Manual of `MATVines` Package

Package Version: `1.1`

Maximilian Coblenz[*]

February 19, 2021

[*]email: maximilian.coblenz@partner.kit.edu

1 Introduction

This manual serves as a documentation for the functions within the `MATVines` package – a vine copula toolbox for MATLAB. For an introduction to copulas the reader is referred to Nelsen (2006), Durante and Sempi (2015), and Genest and Favre (2007). For an introduction to vine copulas the reader is referred to Joe (2015), Czado (2019), Coblenz (2018), or the accompanying `MATVines` paper (Coblenz, 2021).

The manual is structured as follows: The next subsection gives guidance to installing the package. Section 2 gives a brief overview of the functions, scripts, and copulas implemented. Also the unit tests in the `unitTests.m` MATLAB file within the `MATVines` package can be used to see the functionality. Section 3 is a detailed documentation of the implemented functions which mainly corresponds to the in-code documentation. The last section shows examples on how to use the functions. Those and more examples are comprised in the `example.m` MATLAB file within the `MATVines` package.

The current version `1.1` of the `MATVines` package is freely available through the author's website[1]. It comes with a 3-clause BSD license.

1.1 Installing the Package

In order to install the `MATVines` package, put the files to a folder of your choice and add the `MATVines` folder to the MATLAB path. This can be done via *Home/Set Path* $\rightarrow$ *Add Folder...*. Note that the `MATVines` package requires the Statistics and Machine Learning Toolbox provided by MATLAB. The package has been extensively tested with MATLAB versions 2018a and 2018b and should be compatible with newer versions.

[1]`https://sites.google.com/view/maximiliancoblenz/startseite`

2 Overview of Functions, Scripts, and Copulas

2.1 List of Functions

Function Name	Short Description
`cdvinearray`	generates ordered vine array for C- and D-vine
`check_cube`	checks cube properties for function `simrvinetess`
`copmle`	function is needed to conduct MLE of copulas not implemented in MATLAB
`copulapdfadv`	computes PDF of a copula; more copulas than MATLAB's `copulapdf`
`copulaselect`	bivariate copula selection based on MLE, also single copula fit; provides more copulas than MATLAB's `copulafit`
`copulasim`	simulate bivariate copulas
`cpcheck`	checks alignment of copula and parameter
`cscrit`	selection criterion for `copulaselect`
`ecopula`	computes empirical copula
`goftest_a2`	goodness-of-fit test of type A2 in Berg (2009) for simplified vine copulas
`goftest_a4`	goodness-of-fit test of type A4 in Berg (2009) for simplified vine copulas
`hfunc`	h-function of a copula, i.e., conditional copula $C(u\|v)$
`hinv`	inverse of h-function of a copula, i.e., inverse of conditional copula $C^{-1}(u\|v)$
`iarray_vine`	needed for simulation of R-vine copula
`llrvine`	loglikelihood of simplified R-vine copula
`mle_copula`	implements functions to be inserted in `fminsearch` for MLE
`nktest`	model comparison of k models according to Nikoloulopoulos and Karlis (2008)
`nopcount`	counts number of parameters of vine copula
`pobs`	rank transform data
`primmaxst`	determine a maximum spanning tree
`rvineselect`	R-vine copula selection algorithm according to Dißmann et al. (2013)
`simrvine`	simulate simplified R-vine copula
`simrvinens`	simulate non-simplified R-vine copula with function handles
`simrvinetess`	simulate non-simplified R-vine copula with tesselated conditioning spaces (Coblenz, 2019)
`ssp`	stepwise semi-parametric estimation for simplified vine copulas
`thetalink`	link function between value and copula parameter, can be used in non-simplified simulation
`transforma`	transforms arbitrary feasible vine array to one with $a_{jj} = j$
`vinecdf`	CDF of simplified vine copula
`vinepdf`	PDF of simplified vine copula
`vuongtest`	Vuong's model comparison test (Vuong, 1989) for non-nested models

Table 1: List of functions provided by the `MATVines` package.

2.2 List of Additional Scripts

Script Name	Short Description
`example`	examples how to use the package
`unitTests`	unit tests of the functions

Table 2: List of scripts provided by the `MATVines` package.

2.3 List of Copulas

Copula	Computational Parameter Range	String Encoding
Independence	–	`ind`
Gauss	$\rho \in (-1, 1)$	`gauss`
t	$\rho \in (-1, 1), \nu \in [1, 1000000]$	`t`
Clayton	$\theta \in [0.00001, 150]$	`clayton`
Gumbel	$\theta \in [1, 120]$	`gumbel`
Frank	$\theta \in [-700, 700] \setminus \{0\}$	`frank`
Ali-Mikhail-Haq	$\theta \in [-1, 1]$	`amhaq`
Farlie-Gumbel-Morgenstern	$\theta \in [-1, 1] \setminus \{0\}$	`fgm`
Plackett	$\theta \in [0, 1000000]$	`plackett`
Joe	$\theta \in [1, 150]$	`joe`
Tawn	$\theta \in [0, 1]$	`tawn`
Survival Clayton	$\theta \in [0.00001, 150]$	`surclayton`
Survival Gumbel	$\theta \in [1, 120]$	`surgumbel`
Survival Joe	$\theta \in [1, 150]$	`surjoe`

Table 3: List of copulas implemented in the `MATVines` package. Note that the column *Computational Parameter Range* contains parameter ranges which are numerically stable. The theoretical parameter ranges can differ from this. For further reference to a particular copula see Nelsen (2006) and Joe (2015).

3 Function Documentation

This section provides a documentation of the functions. Also, we refer the reader to the in-code documentation.

