





Copulas.jl: A fully Distributions.jl-compliant copula package

Oskar Laverny ¹ and Santiago Jimenez ²

¹ Aix Marseille Univ, Inserm, IRD, SESSTIM, Sciences Economiques & Sociales de la Santé & Traitement de l'Information Médicale, ISSPAM, Marseille, France. ² Federal University of Pernambuco 
Corresponding author

DOI: [10.21105/joss.06189](https://doi.org/10.21105/joss.06189)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: [Øystein Sørensen](#)  

Reviewers:

- [@lucaferranti](#)
- [@AnderGray](#)

Submitted: 17 November 2023

Published: 29 February 2024

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](https://creativecommons.org/licenses/by/4.0/)).

Summary

Copulas are functions that describe dependence structures of random vectors, without describing their univariate marginals. In statistics, the separation is sometimes useful, the quality and/or quantity of available information on these two objects might differ. This separation can be formally stated through Sklar's theorem:

Theorem: existence and uniqueness of the copula (Sklar, 1959): For a given d -variate absolutely continuous random vector \mathbf{X} with marginals X_1, \dots, X_d , there exists a unique function C , the copula, such that

$$F(\mathbf{x}) = C(F_1(x_1), \dots, F_d(x_d)),$$

where F, F_1, \dots, F_d are respectively the distributions functions of $\mathbf{X}, X_1, \dots, X_d$.

Copulas are standard tools in probability and statistics, with a wide range of applications from biostatistics, finance or medicine, to fuzzy logic, global sensitivity and broader analysis. A few standard theoretical references on the matter are Joe (1997), Nelsen (2006), Joe (2014), and Durante & Sempì (2015).

The Julia package `Copulas.jl` brings most standard copula-related features into native Julia: random number generation, density and distribution function evaluations, fitting, construction of multivariate models through Sklar's theorem, and many more related functionalities. Since copulas can combine arbitrary univariate distributions to form distributions of multivariate random vectors, we fully comply with the `Distributions.jl` API (Besançon et al., 2021; Lin et al., 2019), the Julian standard for implementation of random variables and random vectors. This compliance allows interoperability with other packages based on this API such as, e.g., `Turing.jl` (Ge et al., 2018) and several others.

Statement of need

The R package `copula` (Hofert et al., 2020; Ivan Kojadinovic & Jun Yan, 2010; Jun Yan, 2007; Marius Hofert & Martin Mächler, 2011) is the gold standard when it comes to sampling, estimating, or simply working around dependence structures. However, in other languages, the available tools are not as developed and/or not as recognized. We bridge the gap in the Julian ecosystem with this Julia-native implementation. Due to the very flexible type system in Julia, our code's expressiveness and tidiness will increase its usability and maintainability in the long-run. Type-stability allows sampling in arbitrary precision without requiring more code, and Julia's multiple dispatch yields most of the below-described applications.

There are competing packages in Julia, such as `BivariateCopulas.jl` (Gray et al., 2023) which only deals with a few models in bivariate settings but has very nice graphs, or

`DatagenCopulaBased.jl` (Domino, 2019, 2020; Domino et al., 2023; Domino & Glos, 2018), which only provides sampling and does not have exactly the same models as `Copulas.jl`. While not fully covering out both of these package's functionality (mostly because the three projects chose different implementation paths), `Copulas.jl` brings, as a key feature, the compliance with the broader ecosystem. The following table provides a feature comparison between the three:

	<code>Copulas.jl</code>	<code>DatagenCopulaBased.jl</code>	<code>BivariateCopulas.jl</code>
Distributions.jl's API	Yes	No	Yes
Fitting	Yes	No	No
Plotting	No	No	Yes
Available copulas			
- Classic Bivariate	Yes	Yes	Yes
- Classic Multivariate	Yes	Yes	No
- Archimedean	All of them	Selected ones	Selected ones
- Obscure Bivariate	Yes	No	No
- Archimedean Chains	No	Yes	No

