \sum_{i=1}^{n} d_{i,\sigma i} = \min_{\sigma \in \mathfrak{S}_n} \sum_{i=1}^{n} |u_i - v_{\sigma(i)}|$$

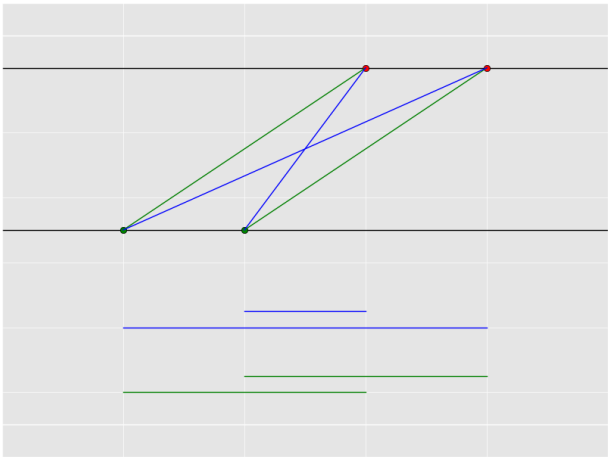

Figure 18: Graphic representation of case 1 : *The upper black line represents* $\boldsymbol{v}$ *and the lower one* $\boldsymbol{u}$*. The red points are from left to right* v_r *and* v_q *and The green points are from left to right* u_p *and* u_q*. The blue lines represent what the cost when using* σ *and the green lines the cost when using* $\tilde{\sigma}$*.*

We now have to prove that this min is reached when $(u_i)_i$ is sorted in the same order than $(v_{\sigma(i)})_i$. Since the indexes have symetric roles and are just notations indicating coefficients, we can consider that u and v are sorted :
$\forall i \leq j, u_i \leq u_j, v_i \leq v_j$
With this notation, we need to prove that this minimum is reached for $\sigma = Id$.

Suppose that $\sigma \neq Id$ reaches the minimum, so $supp(\sigma) \neq \emptyset$ and $supp\sigma$ contains at least two elements. Let us define:

$$q = \max supp(\sigma)$$
$$p = \sigma^{-1}(q)$$
$$r = \sigma(q)$$

We have $q \leq p$ and $r \leq p$, so $u_q \leq u_p$ and $v_r \leq v_p$.

- Case 1 : $u_p \leq u_q \leq v_r \leq v_q$

$$\begin{aligned} |u_p - v_q| + |u_q - v_r| &= v_q - u_p + v_r - u_q \\ &= v_r - u_p + v_q - u_q \\ &= |u_p - v_r| + |u_q - v_q| \end{aligned}$$

- Case 2 : $u_p \leq v_r \leq u_q \leq v_q$

$$\begin{aligned} |u_p - v_q| + |u_q - v_r| &\leq |u_p - v_q| \\ &= v_q - u_p \\ &= v_q - u_q + u_q - u_p \\ &\leq v_q - u_q + v_r - u_p \\ &= |u_q - v_q| + |u_p - v_r| \end{aligned}$$

- Case 3 : $u_p \leq v_r \leq v_q \leq u_q$

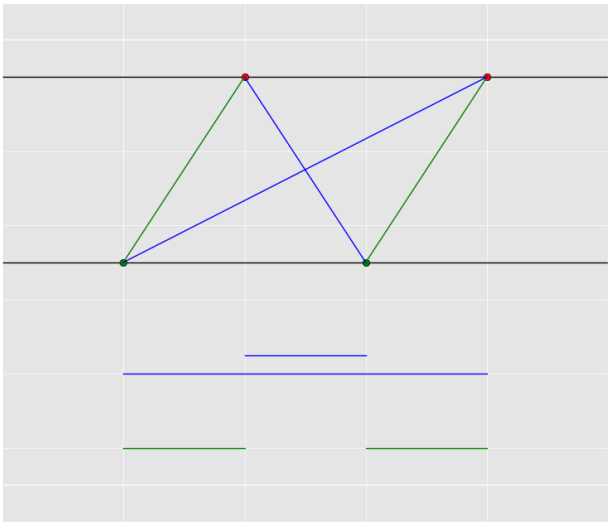

Figure 19: Graphic representation of case 2 : *The upper black line represents* $\boldsymbol{v}$ *and the lower one* $\boldsymbol{u}$*. The red points are from left to right* v_r *and* v_q *and The green points are from left to right* u_p *and* u_q*. The blue lines represent what the cost when using* σ *and the green lines the cost when using* $\tilde{\sigma}$*.*

