Journal of Statistical Software

May 2020, Volume 93, Issue 10. doi: 10.18637/jss.v093.i10

Hierarchical Archimedean Copulas for MATLAB and Octave: The HACopula Toolbox

Jan Górecki
Silesian University in Opava

Marius Hofert
University of Waterloo

Martin Holeňa
Academy of Sciences of the Czech Republic

Abstract

To extend the current implementation of copulas in MATLAB to non-elliptical distributions in arbitrary dimensions enabling for asymmetries in the tails, the toolbox **HACopula** provides functionality for modeling with hierarchical (or nested) Archimedean copulas. This includes their representation as MATLAB objects, evaluation, sampling, estimation and goodness-of-fit testing, as well as tools for their visual representation or computation of corresponding matrices of Kendall's tau and tail dependence coefficients. These are first presented in a quick-and-simple manner and then elaborated in more detail to show the full capability of **HACopula**. As an example, sampling, estimation and goodness-of-fit of a 100-dimensional hierarchical Archimedean copula is presented, including a speed up of its computationally most demanding part. The toolbox is also compatible with Octave, where no support for copulas in more than two dimensions is currently provided.

Keywords: copula, hierarchical Archimedean copula, structure, family, estimation, collapsing, sampling, goodness-of-fit, Kendall's tau, tail dependence, MATLAB, Octave.

1. Introduction

According to Sklar (1959), any continuous d-variate distribution function F can be uniquely decomposed through

$$F(x_1, \dots, x_d) = C(F_1(x_1), \dots, F_d(x_d)), \quad (x_1, \dots, x_d)^\top \in \mathbb{R}^d, \tag{1}$$

into its continuous univariate margins $F_1, \dots, F_d$ and its copula $C : [0,1]^d \to [0,1]$; the copula C itself is a d-variate distribution function with standard uniform univariate margins. This, on the one hand, allows one to study multivariate distribution functions independently of the margins, which is of particular interest in statistical applications. On the other hand, Sklar's

Theorem provides a tool for constructing large classes of multivariate distributions and is therefore often used for sampling multivariate distributions via copulas, which is indispensable for many applications in the realm of risk management, finance and insurance. Standard introductory monographs about copulas are, e.g., Nelsen (2006) and Joe (2014).

Apart from *elliptical copulas*, i.e., the copulas arising from elliptical distributions via Sklar's Theorem, *Archimedean copulas* (ACs) are a popular choice. In contrast to elliptical copulas, they are given explicitly in terms of a univariate function called generator. Another desirable property is their ability to capture different kinds of tail dependencies, e.g., only upper tail dependence and no lower tail dependence or both lower and upper tail dependence but of different magnitude. With a wide set of available parameter estimators, e.g., see Hofert, Mächler, and McNeil (2013), and the algorithm of Marshall and Olkin (1988), ACs are usually easy both to estimate and to sample. The functional symmetry in their arguments, also referred to as *exchangeability*, however, is often considered to be a drawback, e.g., in risk-management applications where the considered portfolios are typically high-dimensional. To circumvent exchangeability, or, in other words, to allow for different multivariate margins, ACs can be *nested* within each other under certain conditions, which results in hierarchical dependence structures. This has also led to their name, *hierarchical* (or *nested*) *Archimedean copulas* (HACs).

As has been recently shown in an empirical study concerning risk management in Okhrin, Ristig, and Xu (2017), such hierarchical constructions enable one to construct copula models that can compete with other recently popular multivariate copula models like *vine* or *factor* copulas. For their recent application to modeling dependence between so-called loss triangles, see Côté, Genest, and Abdallah (2016). Outside of finance, e.g., Górecki, Hofert, and Holeňa (2016) introduce HACs to Bayesian classification. A detailed analysis of their theoretical properties can be found in McNeil (2008); Savu and Trede (2010); Hofert (2011); Okhrin, Okhrin, and Schmid (2013). Considering their sampling, the R (R Core Team 2020) package **copula** (see Jun Yan 2007; Kojadinovic and Yan 2010; Hofert and Mächler 2011; Hofert, Kojadinovic, Maechler, and Yan 2020) offers an implementation of the approaches proposed in Hofert (2011, 2012). For estimating HACs, one of the most advanced frameworks for this purpose is offered by the R package **HAC**, see Okhrin and Ristig (2014, 2019). One can also find packages implementing HACs in proprietary statistical software. As an example, sampling procedures involving three popular families of HACs have been implemented in SAS (SAS Institute Inc. 2013); see Baxter and Huddleston (2014, p. 531). In MATLAB (The MathWorks Inc. 2019), which also provides some support for two popular multivariate elliptical and three popular bivariate Archimedean families, however, no support for HACs is provided. The latter also applies to (open-source) Octave (Eaton, Bateman, Hauberg, and Wehbring 2019), which moreover restricts the provided functionality only to several families of bivariate copulas and does not implement any copula estimators at all.

To fill these gaps, the **HACopula** toolbox for MATLAB and Octave introduces a comprehensive framework focused particularly on HACs. It implements procedures concerning sampling, estimation and goodness-of-fit testing. Not only do these procedures cover the basic features of the aforementioned packages, but also offer an implementation of all estimators recently introduced in Górecki, Hofert, and Holeňa (2017b), which are, to the best of our knowledge, the only estimators enabling for estimation of all three components of a HAC, i.e., its structure, the families of its generators and its parameters. As the estimators from Górecki *et al.* (2017b) introduce several so-called *collapsing* and *re-estimation* procedures, which, generally,

turn binary HAC structures (binary trees often resulting from estimation processes) into non-binary ones (allowing to access all possible HAC structures), these are also provided by the toolbox. To allow for better understanding of how a HAC model has resulted from an estimation process, the toolbox provides detailed tracing of the estimation process, which is also a feature not available in other implementations covering HACs. Finally, as ACs are a special case of HACs, the new toolbox inherently complements their implementation in MATLAB and Octave (currently MATLAB R2019a and Octave 5.2.0) limited to the bivariate case and enables for AC modeling in an arbitrary dimension.

The paper is organized as follows. Section 2 provides an introduction to HACs. In Section 3, the reader gets in touch with the main capabilities of the toolbox, namely with the way HAC models can be constructed, evaluated, sampled, estimated and goodness-of-fit tested. To get a more detailed insight, Section 4 elaborates on the examples described in Section 3 and outlines further features of the toolbox, namely the representation of HAC models, how to cope with negative correlation in data, and how to access the estimators provided by the toolbox. As a proof of feasibility in high dimensions, sampling, estimation and goodness-of-fit of a 100-variate HAC is presented in Section 5, which also includes a way how to speed up its computationally most demanding part. Section 6 concludes.

2. Hierarchical Archimedean copulas

An *Archimedean generator*, or simply *generator*, is a continuous, decreasing function $\psi : [0,\infty) \to [0,1]$ that is strictly decreasing on $[0, \inf\{t : \psi(t) = 0\}]$ and satisfies $\psi(0) = 1$ and $\lim_{t\to\infty} \psi(t) = 0$. If $(-1)^k \psi^{(k)}(t) \geq 0$ for all $k \in \mathbb{N}$, $t \in [0,\infty)$, then ψ is called *completely monotone*; see Kimberling (1974) or Hofert (2010, p. 54). As follows from McNeil and Nešlehová (2009), given a completely monotone generator ψ, the function $C_\psi : [0,1]^d \to [0,1]$ defined by

$$C_\psi(u_1,\dots,u_d) = \psi(\psi^{-1}(u_1) + \dots + \psi^{-1}(u_d)), \tag{2}$$

where ψ^{-1} is the generalized inverse of ψ given by $\psi^{-1}(s) = \inf\{t \in [0,\infty] \mid \psi(t) = s\}, s \in [0,1]$, is a d-dimensional *Archimedean copula* (d-AC) for any $d \geq 2$. In what follows, we assume generators to be completely monotone.

In practice, a generator is mostly assumed to belong to a parametric family. Due to this reason, a generator from a family a with a real parameter θ will be denoted by $\psi^{(a,\theta)}$. Our toolbox implements nine out of the 22 families of Nelsen (2006, p. 116), see Table 1, i.e., we consider $a \in \{\text{A, C, F, G, J, 12, 14, 19, 20}\}$, where the first five family labels denote the popular families of Ali-Mikhail-Haq, Clayton, Frank, Gumbel and Joe, and the last four families are special cases of the families known as BB1 (12 and 14) and BB2 (19 and 20), see Joe (2014, p. 435, Table A.1) for the connection. This subset of families is also chosen since not all of those 22 families can be nested into each other in order to get a proper HAC; see Górecki *et al.* (2017b) for details.

Given a bivariate AC $C_{\psi^{(a,\theta)}}$, there exists a 1-to-1 functional relationship between the parameter θ and Kendall's tau (τ) that can be expressed either in a closed form, e.g., $\tau = \theta/(\theta+2)$ for the Clayton family ($a = \text{C}$), or as a one-dimensional integration; see Table 3 in Górecki *et al.* (2017b) for the family 20 and Hofert (2010, p. 65) for the rest of the families in Table 1. In the following, we denote this relationship by $\tau_{(a)}$, e.g., $\tau_{(\text{C})}(\theta) = \theta/(\theta+2)$.

a	Θ_a	$\psi^{(a,\theta)}(t)$	SNC	λ_l	λ_u
A	$[0,\,1)$	$(1-\theta)/(\mathrm{e}^t-\theta)$	$\theta_1 \le \theta_2$	0	0
C	$(0,\infty)$	$(1+t)^{-1/\theta}$	$\theta_1 \le \theta_2$	$2^{-1/\theta}$	0
F	$(0,\infty)$	$-\log(1-(1-e^{-\theta})\exp(-t))/\theta$	$\theta_1 \le \theta_2$	0	0
G	$[1,\infty)$	$\exp(-t^{1/\theta})$	$\theta_1 \le \theta_2$	0	$2-2^{1/\theta}$
J	$[1,\infty)$	$1-(1-\exp(-t))^{1/\theta}$	$\theta_1 \le \theta_2$	0	$2-2^{1/\theta}$
12	$[1,\infty)$	$(1+t^{1/\theta})^{-1}$	$\theta_1 \le \theta_2$	$2^{-1/\theta}$	$2-2^{1/\theta}$
14	$[1,\infty)$	$(1+t^{1/\theta})^{-\theta}$	unknown	$1/2$	$2-2^{1/\theta}$
19	$(0,\infty)$	$\theta/\ln\left(t+\mathrm{e}^{\theta}\right)$	$\theta_1 \le \theta_2$	1	0
20	$(0,\infty)$	$\ln^{-1/\theta}(t+\mathrm{e})$	$\theta_1 \le \theta_2$	1	0

Table 1: The nine families of generators implemented in **HACopula**. The table contains the family label (a), the corresponding parameter range $\Theta_a \subseteq [0,\infty)$, the form of $\psi^{(a,\theta)}$, the corresponding sufficient nesting condition (SNC, defined in the last paragraph of Section 2) involving two generators $\psi^{(a,\theta_1)}, \psi^{(a,\theta_2)}$, and the lower and upper tail-dependence coefficients $\lambda_l(\theta) = \lim_{t\downarrow 0} C_{\psi^{(a,\theta)}}(t,t)/t$ and $\lambda_u(\theta) = \lim_{t\downarrow 0}(1-2t+C_{\psi^{(a,\theta)}}(t,t))/(1-t)$, respectively, where $C_{\psi^{(a,\theta)}}$ is a 2-AC; see Section 1.7.4 in Hofert (2010).