Since our primary target is maintainability and readability of the implementation, we have not considered the efficiency and the performance of the code yet. However, a (limited in scope) benchmark on Clayton's pdf shows competitive behavior of our implementation w.r.t `DatagenCopulaBased.jl` (but not `BivariateCopulas.jl`). To perform this test we use the `BenchmarkTools.jl` (Chen & Revels, 2016) package and generate 10^6 samples for Clayton copulas of dimensions 2, 5, 10 with parameter 0.8. The execution times (in seconds) are given below:

	2	5	10
<code>Copulas.Clayton</code>	1.1495578	1.3448951	1.8044065
<code>BivariateCopulas.Clayton</code>	0.1331608	X	X
<code>DatagenCopulaBased.Clayton</code>	1.9868345	2.4276321	2.8009263

Code for these benchmarks is available in the repository.

Examples

SklarDist: sampling and fitting examples

The `Distributions.jl`'s API provides a fit function. You may use it to simply fit a compound model to some dataset as follows:

```
using Copulas, Distributions, Random
```

```
# Define the marginals and the copula, then use Sklar's theorem:
```

```
X1 = Gamma(2,3)
X2 = Pareto(0.5)
X3 = Normal(10,0.8)
C = ClaytonCopula(3,0.7)
D = SklarDist(C, (X1, X2, X3))
```

```
# Sample as follows:
```

```
x = rand(D, 1000)
```

```
# You may estimate the model as follows:  
D̂ = fit(SklarDist{ClaytonCopula, Tuple{Gamma, Pareto, Normal}}, x)
```

The API does not fix the fitting procedure, and only loosely specifies it, thus the implemented default might vary on the copula. If you want more control, you may turn to Bayesian estimation using Turing.jl:

```
using Turing  
@model function model(dataset)  
  # Priors  
  θ ~ TruncatedNormal(1.0, 1.0, 0, Inf)  
  γ ~ TruncatedNormal(1.0, 1.0, 0.25, Inf)  
  η ~ Beta(1,1)  
  δ ~ Exponential(1)  
  
  # Define the model through Sklar's theorem:  
  X1 = Gamma(2,θ)  
  X2 = Pareto(γ)  
  X3 = Binomial(10,η)  
  C = ClaytonCopula(3,δ)  
  D = SklarDist(C, (X1, X2, X3))  
  
  # Add the loglikelihood to the model :  
  Turing.Turing.@addlogprob! loglikelihood(D, dataset)  
end
```

The Archimedean interface

Archimedean copulas form a large class of copulas that has seen a lot of theoretical work. Among others, you may take a look at (McNeil & Nešlehová, 2009). We use `WilliamsonTransforms.jl`'s implementation of the Williamson d -transform to sample from any Archimedean copula, including for example the ClaytonCopula with negative dependence parameter in any dimension, which is a first to our knowledge.

To construct an Archimedean copula, you first need to reference its generator through the following API:

```
struct MyGenerator{T} <: Copulas.Generator  
  θ::T  
end  
φ(G::MyGenerator, t) = exp(-G.θ * t)  
Copulas.max_monotony(G::MyGenerator) = Inf  
C = ArchimedeanCopula(4, MyGenerator(1.3)) # 4-dimensional copula
```

The obtained model automatically gets all copula functionalities (pdf, cdf, sampling, dependence measures, etc...). We nevertheless have specific implementation for a (large) list of known generators, and you may implement some other methods if you know closed form formulas for more performance. The use of the (inverse) Williamson d -transform allows the technical boundaries of our Archimedean implementation to *match* the necessary and sufficient conditions for a generator to produce a genuine Archimedean copula.

Broader ecosystem

The package is starting to get used in several other places of the ecosystem. Among others, we noted:

- The package `GlobalSensitivity.jl` exploits `Copulas.jl` to provide Shapley effects implementation, see [this documentation](#).

- `EconomicScenarioGenerators.jl` uses `Copulas.jl`'s dependence structures to construct multivariate financial assets.

Acknowledgement

Santiago Jiménez Ramos thanks FACEPE for the full financing of his postgraduate studies.