`cdvinearray`

Generates the vine array A for a standard C- or D-vine.

call:	`A = cdvinearray(vine,d)`	
inputs	vine:	the type of vine; options are 'c' or 'd'
	d:	dimension
output	A:	$d \times d$ vine array; note that the diagonal elements a_{jj} fulfill $a_{jj} = j$

`check_cube`

Checks whether the hyperrectangles provided in cell array rectangles

1. cover the whole unit hypercube and
2. do not overlap.

call:	`bool = check_cube(rectangles)`	
input	rectangles:	cell array of rectangles as described in `simrvinetess`
output	bool:	0: check failed, 1: check completed

`copmle`

Helper function used in `copulaselect` for Maximum Likelihood Estimation.

call:	`[thetahat, loglik] = copmle(u,family)`	
inputs	u:	$n \times d$ matrix of pseudo observations
	family:	the copula family; see Table 2.3
outputs	thetahat:	the estimated copula parameter(s)
	loglik:	column vector of loglikelihoods

`copulapdfadv`

Computes the PDF of a copula at u.

call:	`pdf = copulapdfadv(family,u,theta)`	
inputs	family:	the copula family; see Table 2.3
	u:	$n \times 2$ matrix of points to be evaluated
	theta:	the copula parameter(s)
output	pdf:	column vector of PDF values at u

`copulaselect`

Performs a bivariate copula selection on data u based on Maximum Likelihood Estimation. The function uses Maximum Likelihood Estimation for copula selection. Depending on the number of inputs, the function chooses either one copula among different families of copulas (the one yielding the best selection criterion), estimates the parameter of one copula prespecified by the user, or selects the best fitting copula for a given set of copulas. If more than one family is prespecified the function selects the best fitting copula among the prespecified set.

call:	`[fam, thetahat, loglik, mcrit] = copulaselect(u[,crit,family1,family2,...])`	
inputs	u:	$n \times 2$ matrix of pseudo-observations
	crit (optional):	the selection criterion: 'aic' (Akaike's information criterion), 'bic' (Bayesian information criterion), 'sll' (sum of loglikelihoods); default is 'aic'
	family1 ... familyk (optional):	the copula family; see Table 2.3
outputs	fam:	the estimated copula family; see Table 2.3
	thetahat	the estimated copula parameter(s)
	loglik	column vector of loglikelihoods
	mcrit	the selection criterion value for the chosen model; in case the optional input family is specified by the user this is the sum of loglikelihoods

`copulasim`

Simulates a sample u from a bivariate copula. The function uses the conditioning and inversion technique for simulation.

call:	`u = copulasim(family,theta,n[,parallel])`	
inputs	family:	the copula family; see Table 2.3
	theta:	the copula parameter(s)
	n:	number of points to be simulated
	parallel (optional):	switch parallelization on (=1) or off (=0); default: 0
output	u:	$n \times 2$ matrix of simulated sample points

cpcheck

Checks whether the copula parameter theta is valid for a given copula family.

call:	`bool = cpcheck(family,theta)`	
inputs	family:	the copula family; see Table 2.3
	theta:	the copula parameter(s)
output	bool:	0: check failed, 1: check completed

cscrit

Helper function that computes the selection criterion for function `copulaselect`.

call:	`val = cscrit(crit,ll,nop)`	
inputs	crit:	the selection criterion: 'aic' (Akaike's information criterion), 'bic' (Bayesian information criterion), 'sll' (sum of loglikelihoods)
	ll:	column vector of loglikelihoods
	nop:	the number of parameters
output	val:	value of selection criterion

ecopula

Calculates the empirical copula values for multivariate data u based on data v.

call:	`ret = ecopula(u[,v])`	
inputs	u:	$n \times d$ data matrix with n observations over p dimensions, for which the empirical copula value is calculated
	v (optional):	$m \times p$ data matrix, on which the empirical copula is based
output	ret:	column vector of the empirical copula values for data u

`goftest_a2`

Conducts a goodness-of-fit test of type A2 from Berg (2009) for a specified simplified vine copula structure and given data. The test is based on a Cramer-von Mises statistic of the difference between empirical copula and estimated copula. Under the null hypothesis the estimated copula model has appropriate fit.

call:	`[tstat, pval] = goftest_a2(u,A,family,theta[,parallel])`	
inputs	u:	$n \times d$ data matrix of pseudo-observations
	A:	a vine array; note that a feasible structure has to be used, since the function does not check this
	family:	a $(d-1) \times (d-1)$ cell variable determining the copula families; see Table 2.3 for possible families
	theta:	a $(d-1) \times (d-1)$ cell variable of copula parameters
	parallel (optional):	switch parallelization on (=1) or off (=0); default: 0
outputs	tstat:	the test statistic value
	pval:	the corresponding p-value

`goftest_a4`

Conducts a goodness-of-fit test of type A4 from Berg (2009) for a specified simplified vine copula structure and given data. The test is based on a Cramer-von Mises statistic of the difference between the Kendall distribution functions of the empirical copula and of the estimated copula. Under the null hypothesis the estimated copula model has appropriate fit.

call:	`[tstat, pval] = goftest_a4(u,A,family,theta[,parallel])`	
inputs	u:	$n \times d$ data matrix of pseudo-observations
	A:	a vine array; note that a feasible structure has to be used, since the function does not check this
	family:	a $(d-1) \times (d-1)$ cell variable determining the copula families; see Table 2.3 for possible families
	theta:	a $(d-1) \times (d-1)$ cell variable of copula parameters
	parallel (optional):	switch parallelization on (=1) or off (=0); default: 0
outputs	tstat:	the test statistic value
	pval:	the corresponding p-value

`hfunc`

Computes the h-function of Aas et al. (2009), which is the conditional copula $C(u|v)$.

call:	`h = hfunc(u,v,family,theta)`	
inputs	u:	column vector of conditioned variable
	v:	column vector of conditioning variable
	family:	the copula family; see Table 2.3
	theta:	the copula parameter(s)
output	h:	column vector of h-function values