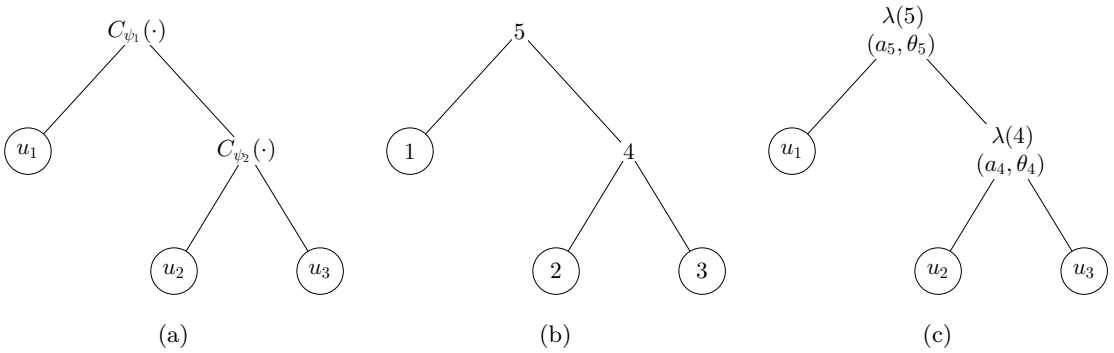


Figure 1: (a) A tree-like representation of a 3-variate HAC given by $C_{\psi_1,\psi_2}(u_1,u_2,u_3) = C_{\psi_1}(u_1, C_{\psi_2}(u_2,u_3))$. (b) An undirected tree $(\mathcal{V},\mathcal{E})$, $\mathcal{V} = \{1,\dots,5\}$, $\mathcal{E} = \{\{1,5\},\{2,4\},\{3,4\},\{4,5\}\}$ derived for the tree representation in Figure 1(a). (c) Our representation of $C_{(\mathcal{V},\mathcal{E},\lambda)}(u_1,u_2,u_3) = C_{\lambda(5)}(u_1, C_{\lambda(4)}(u_2,u_3))$, where $\lambda(4) = \psi^{(a_4,\theta_4)}$ and $\lambda(5) = \psi^{(a_5,\theta_5)}$ and $(\mathcal{V},\mathcal{E})$ is given by Figure 1(b).

As has already been mentioned in the introduction, to construct a *hierarchical Archimedean copula* (HAC), one just needs to replace some arguments in an AC by other (H)ACs, see Joe (1997) or Hofert (2011). Hence, e.g., given two 2-ACs C_{ψ_1} and C_{ψ_2}, a 3-variate HAC, denoted C_{ψ_1,ψ_2}, can be constructed by

$$C_{\psi_1,\psi_2}(u_1,u_2,u_3) = C_{\psi_1}(u_1, C_{\psi_2}(u_2,u_3)). \tag{3}$$

A tree representation of such a construction is depicted in Figure 1(a). Using the language of graph theory, an *undirected tree* $(\mathcal{V},\mathcal{E})$ related to this representation, where $\mathcal{V}$ is a set of nodes $\{1,\dots,m\}$, $m \in \mathbb{N}$, and $\mathcal{E} \subset \mathcal{V}\times\mathcal{V}$, can be derived by enumerating all of its nodes, e.g., in Figure 1(b), where $\mathcal{V} = \{1,\dots,5\}$ and $\mathcal{E} = \{\{1,5\},\{2,4\},\{3,4\},\{4,5\}\}$. As one can observe, not all nodes correspond to the same type of objects: The *leaves* $\{1,2,3\}$ correspond

to the variables u_1, u_2 and u_3, whereas the non-leaf nodes $\{4, 5\}$, called *forks*, correspond to the ACs (uniquely determined by the corresponding generators) nested in C_{ψ_1,ψ_2}. Note that when deriving a particular (undirected) tree for the tree representation in Figure 1(a), we assume that

1. the leaves 1, 2 and 3 in Figure 1(b) correspond to u_1, u_2 and u_3 in Figure 1(a), respectively, and that
2. the fork indices (4 and 5) are set arbitrarily, i.e., one can also derive an undirected tree where the fork indices 4 and 5 are switched.

Also, as each fork corresponds to a generator, we represent this relationship using a labeling denoted λ, which maps the forks to the corresponding generators. In our example,

$$\lambda(4) = \psi_2 \text{ and } \lambda(5) = \psi_1. \tag{4}$$

Using this notation, (3) can be rewritten as

$$C_{\lambda(5)}(u_1, C_{\lambda(4)}(u_2, u_3)).$$

Observe that the indices of the arguments of the inner copula $C_{\lambda(4)}$ correspond to the set of children of fork 4, i.e., to $\{2, 3\}$, and the indices of the arguments of the outer copula $C_{\lambda(5)}$ correspond to the set of children of fork 5, i.e., to $\{1, 4\}$ (4 represents the the inner copula $C_{\lambda(4)}(u_2, u_3)$). This implies that one can express $C_{\psi_1,\psi_2}(u_1, u_2, u_3)$ in terms of the triplet $(\mathcal{V}, \mathcal{E}, \lambda)$. Following this observation, we denote this HAC in an arbitrary dimension by $C_{(\mathcal{V},\mathcal{E},\lambda)}$. For a definition of HACs in this way, see Definition 3.1 in Górecki *et al.* (2017b). For clarity, just recall that the *descendants* of a node $v \in \mathcal{V}$ is the set of nodes consisting of all children of v, all children of all children of v, etc., whereas the *ancestors* of a node $v \in \mathcal{V}$ is the set of nodes consisting of the parent of v, the parent of the parent of v, etc.

As mentioned above, in practice, generators are typically assumed to be members of one-parametric families. E.g., assume that $\lambda(4)$ and $\lambda(5)$ are members of such families denoted by a_4 and a_5 with parameters θ_4 and θ_5, respectively, i.e., $\lambda(i) = \psi^{(a_i,\theta_i)}$, $i \in \{4, 5\}$. Using this notation, the graphical representation depicted in Figure 1(c) fully determines the parametric HAC $C_{(\mathcal{V},\mathcal{E},\lambda)} = C_{\psi_1,\psi_2}$ given by (3) and (4), i.e., its structure, the families of its generators and its parameters, and we will use this notation in this way in arbitrary dimensions.

To guarantee that a proper copula results from nesting ACs, we will use the *sufficient nesting condition* (SNC) proposed by Joe (1997, p. 87) and McNeil (2008). It states that if for all parent-child pairs of forks (i, j) appearing in a nested construction $C_{(\mathcal{V},\mathcal{E},\lambda)}$ the first derivative of $\lambda(i)^{-1}(\lambda(j)(t))$ is completely monotone, then $C_{(\mathcal{V},\mathcal{E},\lambda)}$ is a copula. This SNC has three important practical advantages, which are that, for many pairs from the 22 families discussed above, 1) its expression in terms of the corresponding parameters is known, 2) this expression does not depend on d for all pairs for which it is known, see Tables 1 and 2, and, most importantly, 3) efficient sampling strategies based on a stochastic representation for HACs satisfying the SNC are known; see Hofert (2011) or Hofert (2012). Note that Table 2 lists *all* family combinations of the generators of Nelsen (2006, pp. 116–119) that result in proper HACs according to the SNC, see Hofert (2008) or Theorem 4.3.2 in Hofert (2010) for more details. Note that there also exists a weaker sufficient condition, see Rezapour (2015), which however lacks those three advantages and is thus of limited practical use.

(a_1, a_2)	Θ_{a_1}	Θ_{a_2}	SNC
(A, C)	$[0, 1)$	$(0, \infty)$	$\theta_2 \in [1, \infty)$
(A, 19)	$[0, 1)$	$(0, \infty)$	any θ_1, θ_2
(A, 20)	$[0, 1)$	$(0, \infty)$	$\theta_2 \in [1, \infty)$
(C, 12)	$(0, \infty)$	$[1, \infty)$	$\theta_1 \in (0, 1]$
(C, 14)	$(0, \infty)$	$[1, \infty)$	$\theta_1\theta_2 \in (0, 1]$
(C, 19)	$(0, \infty)$	$(0, \infty)$	$\theta_1 \in (0, 1]$
(C, 20)	$(0, \infty)$	$(0, \infty)$	$\theta_1 \leq \theta_2$

Table 2: All family combinations of the generators of Nelsen (2006, pp. 116–119) that result in proper HACs according to the sufficient nesting condition (SNC); see Hofert (2010, Theorem 4.3.2). The table contains the family labels in a parent-child family combination (a_1, a_2) with the corresponding parameter ranges Θ_{a_1} and Θ_{a_2}. The last column contains the SNC in terms of the parameters of a parent-child pair of generators $\psi^{(a_1,\theta_1)}$ and $\psi^{(a_2,\theta_2)}$, where $\theta_1 \in \Theta_{a_1}$ and $\theta_2 \in \Theta_{a_2}$.

3. A quick example

The aim of this section is to allow the reader to quickly get in touch with the main capabilities of the **HACopula** toolbox. An illustrative example is provided, showing how to construct and evaluate a HAC model, how to generate a sample from this model, how to compute a HAC estimate based on this sample, and finally, how to measure accordance between the estimate and the model or alternatively between the estimate and the sample. Note that the example can be reproduced using the file `quickex.m` in the folder `Demos`. Also note that all the presented results are obtained using MATLAB R2019a, so, if the example is executed in Octave, some slight discrepancies considering formatting of the results might be observed, e.g., Octave per default shows more digits after the decimal point than MATLAB, or in plots, where Octave currently (version 5.2.0) does not implement all necessary features corresponding to the ones provided by MATLAB. Finally note that as the example involves Archimedean families that are, to our best knowledge, not supported by other software packages covering HACs, the results reported below in this section can be reproduced only with the **HACopula** toolbox. However, in cases where an analogue to some discussed function provided by the toolbox is available in other software packages (we consider the R packages **copula** and **HAC**), this is noted accordingly.

3.1. Installing the HACopula toolbox

To install the toolbox, it is enough to unpack the files in a folder and to add the folder `HACopula` with its subfolders to the MATLAB or Octave path. This can be done, in MATLAB, with the button *Home/Set Path*, and in Octave, with the following code provided the current folder is the folder `HACopula`.

```
addpath(genpath(pwd));
savepath;
```

Note that the subfolder `@HACopula`, which is a *method folder* containing the methods of the class '`HACopula`', is implicitly covered in the MATLAB (Octave) path and thus not explicitly shown among the folders that have been added.

For the full functionality in MATLAB, the toolbox requires the **Statistics and Machine Learning Toolbox** and the **Symbolic Math Toolbox**. In Octave, it suffices to install and to load the **statistics** package, see Octave-Forge (2018), and the symbolic package **OctSymPy**, see Macdonald (2017). Finally note that the toolbox has been intensively tested with the MATLAB versions R2017a and R2019a, and its compatibility with Octave has been tested with the version 5.2.0.

3.2. Constructing HACs

The construction of a HAC model with the **HACopula** toolbox reflects the theoretical construction of HACs, in which several ACs are nested. For illustration, we consider a 7-variate HAC $C_{(\mathcal{V},\mathcal{E},\lambda)}$ composed of the four ACs $C_{\lambda(8)},\ldots,C_{\lambda(11)}$ given by

$$\begin{aligned}\lambda(8) &= \psi^{(12,\tau_{(12)}^{-1}(0.8))},\\ \lambda(9) &= \psi^{(19,\tau_{(19)}^{-1}(0.7))},\\ \lambda(10) &= \psi^{(12,\tau_{(12)}^{-1}(0.5))},\\ \lambda(11) &= \psi^{(\mathrm{C},\tau_{(\mathrm{C})}^{-1}(0.2))}.\end{aligned} \tag{5}$$

To clarify this definition, which maps the forks in $(\mathcal{V},\mathcal{E})$ to the corresponding generators, note that $\{1,\ldots,7\}$ is the set of leaves in $(\mathcal{V},\mathcal{E})$, and, as each AC (generator) nested in $C_{(\mathcal{V},\mathcal{E},\lambda)}$ corresponds to one fork in $(\mathcal{V},\mathcal{E})$, $\{8,\ldots,11\}$ is the set of forks in $(\mathcal{V},\mathcal{E})$. Also, observe that the generators are from the three families C, 12 and 19, and their parameters are given in terms of τ. As the SNC implies an ordering of the corresponding Kendall's tau, the definition of λ reflects this ordering, i.e., the index of a fork increases as the value of τ decreases. Finally, when nesting these ACs into each other, we obtain

$$C_{(\mathcal{V},\mathcal{E},\lambda)} = C_{\lambda(11)}(C_{\lambda(9)}(u_2,u_5,u_6),C_{\lambda(10)}(u_1,C_{\lambda(8)}(u_3,u_4,u_7))). \tag{6}$$

Figure 2 depicts our graphical representation of $C_{(\mathcal{V},\mathcal{E},\lambda)}$.