References

- Besaçon, M., Papamarkou, T., Anthoff, D., Arslan, A., Byrne, S., Lin, D., & Pearson, J. (2021). `Distributions.jl`: Definition and modeling of probability distributions in the JuliaStats ecosystem. *Journal of Statistical Software*, 98(16), 1–30. <https://doi.org/10.18637/jss.v098.i16>
- Chen, J., & Revels, J. (2016). Robust benchmarking in noisy environments. *arXiv e-Prints*. <https://arxiv.org/abs/1608.04295>
- Domino, K. (2019). *Selected methods for non-Gaussian data analysis*. <https://arxiv.org/abs/1811.10486>
- Domino, K. (2020). Multivariate cumulants in outlier detection for financial data analysis. *Physica A: Statistical Mechanics and Its Applications*, 558, 124995.
- Domino, K., Adam, Laverny, O., & TagBot, J. (2023). `Itis/DatagenCopulaBased.jl`: v1.4.2 (Version v1.4.2). Zenodo. <https://doi.org/10.5281/zenodo.7944064>
- Domino, K., & Glos, A. (2018). *Introducing higher order correlations to marginals' subset of multivariate data by means of Archimedean copulas*. <https://arxiv.org/abs/1803.07813>
- Durante, F., & Sempi, C. (2015). *Principles of copula theory*. Chapman and Hall/CRC. <https://doi.org/10.1201/b18674>
- Ge, H., Xu, K., & Ghahramani, Z. (2018). Turing: A language for flexible probabilistic inference. *International Conference on Artificial Intelligence and Statistics, AISTATS 2018, 9-11 April 2018, Playa Blanca, Lanzarote, Canary Islands, Spain*, 1682–1690. <http://proceedings.mlr.press/v84/ge18b.html>
- Gray, A., Behrendorf, J., amrods, & Schilling, C. (2023). `AnderGray/BivariateCopulas.jl`: 0.1.5 (Version v0.1.5). Zenodo. <https://doi.org/10.5281/zenodo.10412898>
- Hofert, M., Kojadinovic, I., Maechler, M., & Yan, J. (2020). *Copula: Multivariate dependence with copulas*. <https://CRAN.R-project.org/package=copula>
- Ivan Kojadinovic, & Jun Yan. (2010). Modeling multivariate distributions with continuous margins using the copula R package. *Journal of Statistical Software*, 34(9), 1–20. <https://doi.org/10.18637/jss.v034.i09>
- Joe, H. (1997). *Multivariate models and multivariate dependence concepts*. CRC press. <https://doi.org/10.1201/9780367803896>
- Joe, H. (2014). *Dependence modeling with copulas*. CRC press.
- Jun Yan. (2007). Enjoy the joy of copulas: With a package copula. *Journal of Statistical Software*, 21(4), 1–21. <https://doi.org/10.18637/jss.v021.i04>
- Lin, D., White, J. M., Byrne, S., Bates, D., Noack, A., Pearson, J., Arslan, A., Squire, K., Anthoff, D., Papamarkou, T., Besaçon, M., Drugowitsch, J., Schauer, M., & contributors, other. (2019). *JuliaStats/Distributions.jl: a Julia package for probability distributions and associated functions*. <https://doi.org/10.5281/zenodo.2647458>

- Marius Hofert, & Martin Mächler. (2011). Nested Archimedean copulas meet R: The nacopula package. *Journal of Statistical Software*, 39(9), 1–20. <https://doi.org/10.18637/jss.v039.i09>
- McNeil, A. J., & Nešlehová, J. (2009). Multivariate Archimedean copulas, d -monotone functions and L1 -norm symmetric distributions. *The Annals of Statistics*, 37(5B), 3059–3097. <https://doi.org/10.1214/07-AOS556>
- Nelsen, R. B. (2006). *An introduction to copulas* (2nd ed). Springer. ISBN: 978-0-387-28659-4
- Sklar, A. (1959). Fonction de répartition dont les marges sont données. *Inst. Stat. Univ. Paris*, 8, 229–231.