`hinv`

Computes the inverse of the h-function of Aas et al. (2009), which is the inverse of the conditional distribution $C(v|u)$, i.e. $C^{-1}(v|u)$.

call:	`hi = hinv(w,u,family,theta)`	
inputs	w:	value of conditioned variable
	v:	value of conditioning variable
	family:	the copula family; see Table 2.3
	theta:	the copula parameter(s)
output	hi:	the inverse h-function value

`iarray_vine`

Helper function to speed up algorithms for d-dimensional simplified R-vine copulas.

call:	`I = iarray_rvine(A)`	
input	A:	a vine array; note that a feasible structure has to be used, since the function does not check this
output	I:	indicator matrix needed for R-vine copula algorithms (e.g. simulation)

`llrvine`

Evaluates the loglikelihood function of a simplified R-vine copula.

call:	`[loglik,sll] = llrvine(u,A,family,theta)`	
inputs	u:	$n \times d$ data matrix of pseudo-observations
	A:	a vine array; note that a feasible structure has to be used, since the function does not check this
	family:	a $(d-1) \times (d-1)$ cell variable determining the copula families; see Table 2.3 for possible families
	theta:	a $(d-1) \times (d-1)$ cell variable of copula parameters
outputs	loglik:	column vector of loglikelihoods
	sll:	sum of loglikelihoods

`mle_copula`

Helper function for `fminsearch` in order to perform Maximum Likelihood Estimation for the parameter theta of the copula given in family.

call: `negll = mle_copula(u,family,theta)`

inputs	u:	$n \times 2$ matrix of points to be evaluated
	family:	the copula family; see Table 2.3
	theta:	the copula parameter(s)
output	negll:	negative loglikelihood

`nktest`

Conducts a model comparison test of k models according to Nikoloulopoulos and Karlis (2008) adapted for vine copulas. The minimum number of models compared is two. The function can compare k models at once. The null hypothesis is that model k is correct.

call:
```
[tstat, pval] = nktest(u,
    parallel,family1,theta1,A1,family2,theta2,A2[,family3,theta3,A3,...])
```

inputs	u:	$n \times d$ data matrix of pseudo-observations
	parallel:	switch parallelization on (=1) or off (=0); default: 0
	family1:	a $(d-1) \times (d-1)$ cell variable determining the copula families for model1; see Table 2.3 for possible families
	theta1:	a $(d-1) \times (d-1)$ cell variable of copula parameters for model1
	A1:	vine array of model1; note that a feasible structure has to be used, since the function does not check this
	family2:	a $(d-1) \times (d-1)$ cell variable determining the copula families for model2; see Table 2.3 for possible families
	theta2:	a $(d-1) \times (d-1)$ cell variable of copula parameters for model2
	A2:	vine array of model2; note that a feasible structure has to be used, since the function does not check this
	family3,...,familiyk (optional):	a $(d-1) \times (d-1)$ cell variable determining the copula families
	theta3,...,thetak (optional):	a $(d-1) \times (d-1)$ cell variable of copula parameters
	A3,...,Ak (optional):	a vine array; note that a feasible structure has to be used, since the function does not check this
outputs	tstat:	column vector of test statistic values
	pval:	column vector of corresponding p values

`nopcount`

Counts the number of parameters of a given d-dimensional arbitrary vine copula defined in input family.

call:	`nop = nopcount(family)`	
input	family:	a $(d-1) \times (d-1)$ cell variable determining the copula families used in the vine structure; see Table 2.3 for possible families
output	nop:	the number of parameters

`pobs`

Rank-transforms observation matrix X to pseudo-observation matrix u. The pseudo-observations can be used to work with copulas.

call:	`u = pobs(X)`	
input	X:	$n \times d$ data matrix, where n is the sample size and d is the number of dimensions
output	u:	the rank-transformed $n \times d$ matrix of pseudo-observations

`primmaxst`

Implements Prim's algorithm (Prim, 1957) for adjacency matrix A to determine a maximum(!) spanning tree.

call:	`list = primmaxst(A)`	
input	A:	$n \times n$ adjacency matrix of graph
output	list:	a list of the $n-1$ edges in the maximum spanning tree

`rvineselect`

Estimates an arbitrary simplified R-vine copula from data u according to the algorithm in Dißmann et al. (2013).

call:	`[A,fam,thetahat] = rvineselect(u[,crit])`	
inputs	u:	$n \times d$ data matrix of pseudo-observations
	crit (optional):	the selection criterion: 'aic' (Akaike's information criterion), 'bic' (Bayesian information criterion), 'sll' (sum of loglikelihoods); default is 'aic'
outputs	A:	the estimated vine array
	fam:	a $(d-1) \times (d-1)$ cell variable of copula families of the R-vine structure
	thetahat:	a $(d-1) \times (d-1)$ cell variable of estimated copula parameters corresponding to the copulas in fam

`simrvine`

Simulates a sample of size n from a d-dimensional simplified R-vine copula.

call:	`u = simrvine(n,A,family,theta[,parallel])`	
inputs	n:	number of sample points
	A:	a vine array; note that a feasible structure has to be used, since the function does not check this
	family:	a $(d-1) \times (d-1)$ cell variable determining the copula families; see Table 2.3 for possible families
	theta:	a $(d-1) \times (d-1)$ cell variable of copula parameters
	parallel (optional):	switch parallelization on (=1) or off (=0); default: 0
output	u:	$n \times d$ matrix of simulated points

`simrvinens`

Simulates a sample of size n from a d-dimensional non-simplified R-vine copula using function handles for conditional copula parameters. Note that

1. the function handle always gets handed the full vector of variables. So the variables which appear in the condition function always have to be specified - even in the case when only the first one is used! Make sure you choose the correct conditioning variables, i.e. indices, in the function handle corresponding to your vine structure, as the function will not check correctness on its own!