Using the toolbox, (5) can be implemented using four `cell` arrays.

```
lam8RightRight = {'12', tau2theta('12', 0.8)};
lam9Left       = {'19', tau2theta('19', 0.7)};
lam10Right     = {'12', tau2theta('12', 0.5)};
lam11Root      = {'C', tau2theta('C', 0.2)};
```

Each `cell` array contains the desired family (which can be any from Table 1) and the parameter value computed by the function `tau2theta`, which evaluates $\tau_{(a)}^{-1}(\tau)$; the inverse $\tau_{(a)}(\theta)$ can be evaluated with the function `theta2tau`. Analogues of these two functions are available in the R packages **copula** and **HAC** under the names `tau`, `iTau` and `theta2tau`, `tau2theta`, respectively.

To represent HAC models, the toolbox uses instances of the '`HACopula`' class. The following code shows how to instantiate such a representation (denoted `HACModel`) for $C_{(\mathcal{V},\mathcal{E},\lambda)}$ given by (6).

```
HACModel = HACopula({lam11Root, {lam9Left, 2, 5, 6}, ...
  {lam10Right, 1, {lam8RightRight, 3, 4, 7}}});
```

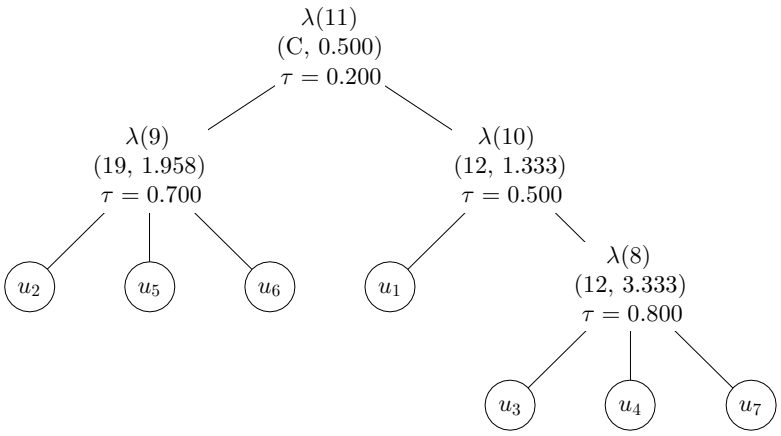


Figure 2: A 7-variate HAC including the three families C, 12 and 19 with the fork indices (the arguments of $\lambda(\cdot)$) ordered according to Kendall's tau (τ).

The only argument of the constructor of the '`HACopula`' class is a `cell` array representing the AC in the root of the tree, i.e., $C_{\lambda(11)}$ in our example, where the first cell contains its generator representation (`lam11Root`) and the remaining cells contain either a leaf index or another such an AC representation. In other words, each appearing `{}` defines one AC nested in the resulting HAC. In the R packages **copula** and **HAC**, such an object can be instantiated using `onacopula` and `hac`, respectively. The plot depicted in Figure 2 can be obtained by `plot(HACModel)`. The density of all bivariate marginal distributions corresponding to `HACModel` can be obtained by `plotbipdfs(HACModel)`, see Figure 3, where the range of levels of dependencies (quantified in terms of Kendall's tau below the main diagonal) as well as different asymmetries in the tails can be observed.

3.3. Computing probabilities involving HACs

Having constructed the HAC $C_{(\mathcal{V},\mathcal{E},\lambda)}$ represented by `HACModel`, one can let the toolbox compute several related quantities. For example, using the method `cdf`, one can evaluate the cumulative distribution function of $C_{(\mathcal{V},\mathcal{E},\lambda)}$ at an arbitrary point from $[0,1]^d$. Note that unless otherwise stated, a *method* in the following means a method of the '`HACopula`' class. The following code computes the value of $C_{(\mathcal{V},\mathcal{E},\lambda)}(0.5,\ldots,0.5)$.

```
cdf(HACModel, 0.5 * ones(1, HACModel.Dim))

ans =

    0.1855
```

Note that `HACModel.Dim` corresponds to the dimension of `HACModel`. In the R packages **copula** and **HAC**, an analogue of this function is available under the names `pCopula` and `pHAC`, respectively.

To compute the probability of a random vector distributed according $C_{(\mathcal{V},\mathcal{E},\lambda)}$ to fall in a hypercube $(l,u]$, where $l \in [0,1]^d$ and $u \in [0,1]^d$ denote the lower and upper corners of the

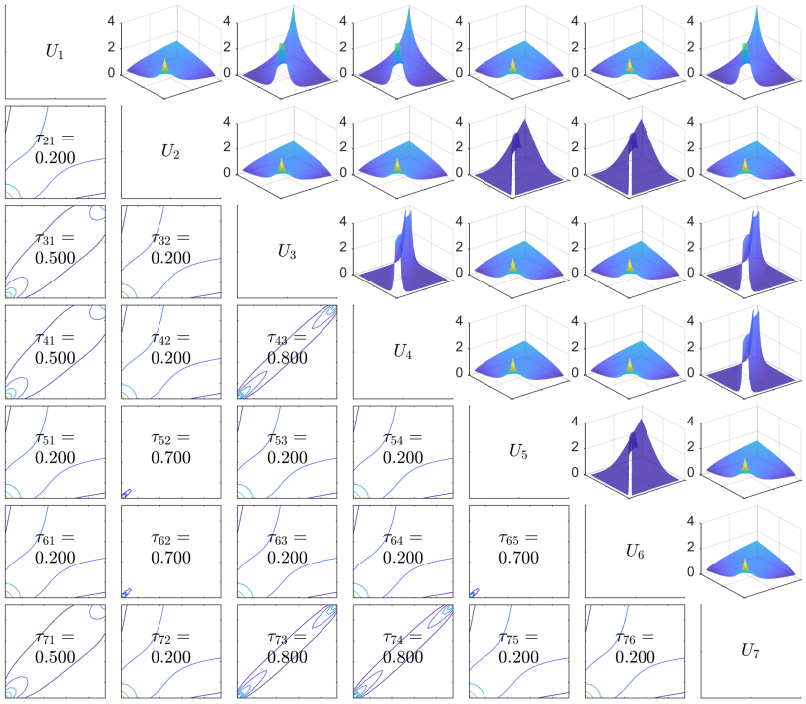


Figure 3: Densities of all bivariate marginal distributions of $C_{(\mathcal{V},\mathcal{E},\lambda)}$ (`HACModel`) are shown above the diagonal whereas below it are shown their contour plots and corresponding Kendall's taus.

hypercube in d dimensions, one can use the method `prob`. The following code computes this probability for the hypercube given by $l = (0.5, \ldots, 0.5)$ and $u = (0.9, \ldots, 0.9)$.

```
prob(HACModel, 0.5 * ones(1, HACModel.Dim), 0.9 * ones(1, HACModel.Dim))
```

```
ans =

    0.0437
```

In the R package **copula**, this function is available under the same name.

The toolbox also provides the method `cdfsurv`, which can be used to evaluate the survival copula of $C_{(\mathcal{V},\mathcal{E},\lambda)}$.

```
cdfsurv(HACModel, 0.5 * ones(1, HACModel.Dim))
```

```
ans =

    0.1748
```

Note that, in general, given a continuous random vector $\boldsymbol{X} = (X_1, \ldots, X_d)$ with *joint survival function* $\bar{F}(x_1, \ldots, x_d) = \mathbb{P}(\bigcap_{i=1}^{d} \{X_i > x_i\})$ and univariate *survival margins* $\bar{F}_i(x_i) = \mathbb{P}(X_i > x_i)$, $i \in \{1, \ldots, d\}$, the *survival copula* $\bar{C}$ associated with $\boldsymbol{X}$ is the copula satisfying

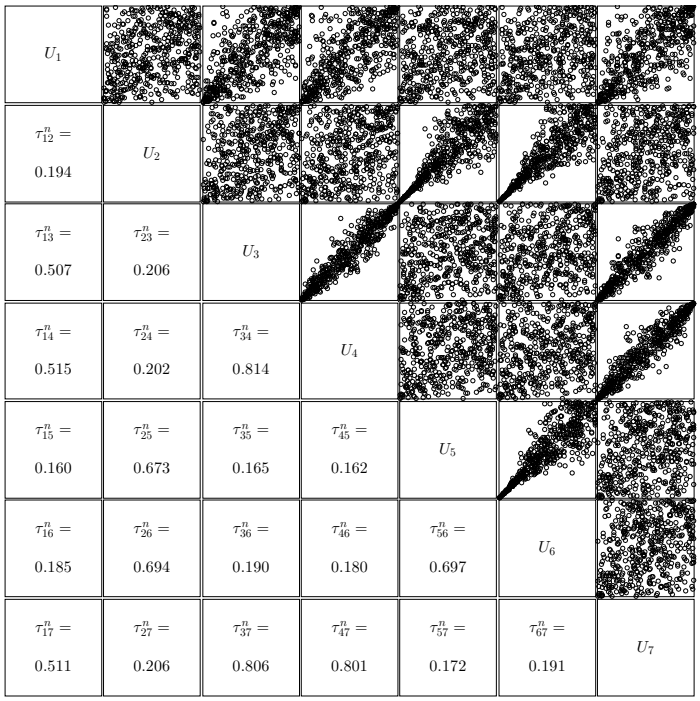


Figure 4: A sample of 500 observations from the $C_{(\mathcal{V},\mathcal{E},\lambda)}$ depicted in Figure 2.

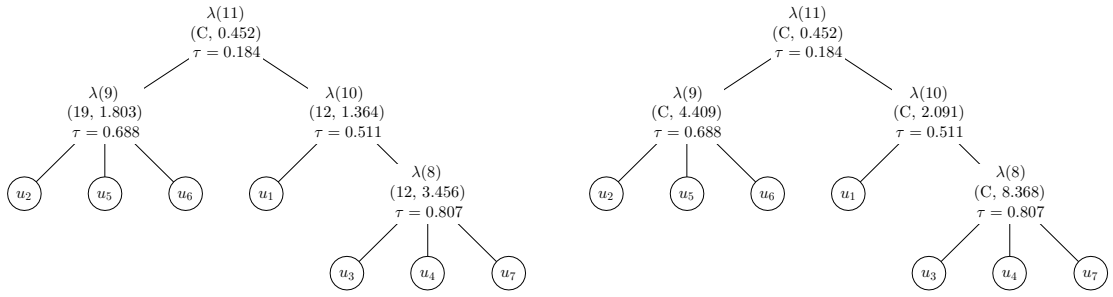
$\bar{F}(x_1,\ldots,x_d) = \bar{C}(\bar{F}_1(x_1),\ldots,\bar{F}_d(x_d))$ for all $(x_1,\ldots,x_d)^\top \in \mathbb{R}^d$. If $U \sim C$ then $1-U \sim \bar{C}$. For more details on survival copulas, see, e.g., Nelsen (2006, p. 31) or Durante and Sempi (2010). Finally note that in the R package **copula**, the survival copula can be evaluated using the function `rotCopula`.

3.4. Sampling HACs

To sample from HACs represented by instances of the '`HACopula`' class, the toolbox provides the method `rnd`, which implements the sampling strategies proposed in Hofert (2011, 2012). The following code generates a sample of 500 observations from `HACModel` and plots all its 2-dimensional projections. Note that the first two lines just set the seed in order for the result to be reproducible.

```
rng('default');
rng(1);
UKnown = rnd(HACModel, 500);
plotbimargins(UKnown);
```

As the function `rng` has not yet been implemented in Octave, the resulting sample `UKnown` produced by MATLAB is available in the file `highdimex_data.mat` in the folder `Demos`, which serves to obtain the same results for MATLAB and Octave. The last line of the code generates the plot shown in Figure 4, i.e., a sample analogously to Figure 3. Note that in the R packages **copula** and **HAC**, random sampling from HACs can be done by the functions `rnacopula` and `rHAC`, respectively.



(a) `plot(fitC1219)` (b) `plot(fitC)`

Figure 5: Two estimates computed for the data depicted in Figure 4.

