Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. *doi: 10.18637/jss.v000.i00*

Hierarchical Archimedean Copulas for MATLAB: 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 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.

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

1. Introduction

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

$$F(x_1, ..., x_d) = C(F_1(x_1), ..., F_d(x_d)), \ \boldsymbol{x} \in \mathbb{R}^d, \tag{1}$$

into its univariate margins $F_1, ..., F_d$ 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 distributions 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 the elliptical copulas, they are given explicitly in terms of a real 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 (2016), such a hierarchical construction enables constructing copula models outperforming other recently popular multivariate copula models like *pair* or *factor* copulas. For their recent application to modeling dependence between so-called loss triangles, see Côté, Genest, and Abdallah (2016). Outside finance, e.g., Górecki, Hofert, and Holeňa (2016a) 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 package **copula** 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, 2015). One can also find packages implementing HACs in proprietary statistical software. As an example, sampling procedures involving three popular families of HACs have been recently implemented in SAS; see Baxter and Huddleston (2014, pp. 531). In MATLAB, which, similarly to SAS, also provides some support for two popular multivariate elliptical and three popular bivariate Archimedean families of copulas, however, no support for HACs is provided.

To fill this gap, our new **HACopula** toolbox for MATLAB introduces a comprehensive framework focused particularly on HACs, which implements procedures concerning sampling, estimation and goodness-of-fit testing. Not only do these procedures cover the basic features of the aforementioned packages, but the new toolbox also offers an implementation of all estimators recently introduced in Górecki, Hofert, and Holeňa (2016b), 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. Finally, as ACs are a special case of HACs, the new toolbox inherently complements their implementation in MATLAB (currently (R2017a) 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 in a quick-and-simple manner 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 concept in high dimensions, sampling, estimation and goodness-of-fit of a 100-variate HAC is presented in Section 5. 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

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

where ψ^{-1} is the general 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, pp. 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. The choice of this subset of families is also influenced by the fact that not all of those 22 families can be nested into each other in order to get a proper HAC; see Górecki *et al.* (2016b) 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.* (2016b) 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., $\tau_{(\text{C})}(\theta) = \theta/(\theta+2)$.

As has already been mentioned in the introduction, to construct a *hierarchical Archimedean copula* (HAC), one just need to replace some arguments in an AC by other HACs, 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 can be depicted like 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 $\{1, ..., m\}$, $m \in \mathbb{N}$, and $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$, can be derived by enumerating all of its nodes, e.g., like in Figure 1(b), where $\mathcal{V} = \{1, ..., 5\}$ 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

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 later 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).

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

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 the leaf indices in the both trees to correspond, i.e, the leaves 1, 2 and 3 in Figure 1(b) correspond to u_1, u_2 and u_3 in Figure 1(a), respectively, whereas the fork indices (4 and 5) are set arbitrarily, i.e., one can also derive a (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, it would be

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

Using this notation, (3) can be rewritten to

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

Observing that the indices of the arguments of the inner copula $C_{\lambda(4)}$ correspond to the set of the children of fork 4, i.e., to $\{2, 3\}$, and the the indices of the arguments of the outer copula $C_{\lambda(5)}$, taking the argument of $\lambda(\cdot)$ if $u_\cdot$ is not available, correspond to the set of the children of fork 5, i.e., to $\{1, 4\}$, 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 by $C_{(\mathcal{V},\mathcal{E},\lambda)}$ and we do so also in an arbitrary dimension; for a definition of HACs leaded in this way, see Definition 3.1 in Górecki *et al.* (2016b). 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

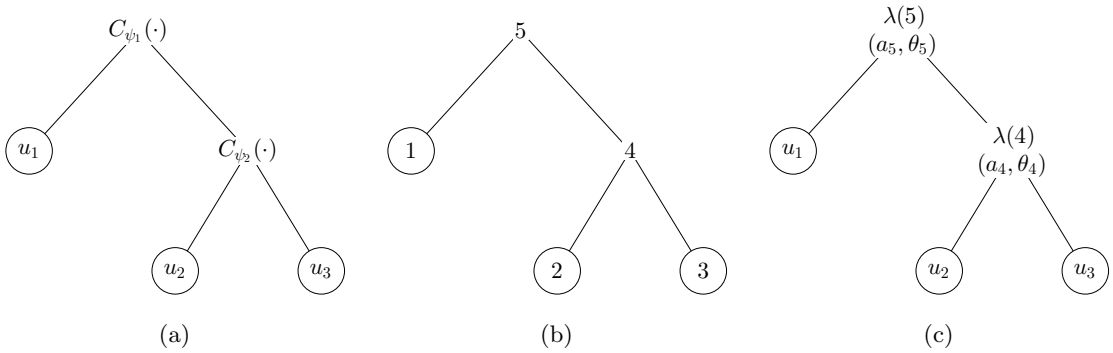


Figure 1: (a) A tree-like representation of a 3-variate HAC given by $C_{\psi_1,\psi_2}(u_1, u_2, u_3) = C_{\psi_1}(u_1, C_{\psi_2}(u_2, u_3))$. (b) An undirected tree $(\mathcal{V}, \mathcal{E})$, $\mathcal{V} = \{1, ..., 5\}$, $\mathcal{E} = \{\{1, 5\}, \{2, 4\}, \{3, 4\}, \{4, 5\}\}$ derived for the 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).

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 (obviously generalized to an arbitrary dimension) in the rest of the work.

To guarantee that a proper copula results from nesting ACs, we will use the *sufficient nesting condition* (SNC) proposed by Joe (1997, pp. 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 $\lambda(i)^{-1} \circ \lambda(j)$ 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 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 (2012). Note that Table 2 lists *all* family combinations of the generators of (Nelsen 2006, p. 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 these three advantages and thus its practical use, particularly in high dimensions, is challenging.