2. the possible values returned by the function handle have to be in line with the boundaries of the copula parameter of the employed copula. The function will not check this! If

you are not sure, whether the parameter boundaries are fulfilled, wrap your function by thetalink, which will transform function values into the correct parameter space, e.g. use `'@(x)(thetalink(3*x(1)^2+3,family))'` instead of `'@(x)(3*x(1)^2+3,family)'`.

3. if the condition includes more than one variable this has to be reflected in the function handle. For example use `'@(x)3*x(1)^2+x(2)+3'` for two conditioning variables.

4. even if the condition includes more than one variable, you can choose to make it depend on less than the full amount. The variables, which are not used can be left out in the function itself. For example use `'@(x)3*x(1)^2+x(3)+3'`, if you want the condition depend on variable one and three only.

call:	`u = simrvinens(n,A,family,theta[,parallel])`	
inputs	n:	number of sample points
	A:	a vine array; note that a feasible structure has to be used, since the function does not check this
	family:	a $(d-1) \times (d-1)$ cell variable determining the copula families; see Table 2.3 for possible families
	theta:	a $(d-1) \times (d-1)$ cell variable of copula parameters
	parallel (optional):	switch parallelization on (=1) or off (=0); default: 0
output	u:	$n \times d$ matrix of simulated points

`simrvinetess`

Simulates a sample of size n from a d-dimensional non-simplified R-vine copula with tesselated conditioning spaces (Coblenz, 2019). Note that the user has to input rectangles such that

1. the whole p-dimensional unit (hyper-)cube is partitioned by the k rectangles;

2. the p-dimensional (hyper-)rectangles do not overlap.

The function will check these conditions not exhaustively! Also note that the conditioning variables are assumed to be in ascending order.

call:	`u = simrvinetess(n,A,family,theta)`	
inputs	n:	number of sample points
	A:	a vine array; note that a feasible structure has to be used, since the function does not check this
	family:	a $(d-1)\times(d-1)$ cell variable determining the copula families; see Table 2.3 for possible families
	theta:	a $(d-1)\times(d-1)$ cell variable of copula parameters
	parallel (optional):	switch parallelization on (=1) or off (=0); default: 0
output	u:	$n \times d$ matrix of simulated points

`ssp`

Implements the stepwise semiparametric estimator for simplified vine copulas.

call:	`[thetahat,loglik,sll,fam] = ssp(u,A,family)`	
inputs	u:	$n \times d$ data matrix of pseudo-observations
	A:	a vine array; note that a feasible structure has to be used, since the function does not check this
	family:	a $(d-1)\times(d-1)$ cell variable determining the copula families; see Table 2.3 for possible families; if instead of a valid family 'aic', 'bic', or sll' is chosen, the program will select the best copula family on its own based on the chosen criterion; alternatively, a 1×1 cell variable with 'aic', 'bic', or 'sll' can be provided if all copulas should be estimated according to the chosen criterion
outputs	thetahat:	$(d-1)\times(d-1)$ cell variable of estimated parameters in the given vine copula structure
	loglik:	column vector of loglikelihoods for each data point
	sll:	the sum of logliklihoods
	fam:	a $(d-1)\times(d-1)$ cell variable indicating the copula families in the vine structure; this will be different to the input 'family' only if some copula families are not prespecified by the user but selected by the program

`thetalink`

Helper function that calculates the corresponding copula parameter value in `simrvinens` for restricting parameter values when using function handles.

call:	`tc = thetalink(beta,family)`	
inputs	beta:	value to be transformed
	family:	the copula family; see Table 2.3
output	tc:	parameter value in the correct range

`transforma`

Transforms an arbitrary feasible vine array A to an ordered vine array, i.e., such that the diagonal entries a_{jj} of A satisfy $a_{jj} = j$. For this the entries in A are permuted. This is equivalent to a renumbering/relabeling of the vine structure. The permutation for the renumbering is given in the output p, which contains the unsorted diagonal elements of A in the first column and the new labels in the second column.

call:	`[At,p,pinv] = transforma(A)`	
input	A:	feasible vine array
outputs	At:	ordered vine array, diagonal entries satisfy $at_{jj} = j$
	p:	the permutation of variables that yields the transformed vine array
	pinv:	the inverse of permutation p

`vinecdf`

Computes the CDF of a simplified vine copula via Monte-Carlo integration.

call:	`value = vinecdf(u,A,family,theta)`	
inputs	u:	$n \times d$ data matrix of pseudo-observations
	A:	a vine array; note that a feasible structure has to be used, since the function does not check this
	family:	a $(d-1) \times (d-1)$ cell variable determining the copula families; see Table 2.3 for possible families
	theta:	a $(d-1) \times (d-1)$ cell variable of copula parameters
output	value:	column vector of CDF values for each point in u

`vinepdf`

Computes the PDF of a simplified vine copula.

call:	`value = vinepdf(u,A,family,theta)`	
inputs	u:	$n \times d$ data matrix of pseudo-observations
	A:	a vine array; note that a feasible structure has to be used, since the function does not check this
	family:	a $(d-1) \times (d-1)$ cell variable determining the copula families; see Table 2.3 for possible families
	theta:	a $(d-1) \times (d-1)$ cell variable of copula parameters
output	value:	column vector of PDF values for each point in u

`vuongtest`

Conducts a model comparison test by Vuong (1989) for non-nested models. The Vuong test is a form of likelihood ratio test for model comparison/selection. This function implements the strictly non-nested version of the test. The output can be interpreted as follows: If pval is below your assumed error rate α, the test is significant. A positive tstat indicates that model 1 is superior compared to model 2. Vice versa, a negative tstat indicates that model 2 is superior compared to model 1.

call:	`[pval, tstat] = vuongtest(loglik1,loglik2,nop1,nop2)`	
inputs	loglik1:	column vector of loglikelihoods of model 1
	loglik2:	column vector of loglikelihoods of model 2
	nop1:	number of parameters of model 1
	nop2:	number of parameters of model 2
outputs	pval:	p-value according to a standard normal distribution
	tstat:	the test statistic

4 Examples

This section provides examples on how to use the `MATVines` package for tasks such as sampling and estimating a vine copula. The examples can be replicated by executing the MATLAB file `example.m` which is provided with the package.