3.5. Estimating HACs

Given i.i.d. observations $(X_{i1}, \ldots, X_{id})$, $i \in \{1, \ldots, n\}$ of a d-variate distribution function F given by (1) with known margins F_j, $j \in \{1, \ldots, d\}$, one can estimate C directly using $(U_{i1}, \ldots, U_{id})$, $i \in \{1, \ldots, n\}$, where $U_{ij} = F_j(X_{ij})$, $i \in \{1, \ldots, n\}$, $j \in \{1, \ldots, d\}$. In practice, the margins are typically unknown and must be estimated parametrically or non-parametrically. In the following, we base estimation of C on the *pseudo-observations*

$$U_{ij} = \frac{n}{n+1} \hat{F}_{n,j}(X_{ij}) = \frac{R_{ij}}{n+1},$$

where $\hat{F}_{n,j}$ denotes the *empirical distribution function* corresponding to the jth margin and R_{ij} denotes the *rank* of X_{ij} among $X_{1j}, \ldots, X_{nj}$.

Taking the sample `UKnown` generated in the previous section and assuming it represents the observations $(X_{i1}, \ldots, X_{id})$, $i \in \{1, \ldots, n\}$, mentioned above, i.e., assuming $F = C_{(\mathcal{V},\mathcal{E},\lambda)}$, the corresponding pseudo-observations `U` can be computed via

```
U = pobs(UKnown);
```

In the R package **copula**, this function is available under the same name.

Based on the pseudo-observations, the following code shows how to obtain two estimates of $C_{(\mathcal{V},\mathcal{E},\lambda)}$. The first one, `fitC1219`, is fitted under the assumption that the set of the underlying families are known, i.e., the family of each generator is chosen from the set of families {C, 12, 19}. The other one, denoted `fitC`, is fitted assuming that this set is unknown and the Clayton family for each generator were imposed arbitrarily.

```
fitC1219 = HACopulafit(U, getfamilies(HACModel));
fitC     = HACopulafit(U, {'C'});
```

Note that in the first line, the method `getfamilies(HACModel)` returns the set of the families involved in `HACModel`, i.e., the first line can alternatively be written as `fitC1219 = HACopulafit(U, {'C', '12', '19'});`. Figure 5 shows the plot of these estimates (the code for obtaining them is given in the captions). A detailed description of the function `HACopulafit` is presented in Section 4.3.

In the R package **HAC**, an analogue to `HACopulafit` is `estimate.copula`, which is, however, restricted to one pre-defined family of generators in the resulting HAC, i.e., only the estimate `fitC` could be obtained.

3.6. Goodness-of-fit testing for HACs

The **HACopula** toolbox provides several tools for measuring how well an estimate approximates the true copula (if it is known) or how well it fits the sample (a common case; the unknown true copula is typically substituted by the so-called *empirical copula*, i.e., the empirical distribution function of the pseudo-observations).

As a first approach, measuring a certain type of a distance between an estimate and the empirical copula evaluated at the given data is illustrated by the following code, where the goodness-of-fit test statistics denoted $S_n^{(E)}$ in Górecki *et al.* (2017b) (proposed in Genest, Rémillard, and Beaudoin (2009) under the notation S_n, see Equation 2 therein) is computed for the two previously considered estimates.

```
[gofdSnE(fitC1219, U)  gofdSnE(fitC, U)]

ans =

    0.0298    0.0524
```

As might be expected, the worse fit (i.e., the larger distance) is reached for the estimate with misspecified families. In the R package **copula**, an analogue to `gofdSnE` is available under the name `gofTstat`.

Based on the test statistic $S_n^{(E)}$, the toolbox also provides the method `computepvalue` that computes an approximate p value of the corresponding goodness-of-fit test via an adapted Monte Carlo method proposed in Genest *et al.* (2009), namely, a parametric bootstrap for S_n.

```
disp('Computing the p value for the first estimate...');
tic;
estimatorC1219 = @(U) HACopulafit(U, getfamilies(HACModel));
computepvalue(fitC1219, U, estimatorC1219, 100)
toc
disp('Computing the p value for the second estimate...');
tic;
estimatorC = @(U) HACopulafit(U, {'C'});
computepvalue(fitC, U, estimatorC, 100)
toc

Computing the p value for the first estimate...

ans =

    0.4700
```

```
Elapsed time is 52.488160 seconds.
Computing the p value for the second estimate...

ans =

    0.0500

Elapsed time is 29.103234 seconds.
```

Similarly to the previous example, one observes a lower (approximately 10 times) p value for the second estimate. As the computation of p values is intensive, a stopwatch timer is involved (`tic` and `toc`); this computation, as well as all the following ones, was done on an Intel Core 2.83 GHz processor. The value of the fourth argument of the method `computepvalue` (see the third line) provides the number of bootstrap replications, which is 100 here. Note that to compute such p values, an estimator of the underlying copula is needed. Therefore as the third argument, an anonymous function (e.g., `estimatorC1219`) implementing it is provided. Finally, note that due to the missing implementation of the previously mentioned function `rng` in Octave, the resulting p value might differ from the ones reported above.

At this place, the reader should be warned about the following. Even if the implementation of the parametric bootstrap method exactly follows its theoretical description proposed in Genest *et al.* (2009, Appendix A), the requirements on the properties of the involved rank-based estimator addressed in Genest and Rémillard (2008, see Equation 31 and below) necessary to guarantee that the parametric bootstrap method yields valid p values for $S_n^{(E)}$ have not been theoretically proven yet. More precisely, according to Genest and Rémillard (2008), the mentioned requirements are known to be satisfied only for 2-ACs. Any results obtained using `computepvalue` should thus be interpreted with care.

Another goodness-of-fit tool provided by the toolbox considers a distance between a copula estimate and a sample (pseudo-observations), where the latter can be substituted by the true copula if available. The distance is viewed in terms of matrices of pairwise coefficients like Kendall's tau or the upper- and lower-tail dependence coefficients. More precisely, the toolbox provides the method `distance`, which returns the quantity given by

$$\sqrt{\frac{1}{\binom{d}{2}} \sum_{i=1}^{d} \sum_{j=i+1}^{d} (\kappa_{ij}^{\square} - \kappa_{ij}^{\triangle})^2},$$

where $(\kappa_{ij}^{\square})$ and $(\kappa_{ij}^{\triangle})$ denote either 1) the matrices of pairwise Kendall's taus corresponding to a *copula estimate* and a *sample*, respectively, i.e., $\kappa = \tau$ (denoted by `kendall (HAC vs sample)` in the output below), or 2) the matrices of (tail) dependence coefficients corresponding to *two HACs* (denoted by `HAC vs HAC`), where $\kappa = \tau$ if the third argument of the method `distance` is `'kendall'`, $\kappa = \lambda_u$ if this argument is `'upper-tail'` or $\kappa = \lambda_l$ if it is `'lower-tail'`; for λ_l and λ_u, see Table 1.

```
K = kendallTauMatrix(U);

disp('kendall (HAC vs sample)');
```

```
[distance(fitC1219, K) distance(fitC, K)]

DISTANCE_TYPE = {'kendall', 'upper-tail', 'lower-tail'};
for i = 1:3
  disp([DISTANCE_TYPE{i} ' (HAC vs HAC)']);
  [distance(fitC1219, HACModel, DISTANCE_TYPE{i}) ...
    distance(fitC, HACModel, DISTANCE_TYPE{i})]
end

kendall (HAC vs sample)

ans =

    0.0129    0.0129

kendall (HAC vs HAC)

ans =

    0.0136    0.0136

upper-tail (HAC vs HAC)

ans =

    0.0081    0.3145

lower-tail (HAC vs HAC)

ans =

    0.0260    0.0868
```

Note that the first line just computes the matrix of pairwise Kendall's taus for `U`. Observe that all resulting distances based on Kendall's tau are the same. This follows from the used estimation method, which computes the parameter values just from the matrix of pairwise Kendall's taus (which are the same for both estimates) using the 1-to-1 relation between Kendall's tau and AC parameters mentioned in Section 3.2. For more details on the estimation method, see Section 4.3 and Górecki *et al.* (2017b). Also observe the relatively large value for the upper-tail distance for `fitC`, which is produced mainly due to the fact that this estimate based on the Clayton family is unable to model non-zero upper-tail dependence; see λ_u for a = C in Table 1.

The matrix of pairwise Kendall's taus related to each '`HACopula`' object can be displayed using the method `getdependencematrix`.

```
getdependencematrix(HACModel, 'kendall')
getdependencematrix(fitC1219, 'kendall')
```

```
ans =

    1.0000    0.2000    0.5000    0.5000    0.2000    0.2000    0.5000
    0.2000    1.0000    0.2000    0.2000    0.7000    0.7000    0.2000
    0.5000    0.2000    1.0000    0.8000    0.2000    0.2000    0.8000
    0.5000    0.2000    0.8000    1.0000    0.2000    0.2000    0.8000
    0.2000    0.7000    0.2000    0.2000    1.0000    0.7000    0.2000
    0.2000    0.7000    0.2000    0.2000    0.7000    1.0000    0.2000
    0.5000    0.2000    0.8000    0.8000    0.2000    0.2000    1.0000

ans =

    1.0000    0.1844    0.5111    0.5111    0.1844    0.1844    0.5111
    0.1844    1.0000    0.1844    0.1844    0.6880    0.6880    0.1844
    0.5111    0.1844    1.0000    0.8071    0.1844    0.1844    0.8071
    0.5111    0.1844    0.8071    1.0000    0.1844    0.1844    0.8071
    0.1844    0.6880    0.1844    0.1844    1.0000    0.6880    0.1844
    0.1844    0.6880    0.1844    0.1844    0.6880    1.0000    0.1844
    0.5111    0.1844    0.8071    0.8071    0.1844    0.1844    1.0000
```

The upper- and lower-tail dependence coefficients for a '`HACopula`' object can be displayed as illustrated below.

```
getdependencematrix(HACModel, 'tails')
getdependencematrix(fitC, 'tails')

ans =

    1.0000         0    0.3182    0.3182         0         0    0.3182
    0.2500    1.0000         0         0         0         0         0
    0.5946    0.2500    1.0000    0.7689         0         0    0.7689
    0.5946    0.2500    0.8123    1.0000         0         0    0.7689
    0.2500    1.0000    0.2500    0.2500    1.0000         0         0
    0.2500    1.0000    0.2500    0.2500    1.0000    1.0000         0
    0.5946    0.2500    0.8123    0.8123    0.2500    0.2500    1.0000

ans =

    1.0000         0         0         0         0         0         0
    0.2159    1.0000         0         0         0         0         0
    0.7179    0.2159    1.0000         0         0         0         0
    0.7179    0.2159    0.9205    1.0000         0         0         0
    0.2159    0.8545    0.2159    0.2159    1.0000         0         0
    0.2159    0.8545    0.2159    0.2159    0.8545    1.0000         0
    0.7179    0.2159    0.9205    0.9205    0.2159    0.2159    1.0000
```

Each tail dependence matrix is represented in a "condensed" form, in which the values above the main diagonal correspond to the upper tail-dependence coefficients, whereas the values

below the main diagonal to the lower ones. With the R package **copula**, these matrices and distances can be computed using the functions `tau`, `lambdaL` and `lambdaU`. With the R package **HAC**, the matrix of pairwise Kendall's taus is available through the function `par.pairs`.

3.7. Auxiliaries

Another handy tool provided by **HACopula** allows one to compare HAC structures, where the *structure* of a HAC is the set consisting of the descendant leaves of all forks. For example, the structure corresponding to `HACModel`, and also to the three estimates considered above, is $\{\{3,4,7\},\{2,5,6\},\{1,3,4,7\},\{1,\ldots,7\}\}$; one can observe that the inner sets correspond to the forks 8, 9, 10 and 11, respectively. This tool, implemented by the method `comparestructures`, returns 1 if the structure is the same for both inputs, and 0 otherwise.

```
comparestructures(HACModel, fitC1219)

ans =

  logical

   1
```

For a more refined insight into how much two HAC structures match, one can require a second output from `comparestructures`, as can be seen in the following code, where the structure of `HACModel` is compared to the (trivial) structure of a 7-AC.

```
[isSameStruc, ratioStruc] = comparestructures(HACModel, ...
  HACopula({{'C', 0.5}, 1, 2, 3, 4, 5, 6, 7}))

isSameStruc =

  logical

   0


ratioStruc =

    0.0571
```

The level of match (`ratioStruc` $\in [0,1]$) is based on the trivariate decomposition of HAC tree structures introduced in Segers and Uyttendaele (2014). Its computation is based on the idea from Uyttendaele (2018), where two HAC structures are first decomposed to two sets of trivariate structures and each pair of corresponding trivariate structures is then checked to match. The level of match is then the ratio of the matching trivariate structures to all trivariate structures. In our example, `ratioStruc = 2/nchoosek(HACModel.Dim, 3)`, i.e., two trivariate structures in `HACModel`, namely the ones with leaf indices (2, 5, 6) and (3, 4, 7), have the same structure (corresponding to a 3-AC) as the trivariate ones corresponding

to the 7-AC model. The remaining trivariate structures in `HACModel` always, in contrast to the corresponding ones in the 7-AC model, involve some kind of hierarchy, cf. Figure 2.

