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.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Øystein Sørensen

Reviewers:

- @lucaferranti
- @AnderGray

Submitted: 17 November 2023
Published: unpublished

Summary

Copulas are functions that describe dependence structures of random vectors, without describing their univariate marginals. In statistics, the separatiopn is sometimes usefull, 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: existance and uniqueness of the copula (Sklar, 1959): For a given d-variate absolutely continuous random vector $\mathbf{X}$ with marginals $X_1, ...X_d$, there exists a unique function C, the copula, such that

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

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

Copulas are standard tools in probability and statistics, with a wide range of applications from biostatistics, finance or medecine, 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 & Sempi, 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. Copulas being fundamentally distributions of 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 complience 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 developped and/or not as recognised. We bridge the gap in the Julian ecosystem with this Julia-native implementation. Due to the very flexible type system in Julia, our code expressiveness and tidyness will increase its usability and maintenability in the long-run. Type-stability allows sampling in arbitrary precision without requiering more code, and Julia's multiple dispatch yields most of the below-described applications.

There are competing packages in Julia, such as `BivariateCopulas.jl` which only deals with a few models in bivariate settings but has very nice graphs, or `DatagenCopulaBased.jl`, 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 copulas to implement), `Copulas.jl` is clearly the must fully featured, and brings, as a key feature, the complience with the broader ecosystem.

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:

```julia
using Copulas, Distributions, Random

# Define the marginals and the copula, then use Sklar's theorem:
X□ = Gamma(2,3)
X□ = Pareto(0.5)
X□ = Binomial(10,0.8)
C = ClaytonCopula(3,0.7)
X = SklarDist(C,(X□,X□,X□))

# Sample from the model:
x = rand(D,1000)

# You may estimate the model as follows:
D□ = fit(SklarDist{FrankCopula,Tuple{Gamma,Normal,Binomial}}, x)
# Although you'll probbaly get a bad fit !
```

The API does not fix the fitting procedure, and only loosely specify it, thus the implemented default might vary on the copula. If you want more control, you may turn to bayesian estimation using `Turing.jl` (Ge et al., 2018):

```julia
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:
  X□ = Gamma(2,θ)
  X□ = Pareto(γ)
  X□ = Binomial(10,η)
  C = ClaytonCopula(3,δ)
  X = SklarDist(C,(X□,X□,X□))

  # Add the loglikelyhood to the model :
  Turing.Turing.@addlogprob! loglikelihood(D, dataset)
end
```

The Archimedean API

Archimedean copulas are a huge family of copulas that has seen a lot of theoretical work. Among others, you may take a look at (McNeil & Nešlehová, 2009). We use `WilliamsonTransformations.jl`'s implementation of the Williamson d-transfrom 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.

The API is consisting of the folloiwng functions:

```
□(C::MyArchimedean, t) # Generator
williamson_dist(C::MyArchimedean) # Williamson d-transform
```

So that implementing your own archimedean copula only requires to subset the `ArchimedeanCopula` type and provide your generator as follows:

```
struct MyUnknownArchimedean{d,T} <: ArchimedeanCopula{d}
    θ::T
end
□(C::MyUnknownArchimedean,t) = exp(-t*C.θ)
```

The obtained model can be used as follows:

```
C = MyUnknownCopula{2,Float64}(3.0)
spl = rand(C,1000)   # sampling
cdf(C,spl)           # cdf
pdf(C,spl)           # pdf
loglikelihood(C,spl) # llh
```

The following functions have defaults but can be overridden for performance:

```
□⁻¹(C::MyArchimedean, t) # Inverse of □
□⁽¹⁾(C::MyArchimedean, t) # first defrivative of □
□⁽□⁾(C::MyArchimedean,t) # dth defrivative of □
τ(C::MyArchimedean) # Kendall tau
τ⁻¹(::Type{MyArchimedean},τ) = # Inverse kendall tau
fit(::Type{MyArchimedean},data) # fitting.
```

Broader ecosystem

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

- The package `GlobalSensitivity.jl` exploit `Copulas.jl` to provide Shapley effects implementation, see this documentation.
- `EconomicScenarioGenerators.jl` uses depndence structures between financial assets.

Acknowledgments

References

Besançon, 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

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

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., Besançon, 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.