4.1 Constructing a Vine Copula

A vine copula is defined by three structures: the vine array A, the copula families in the vine copula, and the specific parameter values of the copulas. This is also reflected in the `MATVines` package since the aforementioned structures have to be provided as inputs in many functions. In the following, we show with an exemplary 5-dimensional simplified vine copula how this is done in order to make the `MATVines` package work.

The vine array A is an upper triangular matrix and it is notated just like this in MATLAB. For example,

```
A = [1 1 2 3 3; 0 2 1 2 2; 0 0 3 1 4; 0 0 0 4 1; 0 0 0 0 5 ];
```

denotes the following 5-dimensional vine array

$$\mathbf{A} = \begin{pmatrix} 1 & 1 & 2 & 3 & 3 \\ & 2 & 1 & 2 & 2 \\ & & 3 & 1 & 4 \\ & & & 4 & 1 \\ & & & & 5 \end{pmatrix}.$$

Thus, the following copulas are present in the vine: c_{12}, c_{23}, c_{34}, c_{35}, $c_{13;2}$, $c_{24;3}$, $c_{25;3}$, $c_{14;23}$, $c_{45;23}$, and $c_{15;234}$. Figure 3 shows a graphical representation of this vine copula.

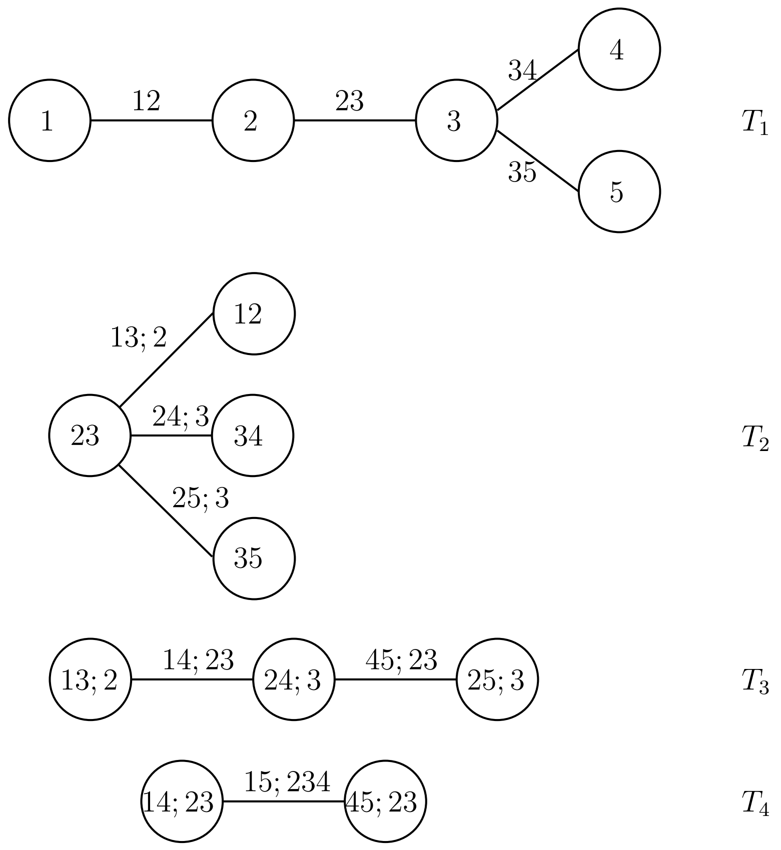

Figure 1: Normalized contour plot matrix of samples generated from a simplified vine copula (top), a non-simplified vine copula (middle), and a non-simplified vine copula with tessellated conditioning spaces (bottom).

The copula families within the vine array are denoted by a *cell* variable with the entries given as Strings, i.e.,

```
family1 = {'gauss', 'surjoe', 'clayton', 'gumbel'; 'clayton', 'clayton', 'frank',0;
'amhaq','clayton',0,0; 't',0,0,0};
```

Table 2.3 lists all copulas implemented in the package with their respective parameter values and String encodings. The structure of the *cell* variable reflects the families in the following way

$$\mathbf{family1} = \begin{pmatrix} \text{family12} & \text{family23} & \text{family34} & \text{family35} \\ \text{family13; 2} & \text{family24; 3} & \text{family25; 3} & 0 \\ \text{family14; 23} & \text{family45; 23} & 0 & 0 \\ \text{family15; 234} & 0 & 0 & 0 \end{pmatrix}.$$

Note that this corresponds to the appearance of the copulas in the vine array A from left to right. The parameter values of the copulas are also provided with a *cell* variable, where its structure follows the same pattern as the family1 variable, i.e.,

```
theta1 = {-0.7,2,3,4;2,3,5,0;0.8,3,0,0;[-0.4 5],0,0,0};
```

Note that for the t-copula there is a parameter vector, where the first entry represents the correlation parameter ρ and the second entry represents the degrees-of-freedom parameter ν.

4.2 PDF and CDF

The `MATVines` package can compute the PDF and CDF values of a vine copula at given points. This can be done with the `vinepdf` and `vinecdf` functions. For example, we want to calculate these values at the points $(0.1, 0.2, 0.3, 0.4, 0.5)$ and $(0.3, 0.4, 0.5, 0.6, 0.7)$ for the vine copula in the previous section. This can be done by calling

```
valuepdf = vinepdf([0.1,0.2,0.3,0.4,0.5;0.3,0.4,0.5,0.6,0.7],A,family1,theta1);
```

and

```
valuecdf = vinecdf([0.1,0.2,0.3,0.4,0.5;0.3,0.4,0.5,0.6,0.7],A,family1,theta1);
```

Hence, these functions take as input arguments the points for which the PDF and CDF have to be evaluated and the three structures vine array, copula families, and parameter values from the previous section.