If the structures are the same for two HAC models, **HACopula** also provides a method that computes how much the families involved in these HACs match to each other.

```
[isSameFams, ratioFams] = comparefamilies(fitC1219, fitC)
```

```
isSameFams =

  logical

   0


ratioFams =

    0.2500
```

The first output (`isSameFams`) returns 1 if the family of a generator in the first argument (`fitC1219`) is the same as the family of the corresponding generator in the second argument (`fitC`) for all the involved forks, 0 otherwise; observe from Figures 5(a) and 5(b) that the family is the same only for $\lambda(11)$. The second output (`ratioFams`) returns the ratio of the forks for which the families match to the number of all forks. Note that the described functionality considering the families and structures is not available from other software packages.

To generate an analytical form of `HACModel` (or of some of its part) exportable to LaTeX, the toolbox provides the method `tolatex`.

```
tolatex(HACModel.Child{2}, 'cdf')
```

The result is the following formula.

$$\frac{1}{\left(\left(\frac{1}{u_1}-1\right)^{\theta_{10}}+\left(\left(\left(\frac{1}{u_3}-1\right)^{\theta_8}+\left(\frac{1}{u_4}-1\right)^{\theta_8}+\left(\frac{1}{u_7}-1\right)^{\theta_8}\right)^{1/\theta_8}\right)^{\theta_{10}}\right)^{1/\theta_{10}}+1}$$

Substituting `'cdf'` by `'pdf'` enables one to access HAC densities, however, one should be aware of the fact that their computation for $d > 5$ is time consuming. Also note that such functionality is not available from other software packages.

As follows from Proposition 2.1 in Okhrin *et al.* (2013), knowing all bivariate margins of a HAC, one can uniquely determine it including its tree structure, see the example below Remark 2.3 therein. It is thus convenient to represent a HAC in terms of its bivariate margins, as is already done in this work, e.g., in Figures 3 or 4. To access easily any bivariate margin of a '`HACopula`' object, one can use its method `getbimargin`, which returns the desired margin represented again as a '`HACopula`' object. One can then use it for further computations, e.g., for accessing its density, as shown in the following example; compare with Figure 3.

```
biMargin = getbimargin(HACModel, 1, 6);
ACpdf(biMargin.Family, biMargin.Parameter, 0.9, 0.5)

ans =

    1.0691
```

4. Elaborating on the quick example

This section provides under-the-hood details of the features presented in the previous section. All the examples from this section can be reproduced using the file `elaboratedex.m` in the folder `Demos`.

4.1. Constructing HACs

As hinted at Section 3, the toolbox is built around the '`HACopula`' class, of which instances serve as HAC models and of which methods provide desired functionality. Its constructor will now be addressed in more detail. To this end, the `HACModel` instantiation described in Section 3.2 will be used again, however, without the semicolon at the end (of the sixth line).

```
lam8RightRight = {'12', tau2theta('12', 0.8)};
lam9Left       = {'19', tau2theta('19', 0.7)};
lam10Right     = {'12', tau2theta('12', 0.5)};
lam11Root      = {'C', tau2theta('C', 0.2)};
HACModel = HACopula({lam11Root, {lam9Left, 2, 5, 6}, ...
  {lam10Right, 1, {lam8RightRight, 3, 4, 7}}})

HACModel =

  HACopula with properties:

        Family: 'C'
     Parameter: 0.5000
           Tau: 0.2000
   TauOrdering: 11
         Level: 1
        Leaves: [2 5 6 1 3 4 7]
           Dim: 7
         Child: {[1×1 HACopula]  [1×1 HACopula]}
        Parent: []
          Root: [1×1 HACopula]
         Forks: {1×4 cell}
```

An instance of the '`HACopula`' class defines an AC at the first level of recursion (indicated by the value of the property `Level`), of which arguments are represented by the `cell` array `Child` containing in each cell either a '`HACopula`' instance (two '`HACopula`' instances in the

example above) or an integer representing a variable of the HAC; for the latter, see the output of the following code.

```
HACModel.Child{2}

ans =

  HACopula with properties:

         Family: '12'
      Parameter: 1.3333
            Tau: 0.5000
    TauOrdering: 10
          Level: 2
         Leaves: [1 3 4 7]
            Dim: 4
          Child: {[1]  [1×1 HACopula]}
         Parent: [1×1 HACopula]
           Root: [1×1 HACopula]
          Forks: {[1×1 HACopula]  [1×1 HACopula]}
```

Given such a HAC representation, the properties `Leaves`, `Parent`, `Root` and `Forks` contain, respectively, its descendant nodes that are leaves, its parent (an empty matrix, if it is the root), the root and the descendant nodes that are forks including itself. The main reason for maintaining these properties is to increase the speed of the calculations regarding the recursive nature of '`HACopula`' instances. Note that the latter three properties contain '`HACopula`' instances. The property `TauOrdering` plays the role of an identifier of a fork, i.e, each fork has its unique value of this property, which is ordered according to the corresponding Kendall's tau stored in the property `Tau` (in case of a tie, forks are ordered lexicographically according to their descendant leaves). This ordering is assigned by the method `addtauordering` to all its forks anytime a new instance of '`HACopula`' is created. Note that the values of `TauOrdering` serve as labels of the nodes in a HAC tree structure that correspond to the generators in the HAC. Hence, as the nodes in this tree that are leaves (which correspond to the variables in the HAC) are already labeled $1, \ldots, d$, the values of `TauOrdering` for the forks are from $\{d+1, \ldots, d+f\}$, where f is the number of forks. The value of `TauOrdering` assigned to the root is $d+f$, and for the remaining forks this ordering is decreasing along with the Kendall's tau of these forks being increasing.

Also note that the '`HACopula`' class is inherited from the abstract class '`handle`', which implies that, if a function modifies a '`HACopula`' object passed as an input argument, the modification affects the original input object; in all such functions provided by the toolbox, this is noted at the help part of the corresponding function. To create a copy of a '`HACopula`' instance, one can use the `copy` method provided by the '`HACopula`' class.

Finally note that, as the toolbox works only with the HACs under the SNC, necessary SNC checks for the parameters are implemented by the method `checksnc` (for the parametric forms of the SNC, see Tables 1 and 2), which is also called anytime a new instance of '`HACopula`' is created, together with the method `checkleaves`, which controls if the leaf indices passed in the nested cell array to the constructor constitute the set $\{1, \ldots, d\}$.

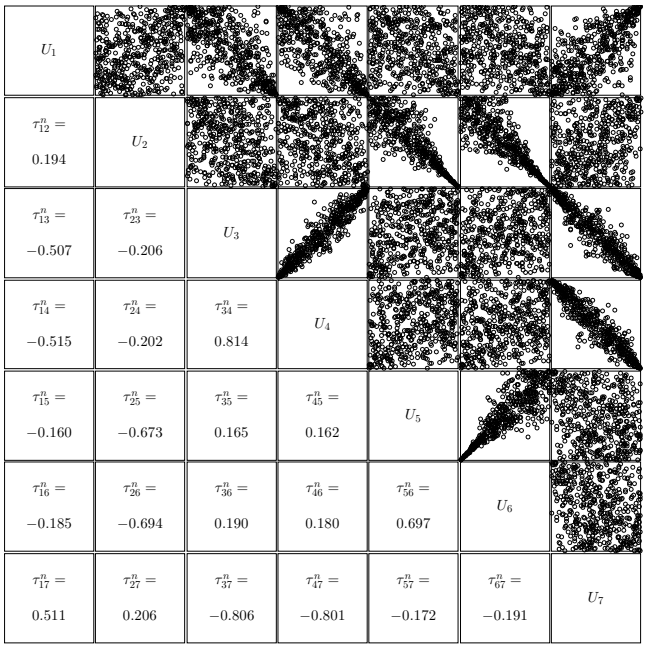


Figure 6: A sample of 500 observations from the HAC $C_{(\mathcal{V},\mathcal{E},\lambda)}$ depicted in Figure 2 with the flipped variables 1, 2 and 7.

4.2. Sampling HACs

On the one hand, the SNC guarantees that a proper copula results, on the other hand, it implies that such a HAC is unable to model negative dependence (in the sense of concordance), e.g., $\tau \geq 0$ for all pairs of variables from a random vector following such a HAC. Although this limitation is typically satisfied by financial return series (possibly after adjusting their sign), it might be too restrictive in certain applications. At the best of our knowledge, the only attempt to at least partially overcome this limitation has been proposed in Górecki *et al.* (2016, Algorithm 4). Although the proposed approach does not solve the problem in general, it might be helpful in several cases, one of which is illustrated below.

Sample again the data as in Section 3.4, but then, impose some negative dependence by the simple argument-reflecting transformation given in the last line.

```
rng('default');
rng(1); % set the seed

UKnown = rnd(HACModel, 500);
UKnown(:, [1 2 7]) = 1 - UKnown(:, [1 2 7]);
```

The transformation (also called *reflection* at 1/2 in each (or only some) component(s)) just flips the data in the columns corresponding to the variables indexed 1, 2 and 7 . The resulting sample obtained by `plotbimargins(UKnown)` is depicted in Figure 6.

Now assume that it is unknown how these data were produced. To fit a reasonable HAC model to these data, one has to somehow reduce the observed negative pairwise correlations,

e.g., by flipping. To detect which of the variables should be flipped, one can use the function `findvars2flip` implementing Algorithm 4 from Górecki *et al.* (2016).

```
KNeg = kendallTauMatrix(UKnown);
toFlip = findvars2flip(KNeg)

toFlip =

     2     7     1
```

With this result, one can flip the data using `UKnown(:, toFlip) = 1 - UKnown(:, toFlip)`, which turn them into the purely positively correlated data depicted in Figure 4, which are more suited to the modeling under the SNC. Note that this functionality is not available in other considered software packages. Also note that this approach does not always provide a solution, e.g., having 3-dimensional (e.g., real-world) data such that $\tau_{12} = \tau_{23} = 0.2$ and $\tau_{13} = -0.5$, no flipping in the sense described above leads to non-negative correlations for all three pairs from (U_1, U_2, U_3). A solution for such cases is unknown.

4.3. Estimating HACs

Section 3.5 has demonstrated, for the sake of simplicity, only one estimator provided by the function `HACopulafit`. However, this function provides a much wider variety of estimators including, e.g., the 192 estimators considered in Górecki *et al.* (2017b, Section 7), none of which is available from the other considered software packages. To get a more complete picture of the possibilities offered by `HACopulafit`, see the following code, which shows all its default settings.

```
U = pobs(UKnown);
families = getfamilies(HACModel);
[fit, fitLog] = HACopulafit(U, families, 'HACEstimator', 'pairwise',...
  'ThetaEstimator', 'invtau', 'ThetaEstimator2', 'invtau', 'g_1', ...
  'average', 'g_2', @(t)mean(t), 'GOF', 'R', 'PreCollapse', true, ...
  'Reestimator', 'Ktauavg', 'nForks', 'unknown', 'Attitude', ...
  'optimistic', 'CheckData', 'on');
```

Note that in this default setting, `HACopulafit` implements the estimator denoted *Coll=pre & Re-est=KTauAvg & Alg=PT-avg & Sn=R & Att=opt #Forks=unknown* in Górecki *et al.* (2017b, Section 7), which is suggested as a good default in the reported simulation study. For the details explaining the theoretical concept behind these input arguments, see Górecki *et al.* (2017b) together with Table 3 here. This links the notation from Górecki *et al.* (2017b) with the names of the arguments used in the `HACopulafit` function. How to use the estimators available in `HACopulafit` is described in the help comments provided with its implementation. Note that apart from the parameters listed in the example above, `HACopulafit` currently also accepts five other parameters – `'KendallMatrix'`, `'Emp2copulas'`, `'PreCollapsedHAC'`, `'CollapsedArray'` and `'MinDistanceArray'`, which serve for delivering certain pre-computed quantities and their description is addressed soon.