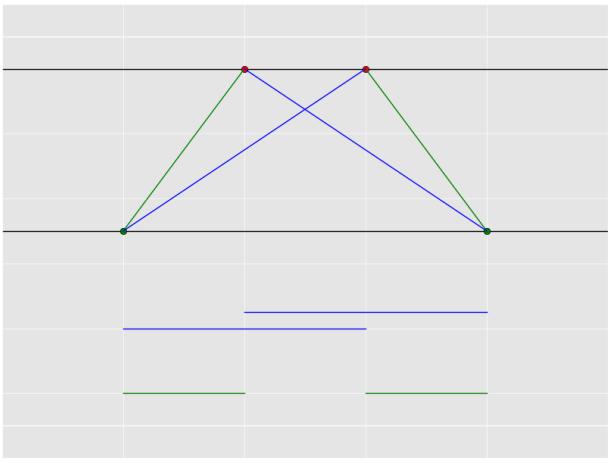

Figure 20: Graphic representation of case 3 : *The upper black line represents* $\boldsymbol{v}$ *and the lower one* $\boldsymbol{u}$*. The red points are from left to right* v_r *and* v_q *and The green points are from left to right* u_p *and* u_q*. The blue lines represent what the cost when using* σ *and the green lines the cost when using* $\tilde{\sigma}$*.*

$$\begin{aligned}|u_p - v_q| + |u_q - v_r| &= v_q - u_p + u_q - v_r \\ &= u_q - u_p + v_q - v_r \\ &\leq u_q - u_p \\ &= u_q - v_q + v_q - u_p \\ &\leq u_q - v_q + v_r - u_p \\ &= |u_q - v_q| + |u_p - v_r|\end{aligned}$$

- Case 4 : $v_r \leq v_q \leq u_p \leq u_q$
  Same as case 1 by inversing the symetric roles of $\mathbf{u}$ and $\mathbf{v}$.
- Case 5 : $v_r \leq u_p \leq v_q \leq u_q$
  Same as case 2 by inversing the symetric roles of $\mathbf{u}$ and $\mathbf{v}$.
- Case 6 : $v_r \leq u_p \leq u_q \leq v_q$
  Same as case 3 by inversing the symetric roles of $\mathbf{u}$ and $\mathbf{v}$.

In all this cases, we have :

$$|u_p - v_q| + |u_q - v_r| \leq |u_q - v_q| + |u_p - v_r|$$

Let us define $\tilde{\sigma} = \sigma \circ (pq)$:

$$\begin{aligned}\forall i \notin \{p, q\}, \tilde{\sigma}(i) &= \sigma(i) \\ \tilde{\sigma}(p) &= r \\ \tilde{\sigma}(q) &= q\end{aligned}$$

Then we have

$$\begin{aligned}\sum_{i=1}^{n} |u_i - v_{\sigma(i)}| &= \sum_{i \notin \{p,q\}} |u_i - v_{\sigma(i)}| + |u_p - v_q| + |u_q - v_r| \\ &\leq \sum_{i \notin \{p,q\}} |u_i - v_{\sigma(i)}| + |u_q - v_q| + |u_p - v_r| \\ &= \sum_{i=1}^{n} |u_i - v_{\tilde{\sigma}(i)}|\end{aligned}$$

So, $\tilde{\sigma}$ reaches the minimum too and $supp(\tilde{\sigma}) = supp(\sigma) \setminus \{\max supp(\sigma)\}$. By doing this operation many times, we can remove all the elements of $supp(\sigma)$. So Id reaches the minimum. □