4.3 Simulation of Vine Copulas

The `MATVines` package implements several simulation algorithms both for simplified and non-simplified vine copulas. Sampling from a simplified R-vine copula can be done with the function `simrvine` as follows

```
u = simrvine(1000,A,family1,theta1,0);
```

The first function input determines how many points are generated from the vine copula introduced in Section 4.1 and represented by A, family1, and theta1. The last input 0 is an optional parameter which controls parallelization (1 = parallelization on, 0 = parallelization off). The output u is a 1000×5 matrix here, where each column represents one of the variables. The simulation algorithm for a simplified R-vine copula can be found in Joe (2015).

Simulation from non-simplified R-vine copulas can be done with several functions. The function `simrvinens` allows the copula parameter to be a function of the conditioning variable(s). We illustrate the input arguments in the following. First, the copula family input is as in the case of simplified vine copulas

```
family2 = {'gauss','clayton','clayton','gumbel'; 'gauss','gumbel','gauss',0;
'gauss','clayton',0,0; 'gauss',0,0,0};
```

Second, the parameter values of the copulas now have to reflect the dependence on the conditioning variable(s). This is achieved by using function handles as parameter values

```
theta2 = {-0.6,3,2,3; 0.6,'@(u)(thetalink((4*u(3) - 2).^2,''gumbel''))','@(u)(thetalink
((4*u(3) - 2).^2,''gauss''))',0; '@(u)(thetalink((4*u(2)2̂+u(3)
- 2).^2,''gauss''))',4,0,0; '@(u)(thetalink((4*u(2)+u(3)+u(4) -
2).^2,''gauss''))',0,0,0};
```

Note that the function `thetalink` ensures that the parameter value stays in the feasible range of the copula. Finally, we can sample from this copula, where the input/output structure is the same as in the simplified case

```
u2 = simrvinens(1000,A,family2,theta2,0);
```

A different route to sample a non-simplified R-vine copula provides the function `simrvinetess`. This function implements sampling from non-simplified vine copulas with tessellation of conditioning spaces as introduced in Coblenz (2019). Here, the copula family input has to reflect the tessellation because on each part of the tessellation a different copula family is allowed.

```
family3 = {'gauss','clayton','clayton','gumbel'; 'gauss',{'gumbel' 'clayton'
'gumbel'},'gauss',0; {'gauss' 'frank'},'clayton',0,0; 'gauss',0,0,0};
```

The parameter value input now determines the exact tessellation. This is done by providing the lower left and upper right corners of the boxes within the tessellation.

```
theta3 = {-0.6,3,2,3; 0.6,{0.2 0.7 1; 2 3 4},{0.2 0.5 1; 0.5 -0.7 -0.1},0; {[0
0.5;0 1],[0.5 1; 0 1]; -0.5, 0.5},4,0,0; {[0 0.5;0 1;0 1],[0.5 1;0 1;0 1]; -0.5,
0.5},0,0,0};
```

Again, we can sample from this copula with the same input/output structure as in the simplified case

```
u3 = simrvinetess(1000,A,family3,theta3);
```

Note that simulation of D- and C-vine copulas can be done by using the functions above by providing an appropriate vine array A, because they are special cases of R-vine copulas.

Figure 2 shows normalized contour plot matrices (Czado, 2019) of samples from the three different vine copulas above.

4.4 Estimation of Vine Copulas

In the following, the sample u generated from the simplified vine copula in the previous section is used. The current version of the `MATVines` package supports two algorithms for simplified vine copula estimation. First, the stepwise semi-parametric (SSP) estimation procedure as introduced in Aas et al. (2009) and theoretically and empirically treated in Hobæk Haff (2012, 2013) is used. The function `ssp` estimates – depending on the input arguments – a simplified D-, C-, or R-vine copula. Apart from the data u as input, the user has to specify the vine array A because the SSP procedure does not estimate this. Also, the family input structure has to be specified. This can be done in different fashions. Either all the copulas in the vine copula are chosen according to an information criterion such as the Akaike Information Criterion (AIC) (Akaike, 1973) or some (or all) of the copula families are fixed beforehand and only the parameters are estimated. In the former case the family input can be specified as follows

```
familyssp = {'AIC'};
```