An important part of the estimation process implemented by `HACopulafit` concerns so-called *collapsing* of a HAC structure, which turns binary HAC structures (binary trees often resulting

Features from Górecki *et al.* (2017b)	Corresponding `HACopulafit` settings
Alg = PT	`'HACEstimator' = 'pairwise'` and `'ThetaEstimator' = 'invtau'`
Alg = DM	`'HACEstimator' = 'diagonal'` and `'ThetaEstimator' = 'mle'`
g = avg	`'g_2' = @(t)mean(t)`
g = max	`'g_2' = @(t)max(t)`
Sn = E	`'GOF' = 'E'`
Sn = K	`'GOF' = 'K'`
Sn = R	`'GOF' = 'R'`
Coll = pre	`'PreCollapse' = true`
Coll = post	`'PreCollapse' = false`
Re-est = KTauAvg	`'Reestimator' = 'Ktauavg'`
Re-est = TauMin	`'Reestimator' = 'taumin'`
Attitude = opt	`'Attitude' = 'optimistic'`
Attitude = pes	`'Attitude' = 'pessimistic'`

Table 3: The left-hand column shows the features of the estimators considered in Górecki *et al.* (2017b, Section 7). The right-hand column shows the corresponding input arguments settings of the function `HACopulafit`. The ones not shown in the table, i.e., `'g_1'` and `'nForks'`, are set by default to `'average'` and to `'unknown'`, respectively.

from estimation processes, e.g., see Górecki and Holeňa (2014, Algorithm 3) for a simple example) to non-binary ones, allowing to access all possible HAC structures. In the simulation study in Górecki *et al.* (2017b), the collapsing approach denoted *Coll = pre & Re-est = KTauAvg* outperformed the remaining collapsing approaches considered, which is why it was chosen as default. The following code highlights this approach.

```
fit2Bin = HACopulafit(U, {'?'}, 'PreCollapse', false);
K = kendallTauMatrix(U);
[colHACArray, minDistArray] = collapse(fit2Bin, 'invtau', ...
  K, U, @(t) mean(t), 'optimistic', 'Ktauavg', false)

colHACArray =

  Columns 1 through 4

    [1x1 HACopula]    [1x1 HACopula]    [1x1 HACopula]    [1x1 HACopula]

  Columns 5 through 6

    [1x1 HACopula]    [1x1 HACopula]

minDistArray =

         0    0.0109    0.0137    0.2960    0.4747    0.3453
```

The first line computes a binary structured HAC estimate without any assumption on the underlying families, which is imposed by using an arbitrary family denoted `'?'`; see Górecki,

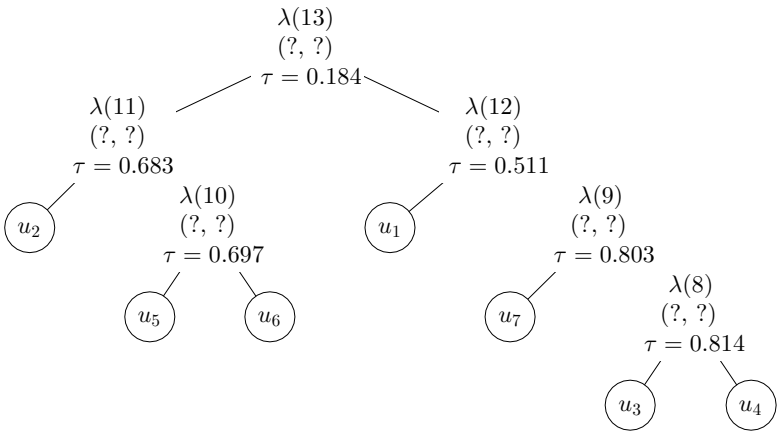


Figure 7: A binary structured HAC estimate obtained for the data depicted in Figure 4, where no assumption on the underlying families has been made, indicated by the arbitrary family denoted `'?'` used in the generators.

Hofert, and Holeňa (2017a) for the theory behind this approach. The resulting estimate is depicted in Figure 7 (note that this plot can be obtained by `plot(fit2Bin)`). In the third line, a sequence of HACs with decreasing number of forks (`colHACArray`) is generated by the method `collapse` such that two parent-child forks with closest values of Kendall's tau are collapsed into one repeatedly until only one fork remains, i.e., the HAC stored in the last cell of `colHACArray` is actually an AC, which can be easily checked using `plot`. Also note that these differences between Kendall's tau of the collapsed parent-child forks are stored in `minDistArray`. The user is then free to choose any collapsed HAC from the generated sequence.

To help the user with this choice, the approach proposed in Górecki *et al.* (2017b, Section 6.1) is implemented by the function `findjump`, which estimates the number of forks in the underlying HAC by detecting the first (relatively) substantial jump in the distances stored in `minDistArray`. For our example, these distances are depicted in Figure 8. One can observe the first substantial jump between the third and the fourth value, which is also detected by the function.

```
iJump = findjump(minDistArray)
```

```
iJump =

     3
```

One can then automatically choose the collapsed HAC according to this output using the following code.

```
fit2UnknownFams = colHACArray{iJump};
```

Its plot is depicted on the right-hand side of Figure 8. If the input parameter `'PreCollapse'` is set `true`, the function `HACopulafit` uses this approach to estimate the number of forks

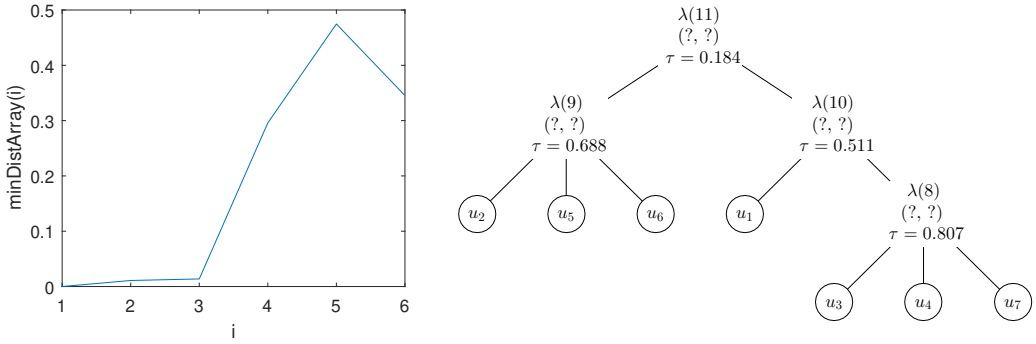


Figure 8: The left-hand side shows the values of `minDistArray`. The right-hand side shows the collapsed HAC `colHACArray{3}`.

as default, which is indicated by setting the parameter `'nForks'` to `'unknown'`; if the user prefers some particular number of forks in the resulting HAC, this number of forks can be enforced by passing it to `HACopulafit` instead of `'unknown'`.

Finally, the families and the parameters can be estimated by supplying the collapsed structure as optional input argument (structure estimation is avoided in such a case).

```
fit2 = HACopulafit(U, families, 'PreCollapsedHAC', fit2UnknownFams);
```

Using `plot(fit2)`, the reader can check that it is the one depicted in Figure 5(a).

Sometimes, it is more convenient to let `HACopulafit` choose the collapsed structure directly from the cell array `colHACArray` (i.e., the function `findjump` is used directly in `HACopulafit`), which can be done as follows.

```
fitCollDir = HACopulafit(U, families, 'CollapsedArray', colHACArray, ...
  'MinDistanceArray', minDistArray);
```

Using `plot(fitCollDir)`, the reader can again see the HAC representation from Figure 5(a).

It is important to note that HACs delivered to `HACopulafit` either by `'PreCollapsedHAC'` or `'CollapsedArray'` are not limited to the family `'?'`, but can contain any families supported by **HACopula**. This also allows one to *re-estimate* the families and parameters (but keeping the structure) of an existing 'HACopula' object. For example, if one wants to re-estimate `fitCollDir` in order to all involved forks be from the Clayton family, one can use the following code.

```
fitClayton = HACopulafit(U, {'C'}, 'PreCollapsedHAC', fitCollDir);
```

The resulting estimate has already been depicted in Figure 5(b), as can be checked by `plot(fitClayton)`.

Once a larger simulation study is to be conducted, optimization comes into play. For this purpose, the function `HACopulafit` accepts several pre-computed quantities. Apart from the parameters `'PreCollapsedHAC'`, `'CollapsedArray'` and `'MinDistanceArray'` addressed above, `'KendallMatrix'`, if supplied, avoids computation of the matrix of Kendall taus for a

given data sample. This matrix can be computed by `kendallTauMatrix(U)` for a data sample `U`, which is useful when several estimation procedures are performed on the same data. Another such a parameter is `'Emp2copulas'` computed by `computeallemp2copulas`, which serves for delivering all bivariate empirical copulas. Supplying this input is useful when several estimation procedures involving goodness-of-fit test statistics are performed on the same data (i.e., when more than one family is assumed for the generators).

Also note that each time the function `HACopulafit` is executed, the input data sample `U` is tested for uniformity of its univariate margins on [0, 1] by the function `iscopuladata`, which for each margin performs the two-sample Kolmogorov-Smirnov test, where the sample is compared to the standard uniform distribution. To switch off this test, set the parameter `CheckData` to `'off'`.

Finally, as the estimation process might become complex, particularly when estimating a HAC involving different families, its details for a particular input are written in the second output argument of `HACopulafit` (denoted `fitLog` in our example). This is particularly useful for explaining how the algorithm came to the resulting estimate, see the following content of the variable `fitLog`. Note that for the sake of brevity, several parts of `fitLog` are omitted.

```
fitLog =

    '***** HACopulafit: Start... *****
     ***** Pre-collapsing: Start... *****
     Estimating the binary structure (no assumptions on the families)...
     ***** HACopulafit: Start... *****
     (tau^n_{ij}):
        .   0.1940 0.5069 0.5154 0.1600 0.1848 0.5112
        .      .   0.2057 0.2024 0.6729 0.6938 0.2057
        .      .      .   0.8144 0.1648 0.1897 0.8063
        .      .      .      .   0.1624 0.1796 0.8006
        .      .      .      .      .   0.6971 0.1723
        .      .      .      .      .      .   0.1911
        .      .      .      .      .      .      .
     ------------------------------------------------------------------
     k = 1 *** Estimating \lambda(8):
     I = [1  2  3  4  5  6  7]
     g_1(K_{\downarrow(3), \downarrow(4)}) = 0.81438
     Join leaves: \downarrow(8) = [3  4]
     ------------------------------------------------------------------
     k = 2 *** Estimating \lambda(9):
     I = [1  2  5  6  7  8]
     g_1(K_{\downarrow(7), \downarrow(8)}) = 0.80345
     Join leaves: \downarrow(9) = [3  4  7]
     ------------------------------------------------------------------
```

(omitting the steps for $k = 3, \ldots, 6$)

```
     ------------------------------------------------------------------
     ***** HACopulafit: Stop. *****
```

```
Generating a sequence of 6 (collapsed) structures from the binary one.
Taking the collapsed structure with 4 forks,
following the estimated number of forks given by findjump, i.e.,
instead of the binary structure given by
\downarrow(8) = {3  4}
\downarrow(9) = {3  4  7}
\downarrow(10) = {5  6}
\downarrow(11) = {2  5  6}
\downarrow(12) = {1  3  4  7}
\downarrow(13) = {1  2  3  4  5  6  7}
taking the non-binary one given by
\downarrow(8) = {3  4  7}
\downarrow(9) = {2  5  6}
\downarrow(10) = {1  3  4  7}
\downarrow(11) = {1  2  3  4  5  6  7}
Note that \downarrow(i) = {i} for i = 1, ..., 7.
***** Pre-collapsing: Done. *****

Estimating the families and parameters...
-------------------------------------------------------------------
k = 1 *** Estimating \lambda(8):
I = [1  2  3  4  5  6  7]
g_1(K_{\downarrow(3), \downarrow(4), \downarrow(7)}) = 0.80709
Admissible families + ranges:
{(C, [eps(0), Inf)), (12, [1, Inf)), (19, [eps(0), Inf))}
Theta estimation:
family = C   theta = 8.3676
family = 12   theta = 3.4559
family = 19   theta = 4.2663
Family estimation:
S_n^{g_2} for the families (C, 12, 19) is (0.6070, 0.0497, 2.2102)
Best-fitting \psi^(family, theta) = \psi^(12, 3.4559)
Join leaves: \downarrow(8) = [3  4  7]
-------------------------------------------------------------------
k = 2 *** Estimating \lambda(9):
I = [1  2  5  6  8]
g_1(K_{\downarrow(2), \downarrow(5), \downarrow(6)}) = 0.68796
Admissible families + ranges:
{(C, [eps(0), Inf)), (12, [1, Inf)), (19, [eps(0), Inf))}
Theta estimation:
family = C   theta = 4.4094
family = 12   theta = 2.1365
family = 19   theta = 1.8031
Family estimation:
S_n^{g_2} for the families (C, 12, 19) is (0.6162, 1.2399, 0.0921)
Best-fitting \psi^(family, theta) = \psi^(19, 1.8031)
```

```
Join leaves: \downarrow(9) = [2  5  6]
-----------------------------------------------------------------
```

(omitting the steps for $k = 3, 4$)

```
-----------------------------------------------------------------
***** HACopulafit: Stop. *****'
```

The notation used in the log above corresponds to the notation from Górecki *et al.* (2017b, Algorithms 1 and 3). Note that as the estimation procedure described by `fitLog` involves pre-collapsing, `HACopulafit` at the beginning calls itself to get a binary structured estimate (assuming the arbitrary family `'?'` for all generators), which is indicated by repeating the log `***** HACopulafit: Start... *****`.