3. A quick example

The aim of this section is allowing the reader 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 quantify 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`.

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 $\theta_2 \in \Theta_{a_2}$.

(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)	$(0, \infty)$	$(0, \infty)$	$\theta_1 \le \theta_2$

3.1. Installing the HACopula toolbox

To install the toolbox, it is enough to unpack the files to a selected folder and to add this folder with its subfolders to the MATLAB path. Note that for the full functionality, the toolbox requires **Statistics and Machine Learning Toolbox** and **Symbolic Math Toolbox**. Also note that the toolbox has been intensively tested with the MATLAB versions R2013a and R2016a.

3.2. Constructing a HAC

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

$$\begin{aligned}\lambda(8) &= \psi^{(12,\tau^{-1}_{(12)}(0.8))},\\ \lambda(9) &= \psi^{(19,\tau^{-1}_{(19)}(0.7))},\\ \lambda(10) &= \psi^{(12,\tau^{-1}_{(12)}(0.5))},\\ \lambda(11) &= \psi^{(\mathrm{C},\tau^{-1}_{(\mathrm{C})}(0.2))}.\end{aligned} \tag{6}$$

To clarify such a definition of λ, which maps the forks in $(\mathcal{V}, \mathcal{E})$ to the corresponding generators, consider that, as $C_{(\mathcal{V},\mathcal{E},\lambda)}$ is assumed 7-variate, $\{1, ..., 7\}$ 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 $(\mathcal{V}, \mathcal{E})$, $\{8, ..., 11\}$ is the set of forks in $(\mathcal{V}, \mathcal{E})$. Also, observe that the generators are from the three different 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, let us nest these ACs into each other as given by

$$C_{(\mathcal{V},\mathcal{E},\lambda)} = C_{\lambda(11)}(C_{\lambda(9)}(u_2, u_5, u_6), C_{\lambda(10)}(u_1, C_{\lambda(8)}(u_3, u_4, u_7))). \tag{7}$$

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

Using the toolbox, (6) can be implemented using four `cell` arrays.

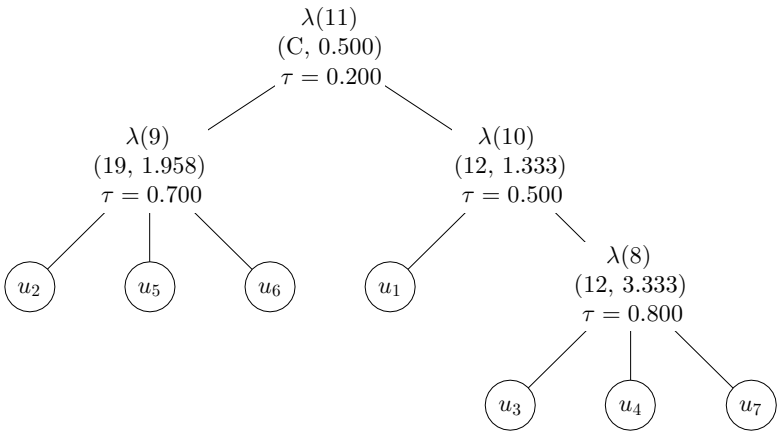


Figure 2: A 7-variate HAC including the three families C, 12 and 19 with the fork indices (the arguments of $\lambda(\cdot)$) ordered according to Kendall's tau (τ).

```
LAM8 = {'12', tau2theta('12', 0.8)}
LAM9 = {'19', tau2theta('19', 0.7)}
LAM10 = {'12', tau2theta('12', 0.5)}
LAM11 = {'C', tau2theta('C', 0.2)}

LAM8 =

    '12'    [3.3333]


LAM9 =

    '19'    [1.9576]


LAM10 =

    '12'    [1.3333]


LAM11 =

    'C'    [0.5000]
```

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 by the function `theta2tau`.

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 (7).

```
HACModel = HACopula({LAM11, {LAM9, 2, 5, 6}, {LAM10, 1, {LAM8, 3, 4, 7}}});
```

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 (`LAM11`) 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. The plot depicted in Figure 2 can be obtained by `plot(HACModel)`.

3.3. Computing probabilities involving a HAC

Having the HAC $C_{(\mathcal{V},\mathcal{E},\lambda)}$ represented by `HACModel`, one can let the toolbox compute several related probabilities. For example, using the method `evaluate`, one can evaluate $C_{(\mathcal{V},\mathcal{E},\lambda)}$ 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 $C_{(\mathcal{V},\mathcal{E},\lambda)}(0.5,...,0.5)$.

```
evaluate(HACModel, 0.5 * ones(1, getdimension(HACModel)))

ans =

    0.1855
```

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

```
prob(HACModel, 0.5 * ones(1, 7), 0.9 * ones(1, 7))

ans =

    0.0437
```

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

```
evalsurv(HACModel, 0.5 * ones(1, 7))

ans =

    0.1748
```

3.4. Sampling a HAC

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 its all 2-dimensional projections. Note that the first two lines just set the seed in order for the result to be reproducible.