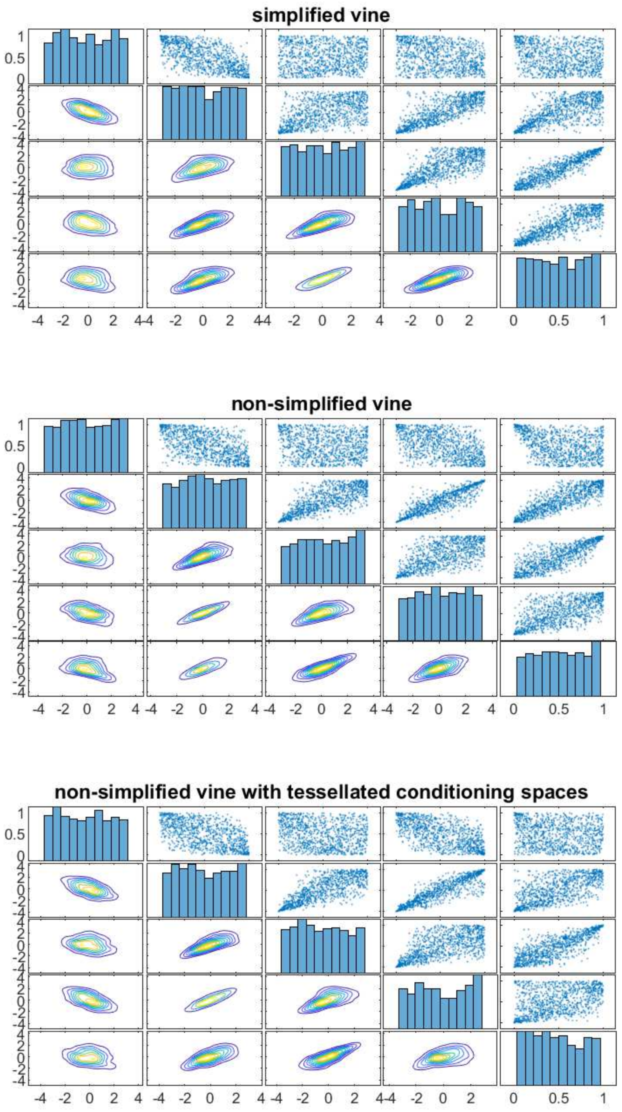

Figure 2: Normalized contour plot matrix of samples generated from a simplified vine copula (top), a non-simplified vine copula (middle), and a non-simplified vine copula with tessellated conditioning spaces (bottom).

Note that the input is a *cell* variable. In the latter case, the family *cell* variable has to be specified completely as outlined for simulation in the previous section. Estimation is accomplished as follows

```
[thetahat,loglik,~,famhat] = ssp(u,A,familyssp);
```

where thetahat contains the estimated parameters, famhat contains the estimated families, and loglik contains a column vector of the loglikelihood for each data point. Figure 3 shows a comparison of the original data u and data simulated from the estimated model above. As can be seen, the plots look similar.

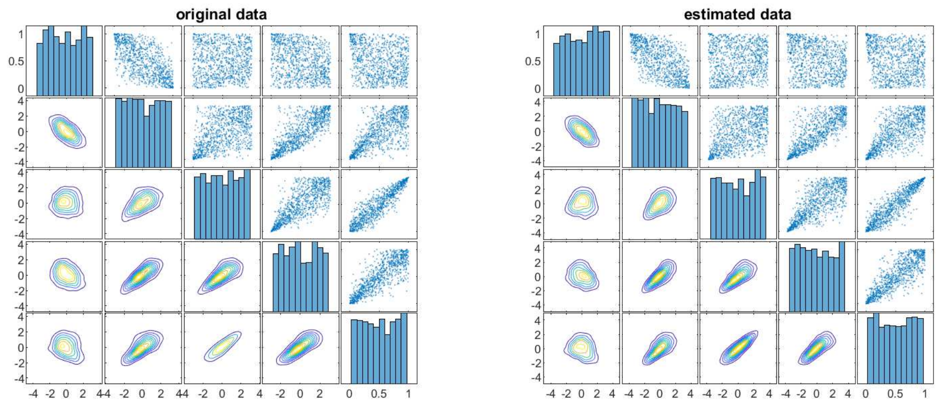

Figure 3: Normalized contour plot matrices of original data (left) and of data from the model estimated by the SSP procedure (right).

Second, the MATVines package implements an estimation heuristic for vine copulas known as Dißmann's algorithm (Dißmann et al., 2013) in the function rvineselect. This procedure also estimates the vine array A and, thus, the only input parameter is the sample given in u

```
[Ahat2, famhat2, thetahat2] = rvineselect(u);
```

Figure 4 shows a normalized contour plot of the original data u and simulated data from the model estimated by Dißmann's algorithm. Again, the plots look similar.

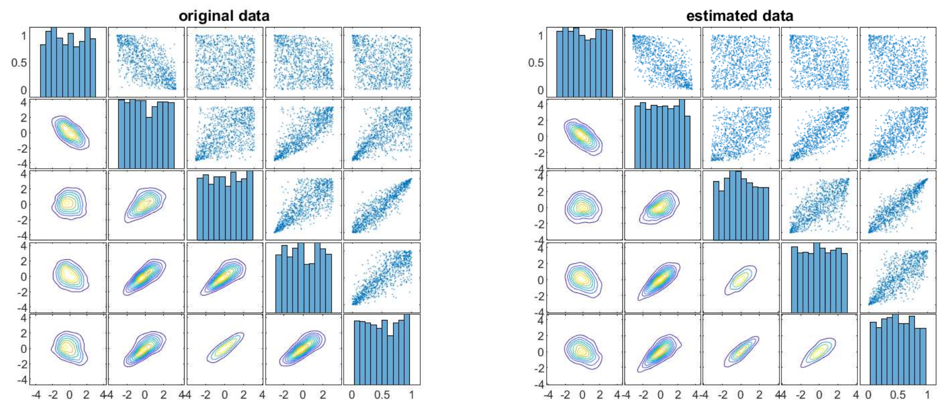

Figure 4: Normalized contour plot matrices of original data (left) and of data from the model estimated by Dißmann's algorithm (right).

4.5 Testing

The `MATVines` package also contains some utilities for goodness-of-fit testing and model comparison. In particular, the goodness-of-fits tests A2 and A4 in Berg (2009) are provided. Test A2 is a Cramer-von-Mises type of test based on the difference between empirical copula and estimated copula, whereas test A4 is a Cramer-von-Mises type of test based on the difference between the Kendall distribution functions of the empirical copula and the estimated copula. The tests can be conducted as follows

```
[tstat, pval] = goftest_a2(u,A,famhat,thetahat,1);

[tstat2, pval2] = goftest_a4(u,A,famhat,thetahat,1);
```

The last input parameter controls parallelization and is here set to 1 which means that parallelization is on. All the other inputs are analogously to the previous sections. The tstat output contains the test statistic and the output pval contains a p-value to check statistical significance directly.

Model comparison can be done with the test outlined in Nikoloulopoulos and Karlis (2008), which compares k models to each other. Since it is based on an excessive resampling strategy, the test has a considerable run time and parallelization as controlled by the second input should be on. The test can be conducted by

```
[tstat3, pval3] = nktest(u,1,famhat,thetahat,A,famhat2,thetahat2,Ahat2);
```

where more than two models can be input. Again, the tstat and pval outputs contain the test statistic and p-value, respectively.