At this place, it is important to note, that, on the one hand, the HAC estimators implemented by `HACopulafit` have been proven to work by means of simulation, see Górecki *et al.* (2017b, Section 7). On the other hand, the statistical (large sample) properties of these estimators, particularly how the collapsing step affects the asymptotic distribution of the estimator (if it exists), are unknown at the moment. One thus should have this in mind when interpreting results obtained with `HACopulafit`.

5. Speed up for high dimensions

In a lot of applications, copula modeling has to be carried out in high dimensions, e.g., see Hofert *et al.* (2013) for a motivation in the area of finance. One purpose of this section is to demonstrate that such high-dimensional modeling can be accomplished with the **HACopula** toolbox. Also, it is shown how to use **HACopula** to speed up the computation of the matrix of pairwise Kendall's taus offered by the standard installation of MATLAB and Octave. This matrix is frequently required, e.g., for estimation of HACs, and, as is illustrated below, its computation becomes demanding especially in high dimensions. As the speed-up requires an installation of an extra compiler compatible with MATLAB or Octave, we have kept this more advanced part of **HACopula** to this penultimate section.

For the purposes mentioned above, an example in which a 100-variate HAC is constructed, sampled, estimated, goodness-of-fit tested and evaluated is provided. Additionally to the previous examples, computation times for all procedures are shown. Also note that in such high dimensions, estimation of a HAC including its structure has not yet been reported in the literature.

To simplify the construction of high-dimensional HAC models, the toolbox provides an auxiliary function `getfullmodel`, which builds a certain type of HAC models that are easily scalable to high dimensions. The following example can be reproduced using the file `highdimex.m` in the folder `Demos`.

In the code below, the 100-variate HAC depicted in Figure 9 is constructed and a sample of 2000 observations from it is generated (to get reproducibility with Octave, load `U` from the file `highdimex_data.mat` in the folder `Demos`).

```
HACModel = getfullmodel(11, 10, 'C', 0.1, 0.08);
rng('default'); rng(1);
tic; disp('Sampling...'); U = pobs(rnd(HACModel, 2000)); toc
```

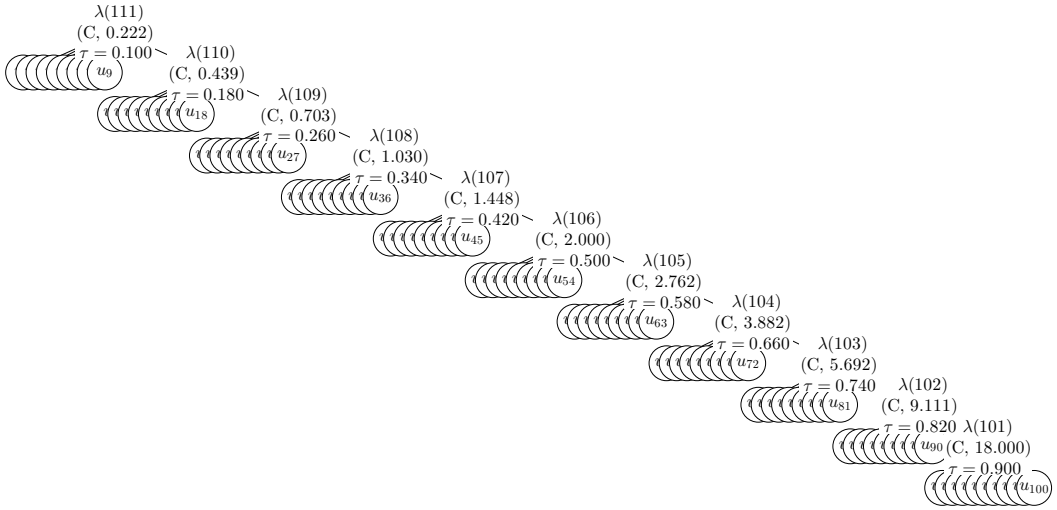


Figure 9: A 100-variate HAC with `11` nesting levels, where each level contains one `10`-variate AC from the Clayton family (`'C'`), with the parameter at the root corresponding to Kendall's tau = `0.1` and the differences between the parameters of a parent and its child corresponding to Kendall's tau = `0.08` (constructed via `getfullmodel(11, 10, 'C', 0.1, 0.08)`).

```
Sampling...
Elapsed time is 3.024096 seconds.
```

Before proceeding to HAC estimation, it is convenient to first compute the matrix of pairwise Kendall's taus for `U`. MATLAB (**Statistics and Machine Learning Toolbox**) as well as Octave provide an implementation computing a Kendall's tau estimate for a pair of vectors of n observations in $\mathcal{O}(n^2)$, which makes the computation of the matrix of pairwise Kendall's taus (overall effort is thus $\mathcal{O}(d^2n^2)$ in this case) in our example by far the most demanding part (as illustrated below). However, there exist faster algorithms that can compute a Kendall's tau estimate in $\mathcal{O}(n\log(n))$, see Knight (1966); Abrevaya (1999); Christensen (2005). One of them implemented in C++, which has been taken from the **VineCopulaMATLAB** toolbox (Kurz 2016), see its file `SDTau.cpp`, is also provided by **HACopula**. As it is in C++ (here in the file `fastKendallTau.cpp`), it has to be compiled before its first use. Note that to compile C++ code, both MATLAB and Octave require some C++ compiler, for example, see https://www.mathworks.com/support/compilers.html which contains information about compatible compilers for different releases and operating systems. The commands for the compilation together with computation times for the standard and for the fast implementation are shown below.

```
tic; disp('Slow (O(n^2)) computation of Kendall''s matrix...');
KSlow = kendallTauMatrix(U);
toc

% compile fastKendallTau.cpp to a MATLAB executable (.mex) file
if isoctave
  mkoctfile --mex fastKendallTau.cpp
else % MATLAB
```

```
  mex fastKendallTau.cpp
end

tic; disp('Fast (O(n*log(n))) computation of Kendall''s matrix...');
K = kendallTauMatrix(U);
toc

disp('Are the results equal?');
sum(sum(K == KSlow)) == numel(K)

Slow (O(n^2)) computation of Kendall's matrix...
Elapsed time is 118.194155 seconds.

Building with 'Microsoft Visual C++ 2017 Professional'.
MEX completed successfully.

Fast (O(n*log(n))) computation of Kendall's matrix...
Elapsed time is 3.962349 seconds.

Are the results equal?

ans =

  logical

   1
```

I.e., with the fast computation, a speed-up of roughly 30 times can be obtained in this example.

Now, two estimates, one based on the re-estimation approach proposed in Górecki *et al.* (2017b) (`fit1Avg`), the other one on the re-estimation approach proposed in Uyttendaele (2018) (`fit2Min`), are computed based on the observations in `U`. The matrix of pairwise Kendall's taus is provided as the last argument.

```
tic; disp('Estimating (1)...');
fit1Avg = HACopulafit(U, {'C'}, 'Reestimator', 'Ktauavg', ...
  'KendallMatrix', K);
toc

tic; disp('Estimating (2)...');
fit2Min = HACopulafit(U, {'C'}, 'Reestimator', 'taumin', ...
  'KendallMatrix', K);
toc

Estimating (1)...
Elapsed time is 4.583466 seconds.
Estimating (2)...
Elapsed time is 4.339935 seconds.
```

In the following code, these two estimates are evaluated in the same way as in Section 3. Note that the computations of the p values and of the probabilities available from the functions `prob` and `evalsurv` are omitted due to their run-time, as well as the computation of the distance matrix considering the upper tail dependence due to the fact that it is always zero for the Clayton family.

```
tic; disp('goodness-of-fit');
[gofdSnE(fit1Avg, U) gofdSnE(fit2Min, U)]
toc

tic; disp('kendall (HAC vs sample)');
[distance(fit1Avg, K) distance(fit2Min, K)]
toc

DISTANCE_TYPE = {'kendall', 'lower-tail'};
for i = 1:2
  tic; disp([DISTANCE_TYPE{i} ' (HAC vs HAC)']);
  [distance(fit1Avg, HACModel, DISTANCE_TYPE{i}) ...
    distance(fit2Min, HACModel, DISTANCE_TYPE{i})]
  toc
end

tic; disp('structures match ratio...')
[~, fit1AvgRatio] = comparestructures(HACModel, fit1Avg);
[~, fit2MinRatio] = comparestructures(HACModel, fit2Min);
[fit1AvgRatio fit2MinRatio]
toc

tic; disp('evaluating CDF at (0.5, ..., 0.5)...')
[cdf(fit1Avg, 0.5 * ones(1, HACModel.Dim)) ...
cdf(fit2Min, 0.5 * ones(1, HACModel.Dim))]
toc
```

```
goodness-of-fit

ans =

    0.0016    0.0016

Elapsed time is 1.565559 seconds.
kendall (HAC vs sample)

ans =

    0.0118    0.0107

Elapsed time is 1.045704 seconds.
```

```
kendall (HAC vs HAC)

ans =

    0.0040    0.0065

Elapsed time is 1.363244 seconds.
lower-tail (HAC vs HAC)

ans =

    0.0055    0.0117

Elapsed time is 1.460142 seconds.
structures match ratio...

ans =

    1.0000    0.9868

Elapsed time is 0.805491 seconds.
evaluating CDF at (0.5, ..., 0.5)...

ans =

    0.0011    0.0011

Elapsed time is 0.028698 seconds.
```

Observe that the times for all the performed procedures (excluding the slow computation of the matrix of pairwise Kendall's taus) have taken less then 5 seconds, which, in our opinion, makes **HACopula** a tool viable also for large-scale copula modeling. Also observe that the results for `fit1Avg` are slightly better (or equal) than the ones for `fit2Min` (of course, not taking into account the last evaluation) except for `kendall (sample)` where the values are relatively close. This is in accordance with the results reported in Górecki *et al.* (2017b). Considering the structure of the estimates, the function `comparestructures` shows 100% match with the structure of `HACModel` for `fit1Avg` and almost 99% match for `fit2Min`, which can also be visualized using `plot`.

Considering run times for the computations reported above, note that these might substantially differ for MATLAB and Octave, e.g., for sampling and for estimation we have observed even more than 10 times longer run times in Octave than in MATLAB.