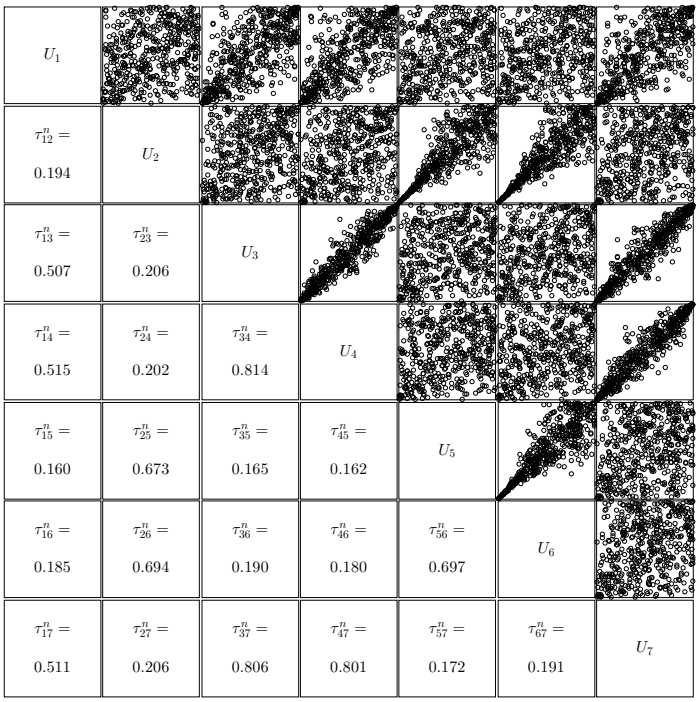


Figure 3: A sample of 500 observations from the $C_{(\mathcal{V},\mathcal{E},\lambda)}$ depicted in Figure 2.

```
rng('default');
rng(1);

UKnown = rnd(HACModel, 500);
plotbimargins(UKnown);
```

In Figure 3, obtained by the last line, one can observe the range of levels of dependencies, which are quantified in terms of Kendall's tau below the main diagonal, as well as different asymmetries in the tails; see also Section 3.6 below.

3.5. Estimating a HAC

Given i.i.d. observations $(X_{i1}, ..., X_{id})$, $i \in \{1, ..., n\}$ of a d-variate distribution function F given by (1), if the margins F_j, $j \in \{1, ..., d\}$ are known, one can estimate C directly using $(U_{i1}, ..., U_{id})$, $i = 1, ..., n$, where $U_{ij} = F_j(X_{ij})$, $i \in \{1, ..., n\}, j \in \{1, ..., d\}$. 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*

$$U_{ij} = \frac{n}{n+1}\hat{F}_{n,j}(X_{ij}) = \frac{R_{ij}}{n+1}, \tag{8}$$

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

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

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

Based on these pseudo-observations, the following code shows how to fit three 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 remaining two denoted by `fitC12` and `fitC` are fitted assuming the that this set is unknown – so we assume an arbitrary set of families from which the family of each generator is chosen – the Clayton family and the family 12, or just the Clayton family, respectively.

```
fitC1219 = HACopulafit(U, getfamilies(HACModel));
fitC12 = HACopulafit(U, {'C', '12'});
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 4 shows the plots of these estimates (the code for obtaining them is given in the captions).

In its default setting, i.e., for all three estimates above, the function `HACopulafit` implements the estimator *Coll=pre & Re-est=KTauAvg & Alg=PT-avg & Sn=R & Att=opt #Forks=unknown* proposed in Górecki *et al.* (2016b, Section 7), which is suggested as a good default in the reported simulation study.

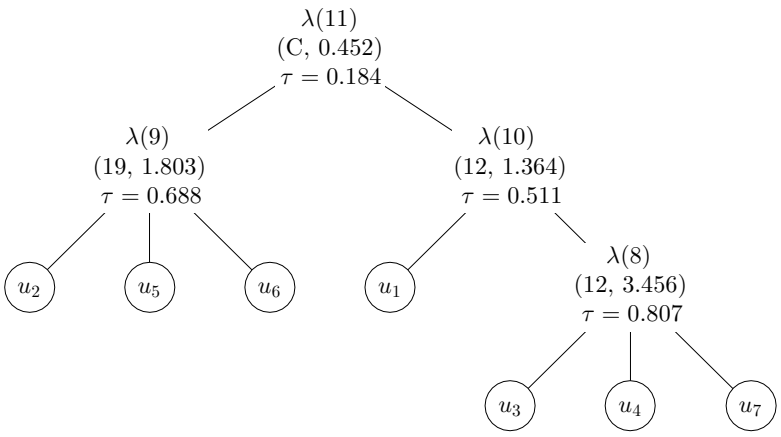
3.6. Goodness-of-fit testing for a HAC

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, where the unknown true copula is substituted by the so-called *empirical copula*).

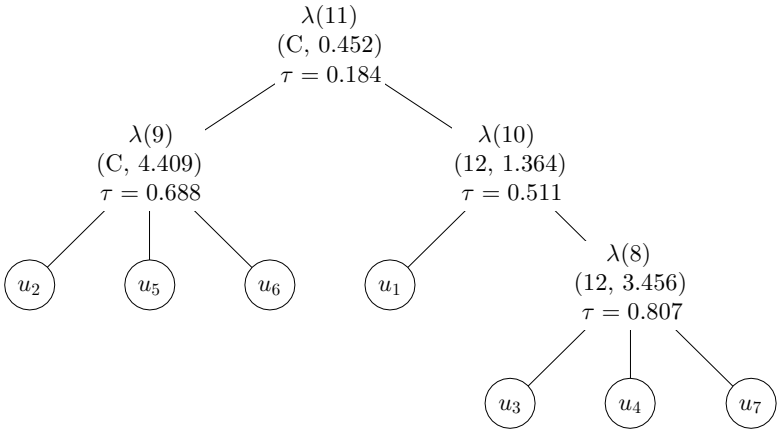
As a first approach, measuring a certain type of a distance between an estimate and the empirical copula corresponding to considered data is illustrated by the following code, where the goodness-of-fit statistics denoted $S_n^{(E)}$ in Górecki *et al.* (2016b) (proposed in Genest, Rémillard, and Beaudoin (2009) under the notation S_n) is computed for the three previously considered estimates.