Furthermore, the `MATVines` package provides the Vuong model comparison test for two non-nested models (Vuong, 1989). For this test, the loglikelihood of the data has to be computed for each data point. This can be achieved with the function `llrvine` by providing the usual inputs (note that the loglikelihoods are also an output of the estimation procedures, see the previous section)

```
ll1 = llrvine(u,A,famhat,thetahat);
```

```
ll2 = llrvine(u,Ahat2,famhat2,thetahat2);
```

Then, these loglikelihoods are input parameters for the `vuongtest` function together with the number of parameters of the two models, which can be computed via the function `nopcount`.

```
[pval4, tstat4] = vuongtest(ll1,ll2,nopcount(famhat),nopcount(famhat2));
```

4.6 Further Functionalities

With the function `copulasim` the `MATVines` package can be used to generate a bivariate sample of any copula implemented (see Table 2.3) as follows

```
u = copulasim('surgumbel',4,1000);
```

The first input determines the copula, the second input the copula parameter, and the third input the number of points sampled.

Rank-transformation of data can be done using the function `pobs`

```
u2 = pobs(u);
```

Figure 5 shows normalized contour plots of the original sample of the Survival Gumbel copula and the rank-transformed sample.

The `ecopula` function computes the empirical copula as follows

```
cophat = ecopula(u2);
```

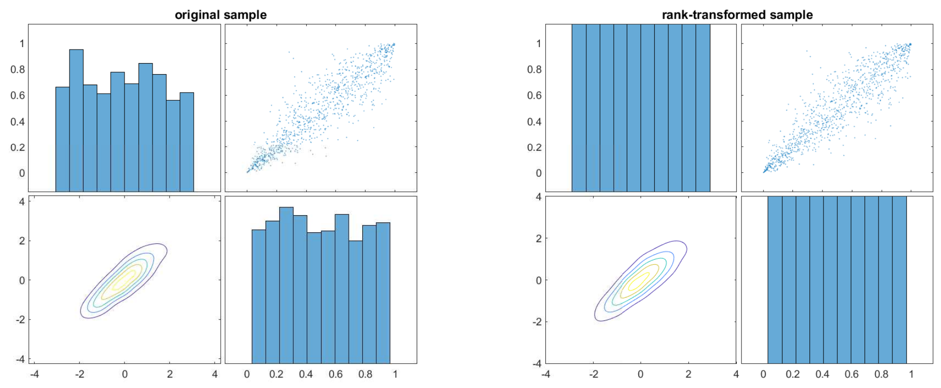

Figure 5: Normalized contour plot matrix of a sample generated from a Survival Gumbel copula (left) and the same sample rank-transformed (right).

The function `cdvinearray` creates the vine array of an ordered D-vine or ordered C-vine of any dimension. This can be helpful, when the vine array for simulation or estimation is needed. The first input parameter denotes the type of vine, the second input parameter the number of dimensions.

```
AC = cdvinearray('c',10);

AD = cdvinearray('d',15);
```

References

Aas, K., Czado, C., Frigessi, A., and Bakken, H. (2009). Pair-copula constructions of multiple dependence. *Insurance: Mathematics and Economics*, 44(2):182–198.

Akaike, H. (1973). Information Theory and an Extension of the Maximum Likelihood Princicple. In Petrov, B. and Csaki, E., editors, *Proceedings of the 2nd International Symposium of Information Theory*, pages 267–281, Budapest. Akademiai Kiado.

Berg, D. (2009). Copula goodness-of-fit testing: an overview and power comparison. *The European Journal of Finance*, 15(7-8):675–701.

Coblenz, M. (2018). *Advances in Dependence Modeling: Multivariate Quantiles, Copula Level Curve Lengths, and Non-Simplified Vine Copulas.* PhD thesis, Karlsruher Institut für Technologie (KIT).

Coblenz, M. (2019). Non-Simplified Vine Copulas via Tessellation of Conditioning Spaces. Working Paper.

Coblenz, M. (2021). MATVines: A Vine Copula Package for MATLAB. Working Paper.

Czado, C. (2019). *Analyzing Dependent Data with Vine Copulas: A Practical Guide With R.* Lecture Notes in Statistics ; 222Springer eBooks. Springer, Cham.

Dißmann, J., Brechmann, E., Czado, C., and Kurowicka, D. (2013). Selecting and estimating regular vine copulae and application to financial returns. *Computational Statistics & Data Analysis*, 59:52–69.

Durante, F. and Sempi, C. (2015). *Principles of copula theory.* CRC/Chapman & Hall.

Genest, C. and Favre, A.-C. (2007). Everything You Always Wanted to Know about Copula Modeling but Were Afraid to Ask. *Journal of Hydrologic Engineering*, 12(4):347–368.

Hobæk Haff, I. (2012). Comparison of estimators for pair-copula constructions. *Journal of Multivariate Analysis*, 110(0047):91–105.

Hobæk Haff, I. (2013). Parameter estimation for pair-copula constructions. *Bernoulli*, 19(2):462–491.

Joe, H. (2015). *Dependence Modeling with Copulas.* Chapman & Hall.

Nelsen, R. B. (2006). *An Introduction to Copulas.* Springer Series in Statistics. Springer New York, New York, NY.

Nikoloulopoulos, A. K. and Karlis, D. (2008). Copula model evaluation based on parametric bootstrap. *Computational Statistics & Data Analysis*, 52(7):3342–3353.

Prim, R. (1957). Shortest Connection Networks And Some Generalizations. *Bell System Technical Journal*, 36(6):1389–1401.

Vuong, Q. H. (1989). Likelihood Ratio Tests for Model Selection and Non-Nested Hypotheses. *Econometrica*, 57(2):307–333.