In the R package **HAC**, an analogue to the estimators used above (including collapsing and re-estimation) is provided by the function `estimate.copula` with the input argument `method` set to `4`, which corresponds to so-called *penalized maximum likelihood* method; see Okhrin, Ristig, Sheen, and Trück (2015); Okhrin and Ristig (2019). To compare this estimator with the ones considered above, one can use the R script `highdimex.R` available in the folder

`Demos`, which computes a HAC estimate for the data generated above (`U`, also stored in the file `highdimex_data.mat` in the folder `Demos`). At our machine, the computation has taken approximately 6.5 hours and, by contrast to `fit1Avg`, the true (model's) structure has not been recovered. The latter can be verified by plotting the resulting estimate (which is also available as `highdimex_est.RData` in the folder `Demos`). The plot is obtained in R using the following code:

```
R> load("highdimex_est.RData")
R> library("HAC")
R> plot(est.obj)
```

Recall that with **HACopula**, the same job took approximately 8.5 seconds (= 4 seconds to get the matrix of pairwise Kendall's taus `K` plus 4.5 seconds to get `fit1Avg`).

6. Conclusion

The toolbox **HACopula** extends the current implementation of copulas in MATLAB and Octave to hierarchical Archimedean copulas. This provides the possibility to work with non-elliptical distributions in arbitrary dimensions allowing for asymmetries in the tails. In Octave, this moreover allows one to work with copulas in more than two dimensions. The toolbox implements functionality for constructing, evaluating, sampling, estimating and goodness-of-fit testing of hierarchical Archimedean copulas, as well as tools for their visual representation, accessing their analytic forms or computing matrices of pairwise Kendall's taus or tail dependence coefficients. This was demonstrated with several examples available as demos.

Acknowledgments

The authors are grateful to the project No. CZ.02.2.69/0.0/0.0/16_027/0008521 "Support of International Mobility of Researchers at SU" which supports international cooperation. The work has also been supported by the Czech Science Foundation (GAČR) grant 17-01251. The authors would especially like to thank Malte Kurz for sharing the C++ implementation of the Kendall's tau estimator from the **VineCopulaMATLAB** toolbox. Moreover, they are grateful for the valuable and detailed suggestions provided by two unknown reviewers.

References

Abrevaya J (1999). "Computation of the Maximum Rank Correlation Estimator." *Economics Letters*, **62**(3), 279–285. doi:10.1016/s0165-1765(98)00255-9.

Baxter A, Huddleston E (2014). *SAS/ETS 13.2 User's Guide*. SAS Institute Inc. URL http://support.sas.com/documentation/cdl/en/etsug/67525/PDF/default/etsug.pdf.

Christensen D (2005). "Fast Algorithms for the Calculation of Kendall's τ." *Computational Statistics*, **20**(1), 51–62. doi:10.1007/bf02736122.

Côté MP, Genest C, Abdallah A (2016). "Rank-Based Methods for Modeling Dependence Between Loss Triangles." *European Actuarial Journal*, **6**(2), 377–408. doi:10.1007/s13385-016-0134-y.

Durante F, Sempi C (2010). "Copula Theory: An Introduction." In P Jaworski, F Durante, WK Härdle, T Rychlik (eds.), *Copula Theory and Its Applications*, volume 198 of *Lecture Notes in Statistics*, pp. 3–31. Springer-Verlag. doi:10.1007/978-3-642-12465-5_1.

Eaton JW, Bateman D, Hauberg S, Wehbring R (2019). *GNU Octave Version 5.1.0 Manual: A High-Level Interactive Language for Numerical Computations*. URL https://www.gnu.org/software/octave/.

Genest C, Rémillard B (2008). "Validity of the Parametric Bootstrap for Goodness-of-Fit Testing in Semiparametric Models." *Annales De l'Institut Henri Poincaré, Probabilités et Statistiques*, **44**(6), 1096–1127. doi:10.1214/07-aihp148.

Genest C, Rémillard B, Beaudoin D (2009). "Goodness-of-Fit Tests for Copulas: A Review and a Power Study." *Insurance: Mathematics and Economics*, **44**(2), 199–213. doi:10.1016/j.insmatheco.2007.10.005.

Górecki J, Hofert M, Holeňa M (2016). "An Approach to Structure Determination and Estimation of Hierarchical Archimedean Copulas and Its Application to Bayesian Classification." *Journal of Intelligent Information Systems*, **46**(1), 21–59. doi:10.1007/s10844-014-0350-3.

Górecki J, Hofert M, Holeňa M (2017a). "Kendall's Tau and Agglomerative Clustering for Structure Determination of Hierarchical Archimedean Copulas." *Dependence Modeling*, **5**(1), 75–87. doi:10.1515/demo-2017-0005.

Górecki J, Hofert M, Holeňa M (2017b). "On Structure, Family and Parameter Estimation of Hierarchical Archimedean Copulas." *Journal of Statistical Computation and Simulation*, **87**(17), 3261–3324. doi:10.1080/00949655.2017.1365148.

Górecki J, Holeňa M (2014). "Structure Determination and Estimation of Hierarchical Archimedean Copulas Based on Kendall Correlation Matrix." In A Appice, M Ceci, C Loglisci, G Manco, E Masciari, ZW Ras (eds.), *New Frontiers in Mining Complex Patterns*, Lecture Notes in Computer Science, pp. 132–147. Springer-Verlag. doi:10.1007/978-3-319-08407-7_9.

Hofert M (2008). "Sampling Archimedean Copulas." *Computational Statistics & Data Analysis*, **52**(12), 5163 – 5174. doi:10.1016/j.csda.2008.05.019.

Hofert M (2010). *Sampling Nested Archimedean Copulas with Applications to CDO Pricing*. Ph.D. thesis, Universität Ulm. doi:10.18725/oparu-1787.

Hofert M (2011). "Efficiently Sampling Nested Archimedean Copulas." *Computational Statistics & Data Analysis*, **55**(1), 57–70. doi:10.1016/j.csda.2010.04.025.

Hofert M (2012). "A Stochastic Representation and Sampling Algorithm for Nested Archimedean Copulas." *Journal of Statistical Computation and Simulation*, **82**(9), 1239–1255. doi:10.1080/00949655.2011.574632.

Hofert M, Kojadinovic I, Maechler M, Yan J (2020). ***copula**: Multivariate Dependence with Copulas*. R package version 0.999-20, URL https://CRAN.R-project.org/package=copula.

Hofert M, Mächler M (2011). "Nested Archimedean Copulas Meet R: The **nacopula** Package." *Journal of Statistical Software*, **39**(9), 1–20. doi:10.18637/jss.v039.i09.

Hofert M, Mächler M, McNeil AJ (2013). "Archimedean Copulas in High Dimensions: Estimators and Numerical Challenges Motivated by Financial Applications." *Journal de la Société Française de Statistique*, **154**(1), 25–63.

Joe H (1997). *Multivariate Models and Dependence Concepts*. Chapman & Hall, London.

Joe H (2014). *Dependence Modeling with Copulas*. Chapman & Hall/CRC, New York. doi:10.1201/b17116.

Jun Yan (2007). "Enjoy the Joy of Copulas: With a Package **copula**." *Journal of Statistical Software*, **21**(4), 1–21. doi:10.18637/jss.v021.i04.

Kimberling CH (1974). "A Probabilistic Interpretation of Complete Monotonicity." *Aequationes Mathematicae*, **10**(2), 152–164. doi:10.1007/bf01832852.

Knight WR (1966). "A Computer Method for Calculating Kendall's Tau with Ungrouped Data." *Journal of the American Statistical Association*, **61**(314), 436–439. doi:10.1080/01621459.1966.10480879.

Kojadinovic I, Yan J (2010). "Modeling Multivariate Distributions with Continuous Margins Using the **copula** R Package." *Journal of Statistical Software*, **34**(9), 1–20. doi:10.18637/jss.v034.i09.

Kurz M (2016). ***VineCopulaMATLAB** Toolbox*. URL https://github.com/MalteKurz/VineCopulaMATLAB.

Macdonald CB (2017). ***OctSymPy**: A Symbolic Package for Octave Using **SymPy***. URL https://github.com/cbm755/octsympy.

Marshall AW, Olkin I (1988). "Families of Multivariate Distributions." *Journal of the American Statistical Association*, **83**(403), 834–841. doi:10.1080/01621459.1988.10478671.

McNeil AJ (2008). "Sampling Nested Archimedean Copulas." *Journal of Statistical Computation and Simulation*, **78**(6), 567–581. doi:10.1080/00949650701255834.

McNeil AJ, Nešlehová J (2009). "Multivariate Archimedean Copulas, d-Monotone Functions and l_1-Norm Symmetric Distributions." *The Annals of Statistics*, **37**(5B), 3059–3097. doi:10.1214/07-aos556.

Nelsen RB (2006). *An Introduction to Copulas*. 2nd edition. Springer-Verlag. doi:10.1007/0-387-28678-0.

Octave-Forge (2018). *The **Statistics** Package for Octave*. URL https://octave.sourceforge.io/statistics/.

Okhrin O, Okhrin Y, Schmid W (2013). "Properties of Hierarchical Archimedean Copulas." *Statistics & Risk Modeling*, **30**(1), 21–54. doi:10.1524/strm.2013.1071.

Okhrin O, Ristig A (2014). "Hierarchical Archimedean Copulae: The **HAC** Package." *Journal of Statistical Software*, **58**(4), 1–20. doi:10.18637/jss.v058.i04.

Okhrin O, Ristig A (2019). ***HAC**: Estimation, Simulation and Visualization of Hierarchical Archimedean Copulae (HAC)*. R package version 1.0-6, URL https://CRAN.R-project.org/package=HAC.

Okhrin O, Ristig A, Sheen JR, Trück S (2015). "Conditional Systemic Risk with Penalized Copula." *Discussion Paper 2015-038*, Sonderforschungsbereich 649: Ökonomisches Risiko, Humboldt-Universität Berlin. URL http://hdl.handle.net/10419/121999.

Okhrin O, Ristig A, Xu YF (2017). "Copulae in High Dimensions: An Introduction." In *Applied Quantitative Finance*, pp. 247–277. Springer-Verlag. doi:10.1007/978-3-662-54486-0_13.

R Core Team (2020). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Rezapour M (2015). "On the Construction of Nested Archimedean Copulas for d-Monotone Generators." *Statistics & Probability Letters*, **101**, 21–32. doi:10.1016/j.spl.2015.03.001.

SAS Institute Inc (2013). *The SAS System, Version 9.4*. SAS Institute Inc., Cary. URL http://www.sas.com/.

Savu C, Trede M (2010). "Hierarchies of Archimedean Copulas." *Quantitative Finance*, **10**(3), 295–304. doi:10.1080/14697680902821733.

Segers J, Uyttendaele N (2014). "Nonparametric Estimation of the Tree Structure of a Nested Archimedean Copula." *Computational Statistics & Data Analysis*, **72**, 190–204. doi:10.1016/j.csda.2013.10.028.

Sklar A (1959). "Fonctions De Répartition a n Dimensions et Leurs Marges." *Publications de l'Institut Statistique de l'Université de Paris*, **8**, 229–231.

The MathWorks Inc (2019). *MATLAB – The Language of Technical Computing, Version R2019a*. Natick, Massachusetts. URL http://www.mathworks.com/products/matlab/.

Uyttendaele N (2018). "On the Estimation of Nested Archimedean Copulas: A Theoretical and an Experimental Comparison." *Computational Statistics*, **33**(2), 1047–1070. doi:10.1007/s00180-017-0743-1.

Affiliation:

Jan Górecki
Department of Informatics
School of Business Administration in Karviná
Silesian University in Opava
Univerzitni namesti 1934/3, Karviná, Czech Republic
E-mail: gorecki@opf.slu.cz
URL: http://suzelly.opf.slu.cz/~gorecki/

Marius Hofert
Department of Statistics and Actuarial Science
Faculty of Mathematics
University of Waterloo
200 University Avenue West, Waterloo, ON, Canada
E-mail: marius.hofert@uwaterloo.ca
URL: http://www.math.uwaterloo.ca/~mhofert/

Martin Holeňa
Institute of Computer Science
Academy of Sciences of the Czech Republic
Pod vodárenskou věží 271/2, 182 07 Praha, Czech Republic
E-mail: martin@cs.cas.cz
URL: http://www2.cs.cas.cz/~martin/

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
May 2020, Volume 93, Issue 10 *Submitted:* 2017-03-16
doi:10.18637/jss.v093.i10 *Accepted:* 2018-12-16