```
[gofdSnE(fitC1219, U) ...
 gofdSnE(fitC12, U) ...
 gofdSnE(fitC, U)]

ans =

    0.0298    0.0318    0.0524
```

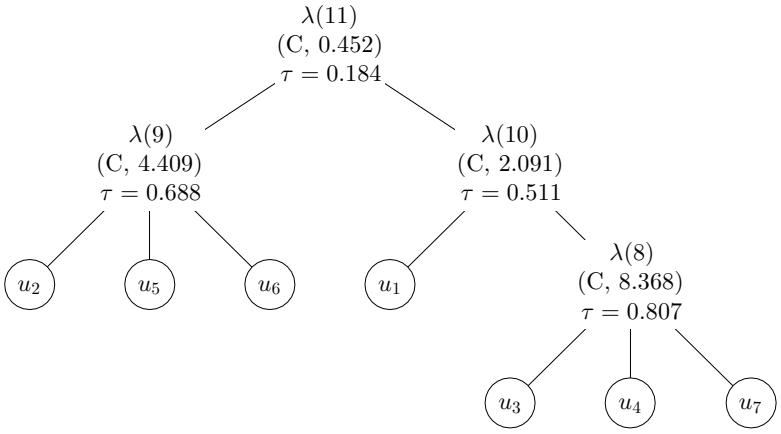
As might be expected, the more the underlying families are misspecified, the worse the fit (i.e., a larger distance) is reached.



(a) `plot(fitC1219)`



(b) `plot(fitC12)`



(c) `plot(fitC)`

Figure 4: Three estimates computed for the data depicted in Figure 3.

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

```
tic;
estimator1 = @(U) HACopulafit(U, getfamilies(HACModel));
computepvalue(fitC1219, U, estimator1, 100)
toc

ans =

    0.4700

Elapsed time is 73.730580 seconds.
```

As this computation is intensive, a stopwatch timer is involved at the first and the last input line (`tic` and `toc`); this computation, as well as all the following ones, was done on Intel Core 2.83 GHz processor. The value of the fourth argument of the method `computepvalue` (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, so, as the third argument, an anonymous function (defined at the second line) implementing it is provided. It is also worth mentioning that computing p values for HACs involving different families has not yet been reported in the literature.

Another goodness-of-fit provided by the toolbox considers a distance between a copula estimate and a sample (pseudo-observations), where the latter can be substituted by the the true copula if it is known. 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

$$\sqrt{\frac{1}{\binom{d}{2}} \sum_{i=1}^{d} \sum_{j=i+1}^{d} (\kappa_{ij}^{\square} - \kappa_{ij}^{\triangle})^2}, \tag{9}$$

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 dependence coefficients corresponding to *two HACs* (below denoted by `HAC vs HAC`), where $\kappa = \tau$ if the third argument of the method `distace` is `'kendall'`, $\kappa = \Lambda_u$ if this argument is `'upper-tail'` or $\kappa = \Lambda_l$ if it is `'lower-tail'`; see Table 1 for Λ_l and Λ_u.

```
K = corr(U,'type','kendall');

disp('kendall (HAC vs sample)');
[distance(fitC1219, K) ...
 distance(fitC12, 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(fitC12, HACModel, DISTANCE_TYPE{i}) ...
     distance(fitC, HACModel, DISTANCE_TYPE{i})]
end

kendall (HAC vs sample)

ans =

    0.0129    0.0129    0.0129

kendall (HAC vs HAC)

ans =

    0.0136    0.0136    0.0136

upper-tail (HAC vs HAC)

ans =

    0.0081    0.0081    0.3145

lower-tail (HAC vs HAC)

ans =

    0.0260    0.0608    0.0868
```

Note that the first line just computes the matrix of Kendall's pairwise taus for `U`. Observe that all resulting distances based on Kendall's tau are the same, which can be easily explained by looking at Figure 4, particularly at the τ values corresponding to the estimated generators. As these are the same for all three estimates, which implies that the Kendall's tau matrices corresponding to these estimates are the same, the result is not so surprising. To get a more detailed insight into where these values came from, one can let the toolbox compute the underlying Kendall's tau matrices by 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
```

Getting back to the distances computed according to (9), observe that the discrepancies among the estimates in terms of pairwise dependence coefficients are exposed through the distances based on the (both upper and lower) tail dependence coefficients. One again observes that the distances increase as the underlying families are more and more misspecified, particularly observe the relatively large value for the upper-tail distance for `fitC`, which is produced mainly due to the fact that this estimate is unable to model non-zero upper-tail dependence; see Λ_u for $a = \mathrm{C}$ in Table 1. One can also look at the underlying tail dependence matrices shown below 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.

```
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
```

3.7. Auxiliaries

Another handy tool, which does not fit to the previous sections, allows one to compare HAC structures, where the *structure* of a HAC is the set consisting of the sets of the descendant leaves of all forks, e.g, this set for `HACModel`, and also for the three estimates considered above, is $\{\{3,4,7\},\{2,5,6\},\{1,3,4,7\},\{1,...,7\}\}$; 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 =

     1
```

If the structures are the same for two HAC models, this method can also be used to compare the families of the involved generators.

```
[isSameStruc, isSameFams, nSameFams] = comparestructures(fitC1219, fitC12)

isSameStruc =

     1


isSameFams =

     0


nSameFams =

     3
```

The second 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 (`fitC12`) for all the involved generators, 0 otherwise; observe from Figures 4(a) and 4(b) that the families differ for $\lambda(9)$. The third output (`nSameFams`) returns the number of the families that are the same for the corresponding generators; observe the same families for $\lambda(8)$, $\lambda(10)$ and $\lambda(11)$ in Figures 4(a) and 4(b).

To generate an analytical form of `HACModel` exportable to LaTeX, the toolbox provides the method `tolatex`.

```
tolatex(HACModel, 'cdf')
```

The result is the following formula.

$$\left(\frac{1}{\left(\left(\frac{1}{u_1}-1\right)^{\theta_{10}}+\left(\left(\left(\frac{1}{u_3}-1\right)^{\theta_8}+\left(\frac{1}{u_4}-1\right)^{\theta_8}+\left(\frac{1}{u_7}-1\right)^{\theta_8}\right)^{\frac{1}{\theta_8}}\right)^{\theta_{10}}\right)^{\frac{1}{\theta_{10}}}+1}\right)^{-\theta_{11}}+\left(\frac{\theta_9}{\log\left(\mathrm{e}^{\frac{\theta_9}{u_2}}+\mathrm{e}^{\frac{\theta_9}{u_5}}+\mathrm{e}^{\frac{\theta_9}{u_6}}-2\,\mathrm{e}^{\theta_9}\right)}\right)^{-\theta_{11}}-1\right)^{-\frac{1}{\theta_{11}}}$$

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.

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 a HAC

As hinted in Section 3, the toolbox is built around its essential part – 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 fifth line).

```
LAM8 = {'12', tau2theta('12', 0.8)};
LAM9 = {'19', tau2theta('19', 0.7)};
LAM10 = {'12', tau2theta('12', 0.5)};
LAM11 = {'C', tau2theta('C', 0.2)};
HACModel = HACopula({LAM11, {LAM9, 2, 5, 6}, {LAM10, 1, {LAM8, 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]
         Child: {[1x1 HACopula]  [1x1 HACopula]}
        Parent: []
          Root: [1x1 HACopula]
         Forks: {[1x1 HACopula]  [1x1 HACopula]  [1x1 HACopula]  [1x1 HACopula]}
```

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]
         Child: {[1]  [1x1 HACopula]}
        Parent: [1x1 HACopula]
          Root: [1x1 HACopula]
         Forks: {[1x1 HACopula]  [1x1 HACopula]}
```

Given such a HAC representation, the properties `Leaves`, `Parent`, `Root` and `Forks` contain, respectively, its descendant nodes that are leaves, its parent (the 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`. This ordering is assigned by method `addtauordering` to all its forks anytime a new instance of `HACopula` is created.

Also note that the `HACopula` class is inherited from the abstract handle class `matlab.mixin.Copyable`, i.e., an instance of the `HACopula` is a *handle* object. This implies that, if a function modifies a `HACopula` object passed as an input argument, the modification affects the original input object. To create a copy of a `HACopula` instance, the toolbox provides an implementation of the `copy` method inherited from `matlab.mixin.Copyable`.

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 $\{1, ..., d\}$.

4.2. Sampling a HAC

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),

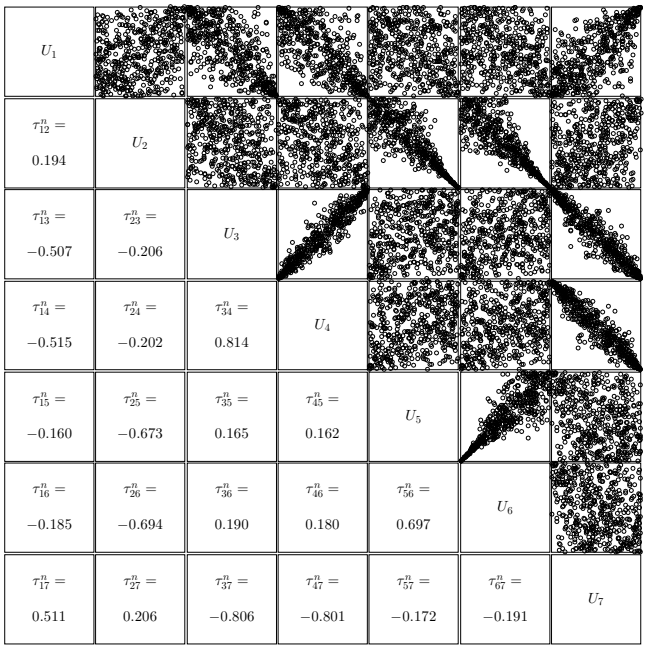


Figure 5: A sample of 500 observations from the HAC $C_{(\mathcal{V},\mathcal{E},\lambda)}$ depicted in Figure 2 with the flipped variables 1, 2 and 7.

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.* (2016a, 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 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 just flips the data in the columns corresponding to the variables indexed 1, 2 and 7; also called *rotation by 90° (180°)* if one (two) variable(s) in a pair is (are) flipped. The resulting sample obtained by `plotbimargins(UKnown)` is depicted in Figure 5.

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.* (2016a).

```
KNeg = corr(UKnown, 'type', 'kendall');
toFlip = findvars2flip(KNeg)

ans =

     2     7     1
```

Using this result, one can flip the data using `UKnown(:, toFlip) = 1 - UKnown(:, toFlip);`, which turn them to the purely positively correlated data depicted in Figure 3, which are yet suitable for modeling under the SNC. This approach does not always provide a solution, e.g., having 3-dimensional (e.g., real-world) data such that $\tau_{12} = \tau_{23} = 0.5$ 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 however unknown.

4.3. Estimating a HAC

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.* (2016b). To get a more complete picture of the possibilities offered by the `HACopulafit` function, 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');
```

For the details explaining the theoretical concept behind these input arguments, see Górecki *et al.* (2016b) together with Table 3, which links the notation from Górecki *et al.* (2016b) with the names of the arguments used in the `HACopulafit` function. This table, e.g., enables one to access all the estimators considered in the cited article, see Section 7 therein. How to use the estimators available in `HACopulafit` is described in the help comments provided with its implementation.

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 from estimation processes) to non-binary ones, allowing to access all possible HAC structures. In the simulation study reported in Górecki *et al.* (2016b), 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 example puts more light on such an approach.

Table 3: The left-hand column shows the features of the estimators considered in Górecki *et al.* (2016b, 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 as default, i.e., to `'average'` and to `'unknown'`, respectively.

Features from Górecki *et al.* (2016b)	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'`

```
fit2Bin = HACopulafit(U, {'?'}, 'PreCollapse', false);
K = corr(U, 'type', 'kendall');
[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 `'?'`. The resulting estimate is depicted in Figure 6 (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` in the way that two parent-child forks with the closest values of

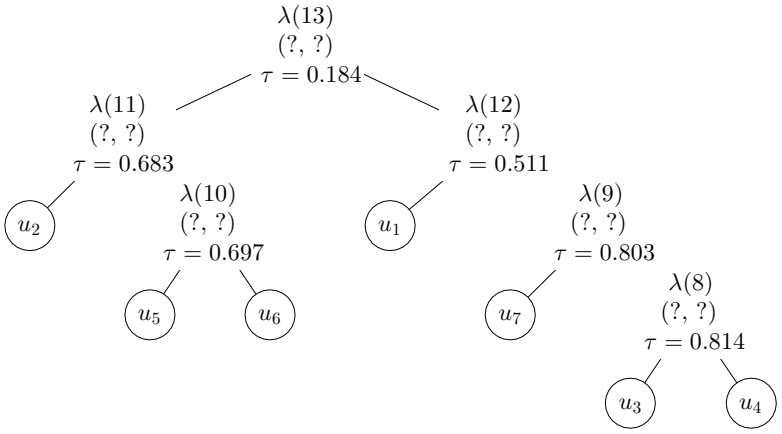


Figure 6: A binary structured HAC estimate obtained for the data depicted in Figure 3, where no assumption on the underlying families has been done, which is indicated by the arbitrary family denoted `'?'` used in the generators.

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. 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 according to her/his needs.

To help the user with this choice, the approach proposed in (Górecki *et al.* 2016b, Section 6.1) is implemented by the function `findjump`, which estimates the number of forks in the underlying HAC by detecting the first substantial jump in the distances stored in `minDistArray`. For our example, these distances are depicted in Figure 7. 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 7. Note that the function `HACopulafit`, if its input parameter `'PreCollapse'` is set `true`, uses this approach to estimate the number of forks 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 supplying the collapsed structure as optional input argument (structure estimation is avoided in such a case).

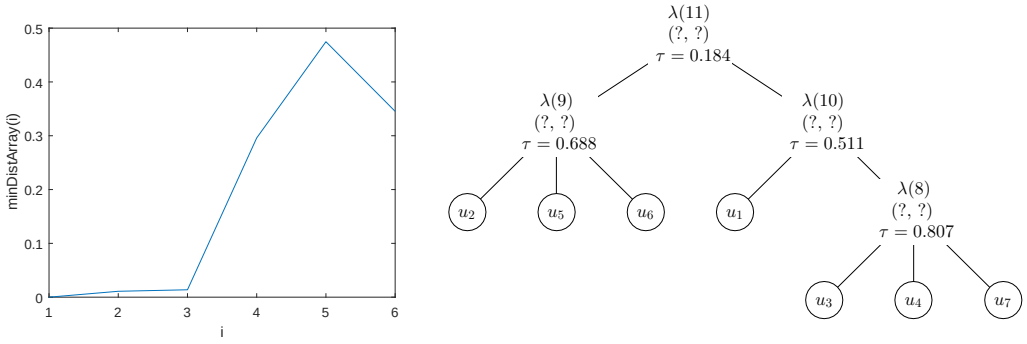


Figure 7: The left-hand side shows the values of `minDistArray`. The right-hand side shows the collapsed HAC `colHACArray{3}`.

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

Using `plot(fit2)`, the reader can check that it is the one depicted in Figure 4(a).

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 parameter `'PreCollapsedHAC'` addressed above, `'KendallMatrix'`, if supplied, avoids computation of the matrix of Kendall taus for a given data sample. This matrix can be computed by `corr(U, 'type', 'kendall')` 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 testing are performed on the same data (i.e., when more than one family is assumed for the generators), due to the fact that the goodness-of-fit testing intensively uses this input.

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 perfect 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`.

```
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]
------------------------------------------------------------------------
k = 3 *** Estimating \lambda(10):
I = [1  2  5  6  9]
g_1(K_{\downarrow(5), \downarrow(6)}) = 0.69712
Join leaves: \downarrow(10) = [5  6]
------------------------------------------------------------------------
k = 4 *** Estimating \lambda(11):
I = [1   2   9  10]
g_1(K_{\downarrow(2), \downarrow(10)}) = 0.68338
Join leaves: \downarrow(11) = [2  5  6]
------------------------------------------------------------------------
k = 5 *** Estimating \lambda(12):
I = [1   9  11]
g_1(K_{\downarrow(1), \downarrow(9)}) = 0.51114
Join leaves: \downarrow(12) = [1  3  4  7]
------------------------------------------------------------------------
k = 6 *** Estimating \lambda(13):
I = [11  12]
g_1(K_{\downarrow(11), \downarrow(12)}) = 0.18438
Join leaves: \downarrow(13) = [1  2  3  4  5  6  7]
------------------------------------------------------------------------
***** 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, 2.3852)
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]
------------------------------------------------------------------
k = 3 *** Estimating \lambda(10):
I = [1  8  9]
g_1(K_{\downarrow(1), \downarrow(8)}) = 0.51114
Admissible families + ranges:
{(C, [eps(0), 1]), (12, [1, 3.4559])}
Theta estimation:
family = C   theta = 2.0911
trimming theta to: 1 , i.e.,
```

```
family = C   theta = 1
family = 12   theta = 1.3637
Family estimation:
S_n^{g_2} for the families (C, 12) is (0.7264, 0.0370)
Best-fitting \psi^(family, theta) = \psi^(12, 1.3637)
Join leaves: \downarrow(10) = [1  3  4  7]
-----------------------------------------------------------------------
k = 4 *** Estimating \lambda(11):
I = [9  10]
g_1(K_{\downarrow(9), \downarrow(10)}) = 0.18438
Admissible families + ranges:
{(C, [eps(0), 1])}
Theta estimation:
family = C   theta = 0.45211
Family estimation:
S_n^{g_2} for the families (C) is (0.0000)
Best-fitting \psi^(family, theta) = \psi^(C, 0.45211)
Join leaves: \downarrow(11) = [1  2  3  4  5  6  7]
-----------------------------------------------------------------------
***** HACopulafit: Stop. *****
```

The notation used in the log above corresponds to the notation from Górecki *et al.* (2016b, 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... *****`. Also, after pre-collapsing, in step `k = 3`, one can observe that the set of the admisible families for the generator $\lambda(10)$ is reduced to the two families C and 12 (from the initial set of the three families C, 12 and 19), which follows from the SNC; see Table 2 for the admissible families if a child is from family 12 (here $\lambda(8)$ estimated in step `k = 1`). The algorithm also performs trimming in step `k = 3` under the assumption of the family C, which is motivated by an effort to satisfy the SNC under `'Attitude'` set to `'optimistic'`; see Table 2 and (Górecki *et al.* 2016b, Section 6.4.3) for details.

5. A high-dimensional example

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. The aim of this section is to demonstrate that such high-dimensional modeling can be accomplished with the `HACopula` toolbox. For this purpose, an example in which a 100-variate HAC is constructed, sampled, estimated, goodness-of-fit tested and evaluated is provided. As most of the functions and methods used in this example have already been addressed in the previous sections, we focus more on the computation times here. To simplify the construction of high-dimensional HAC models, the toolbox provides the auxiliary function `gethomomodel`, which builds a certain type of HAC models that are easily scalable to high dimensions. It is also worth to mention that in such high dimensions, estimation of a HAC including its structure has not yet been reported in the literature. Finally note that the example from this section can be reproduced using the file `highdimex.m` in the folder `Demos`.

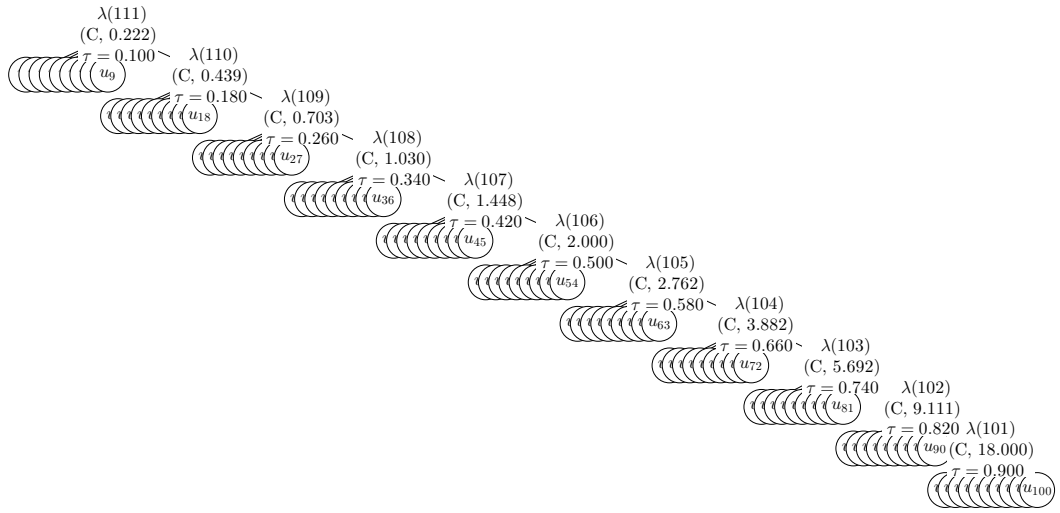


Figure 8: A 100-variate HAC with `11` nesting levels, where each level contains one `10`-variate AC from the Clayton family (`'C'`), with the parameter at the root corresponding to Kendall's tau = `0.1` and the differences between the parameters of a parent and its child corresponding to Kendall's tau = `0.08` (constructed via `gethomomodel(11, 10, 'C', 0.1, 0.08)`).

In the following code, the 100-variate HAC depicted in Figure 8 is constructed, a sample of 2000 observations from it is generated and two estimates, one based on the re-estimation approach proposed in Górecki *et al.* (2016b) (`fit1Avg`), the other on the re-estimation approach proposed in Uyttendaele (2016) (`fit2Min`), are computed for these observations. The corresponding computation times are shown in the output.

```
HACModel = gethomomodel(11, 10, 'C', 0.1, 0.08);

rng('default'); rng(1);
tic; disp('Sampling...'); U = pobs(rnd(HACModel, 2000));toc
tic; disp('Kednall''s matrix...'); K = corr(U,'type','kendall');toc

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
```

```
Sampling...
Elapsed time is 3.306638 seconds.
```

```
Kendall's matrix...
Elapsed time is 117.901371 seconds.
Estimating (1)...
Elapsed time is 5.574081 seconds.
Estimating (2)...
Elapsed time is 5.434821 seconds.
```

The most time expensive computation is clearly the computation of the Kendall correlation matrix, which involves $\binom{100}{2} = 4950$ computations of the sample version of Kendall's tau. However, computing it and passing it to `HACopulafit` results in comparably small estimation time.

In the following code, the 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 its run-time, as well as the computation of the distance 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('comparing the structures...')
[comparestructures(HACModel, fit1Avg) ...
comparestructures(HACModel, fit2Min)]
toc

tic; disp('evaluating at (0.5, ..., 0.5)...')
[evaluate(fit1Avg, 0.5 * ones(1, getdimension(HACModel))) ...
evaluate(fit2Min, 0.5 * ones(1, getdimension(HACModel)))]
toc

goodness-of-fit
```

```
ans =

    0.0016    0.0016

Elapsed time is 2.917311 seconds.
kendall (HAC vs sample)

ans =

    0.0118    0.0107

Elapsed time is 15.544844 seconds.
kendall (HAC vs HAC)

ans =

    0.0040    0.0065

Elapsed time is 28.667980 seconds.
lower-tail (HAC vs HAC)

ans =

    0.0055    0.0117

Elapsed time is 28.874969 seconds.
comparing the structures...

ans =

     1     0

Elapsed time is 0.020054 seconds.
evaluating at (0.5, ..., 0.5)...

ans =

    0.0011    0.0011

Elapsed time is 0.019727 seconds.
```

As one can observe, the considered distances (`kendall (sample)`, `kendall` and `lower-tail`) involve the most demanding calculations. Also observe that `fit1Avg` has more accurate (or equal) results than obtained for `fit2Min` (of course, not taking into account the last evaluation) except for `kendall (sample)` where the values are relatively close. Such a result is in accordance with the results reported in Górecki *et al.* (2016b). Also, if one considers the structure of `fit2Min` (e.g., using `plot(fit2Min)`), it can be observed that although the

structure is different to the structure of `HACModel`, both are rather similar, which also affects the evaluated results in the way that they are relatively close to the ones obtained for `fit1Avg`, which has the same structure as `HACModel`.

6. Conclusion

The toolbox **HACopula** extends the current implementation of copulas in MATLAB to hierarchical Archimedean copulas. This provides the possibility to work with non-elliptical distributions in arbitrary dimensions allowing for asymmetries in the tails. 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 Kendall matrices and tail dependence coefficients. This was demonstrated with several examples available as demos.

Acknowledgement

The research was supported by the Czech Science Foundation (GAČR) grant 17-01251.

References

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

Côté MP, Genest C, Abdallah A (2016). "Rank-based methods for modeling dependence between loss triangles." *European Actuarial Journal*, **6**(2), 377–408.

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. ISSN 0167-6687.

Górecki J, Hofert M, Holeňa M (2016a). "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.

Górecki J, Hofert M, Holeňa M (2016b). "On structure, family and parameter estimation of hierarchical Archimedean copulas." *arXiv preprint arXiv:1611.09225*.

Hofert M (2008). "Sampling Archimedean copulas." *Computational Statistics & Data Analysis*, **52**(12), 5163 – 5174. ISSN 0167-9473.

Hofert M (2010). *Sampling Nested Archimedean Copulas with Applications to CDO Pricing*. Suedwestdeutscher Verlag fuer Hochschulschriften.

Hofert M (2011). "Efficiently sampling nested Archimedean copulas." *Computational Statistics and Data Analysis*, **55**(1), 57–70. ISSN 0167-9473.

Hofert M (2012). "A stochastic representation and sampling algorithm for nested Archimedean copulas." *Journal of Statistical Computation and Simulation*, **82**(9), 1239–1255. doi:http://dx.doi.org/10.1080/00949655.2011.574632.

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*. CRC Press.

Kimberling CH (1974). "A probabilistic interpretation of complete monotonicity." *Aequationes Mathematicae*, **10**, 152–164.

Marshall AW, Olkin I (1988). "Families of multivariate distributions." *Journal of the American Statistical Association*, **83**(403), 834–841.

McNeil AJ (2008). "Sampling nested Archimedean copulas." *Journal of Statistical Computation and Simulation*, **78**(6), 567–581.

McNeil AJ, Nešlehová J (2009). "Multivariate Archimedean copulas, d-monotone functions and l_1-norm symmetric distributions." *The Annals of Statistics*, **37**, 3059–3097.

Nelsen RB (2006). *An Introduction to Copulas*. 2nd edition. Springer.

Okhrin O, Okhrin Y, Schmid W (2013). "Properties of hierarchical Archimedean copulas." *Statistics & Risk Modeling*, **30**(1), 21–54.

Okhrin O, Ristig A (2014). "Hierarchical Archimedean Copulae: The HAC Package." *Journal of Statistical Software*, **58**(4). ISSN 1548-7660. URL http://www.jstatsoft.org/v58/i04.

Okhrin O, Ristig A (2015). "Package 'HAC'." URL ftp://journal.r-project.org/pub/R/web/packages/HAC/HAC.pdf.

Okhrin O, Ristig A, Xu Y (2016). "Copulae in High Dimensions: An Introduction." Https://www.researchgate.net/profile/Yafei_Xu3/publication/309204418_ Copulae_in_High_Dimensions_An_Introduction/links/58052cad08aef179365e6dc2.pdf.

Rezapour M (2015). "On the construction of nested Archimedean copulas for d-monotone generators." *Statistics & Probability Letters*, **101**, 21–32.

Savu C, Trede M (2010). "Hierarchies of Archimedean Copulas." *Quantitative Finance*, **10**, 295–304.

Sklar A (1959). "Fonctions de répartition a n dimensions et leurs marges." *Publ. Inst. Stat. Univ. Paris*, **8**, 229–231.

Uyttendaele N (2016). "On the estimation of nested Archimedean copulas: A theoretical and an experimental comparison." *Technical report*, UCL. Http://hdl.handle.net/2078.1/171500.

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/en/

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/
MMMMMM YYYY, Volume VV, Issue II *Submitted:* yyyy-mm-dd
doi:10.18637/jss.v000.i00 *Accepted:* yyyy-mm-dd