Package ‘FactorCopulaModel’

October 29, 2025

Title Factor Copula Models

Version 0.1.0

Description Inference methods for factor copula models for continuous data in Krupskii and Joe (2013) <doi:10.1016/j.jmva.2013.05.001>, Krupskii and Joe (2015) <doi:10.1016/j.jmva.2014.11.002>, Fan and Joe (2024) <doi:10.1016/j.jmva.2023.105263>, one factor truncated vine models in Joe (2018) <doi:10.1002/cjs.11481>, and Gaussian oblique factor models. Functions for computing tail-weighted dependence measures in Lee, Joe and Krupskii (2018) <doi:10.1080/10485252.2017.1407414> and estimating tail dependence parameter.

License GPL-3

Imports cubature, igraph, VineCopula

Encoding UTF-8

LazyData true

NeedsCompilation yes

Author Harry Joe [aut],
Pavel Krupskii [aut, cre],
Xinyao Fan [aut],
Allan Macleod [cph],
Robert Gentleman [cph],
Ross Ihaka [cph]

Maintainer Pavel Krupskii <pavel.krupskiy@unimelb.edu.au>

Depends R (>= 3.5.0)

Repository CRAN

Date/Publication 2025-10-29 20:00:11 UTC

Contents

bb1_cpar2td — *BB1 copula parameter (theta,delta) to tail dependence parameters*

Description

BB1 copula parameter (theta,delta) to tail dependence parameters

Usage

```
bb1_cpar2td(cpar)
```

Arguments

cpar	copula parameter with theta>0, delta>1 (vector of length 2) or mx2 matrix with columns for theta and delta

Value

vector or matrix with lower and upper tail dependence

Examples

```
cpar = matrix(c(0.5,1.5,0.8,1.2),byrow=TRUE,ncol=2)
bb1_cpar2td(cpar)
```

`bb1_tau2eqtd` *BB1, given 0<tau<1, find theta and delta with lower tail dependence equal upper tail dependence*

Description

BB1, given 0<tau<1, find theta and delta with equal lower/upper tail dependence

Usage

```
bb1_tau2eqtd(tau,destart=1.5,mxiter=30,eps=1.e-6,iprint=FALSE)
```

Arguments

`tau`	Kendall tau value
`destart`	starting point for delta
`mxiter`	maximum number of iterations
`eps`	tolerance for convergence
`iprint`	print flag for iterations

Value

copula parameter (theta,delta) with ltd=utd given tau

Examples

```
bb1_tau2eqtd(c(0.1,0.2,0.5))
```

`bb1_td2cpar` *BB1 tail dependence parameters to copula parameter (theta,delta)*

Description

BB1 map (lower,upper) tail dependence to copula parameter vector

Usage

```
bb1_td2cpar(taildep)
```

Arguments

`taildep`	tail dependence parameter in mx2 matrix, by row (ltd,utd) in (0,1)^2

Value

matrix of copula parameters, by row (theta,delta), theta>0, delta>1

Examples

```
cpar = bb1_td2cpar(c(0.4,0.6))
print(cpar)
#          theta    delta
#[1,] 0.3672112 2.060043
print(bb1_cpar2td(cpar))
```

---

bifactor2cor *Bi-factor partial correlations to correlation matrix*

Description

Bi-factor partial correlations to correlation matrix, determinant, inverse

Usage

```
bifactor2cor(grsize,rh1,rh2)
```

Arguments

grsize	vector with group sizes: d_1,d_2,...,d_G for G groups
rh1	vector of length sum(grsize) of correlation with global latent variable, ordered by group index
rh2	vector of length sum(grsize) of partial correlation with group latent variable given global

Value

list with Rmat = correlation matrix; det = det(Rmat); Rinv = solve(Rmat)

Examples

```
grsize = c(5,5,3)
d = sum(grsize)
bifpar = c(0.84,0.63,0.58,0.78,0.79, 0.87,0.80,0.74,0.71,0.57, 0.83,0.77,0.80,
0.67,0.58,0.15,0.70,0.47,   0.32,0.27,0.73,0.19,0.12,   0.35,0.23,0.53)
bifobj = bifactor2cor(grsize,bifpar[1:d],bifpar[(d+1):(2*d)])
rmat = bifobj$Rmat
print(det(rmat)-bifobj$det)
print(max(abs(solve(rmat)-bifobj$Rinv)))
bifobj2 = bifactor2cor_v2(grsize,bifpar[1:d],bifpar[(d+1):(2*d)])
rmat2 = bifobj2$Rmat
print(det(rmat2)-bifobj2$det)
print(max(abs(solve(rmat2)-bifobj2$Rinv)))
```

`bifactor2cor_v2` *Bi-factor partial correlations to correlation matrix version 2, using the inverse and determinant of a smaller matrix*

Description

Bi-factor partial correlations to correlation matrix, determinant, inverse

Usage

```
bifactor2cor_v2(grsize,rh1,rh2)
```

Arguments

`grsize`	vector with group sizes: d_1,d_2,...,d_G for G groups
`rh1`	vector of length sum(grsize) of correlation with global latent variable, ordered by group index
`rh2`	vector of length sum(grsize) of partial correlation with group latent variable given global

Value

list with Rmat = correlation matrix; det = det(Rmat); Rinv = solve(Rmat)

Examples

```
# see examples for bifactor2cor()
```

`bifactorcop_nllk` *negative log-likelihood of bi-factor structured factor copula and derivatives computed in f90 for input to posDefHessMinb*

Description

negative log-likelihood of bi-factor structured factor copula and derivatives

Usage

```
bifactorcop_nllk(param,dstruct,iprfn=FALSE)
```

Arguments

`param`	parameter vector; parameters for copulas linking U_ij and V_0 go first; parameters for copulas linking U_ij and V_g (j in group g) given V_0 go next. For BB1 linking copulas for the global latent, the order is theta1,..,theta[d],delta1, ...,delta[d]; V_0 is the global latent variable that loads on all variables; V_j is a latent variable that loads only for variables in group j (by group g=1,2,..,mgrp etc)
`dstruct`	list with data set $data, copula name $copname, $quad is a Gauss-Legendre quadrature object, $repar is a flag for reparametrization (for Gumbel, BB1), $nu is a 2-vector with 2 degree of freedom parameters (for t) $grsize is a vector with group sizes; if dstruct$pdf == 1 the function evaluates nllk only (and returns zero gradient and hessian). Options for copname are: frank, gumbel, gumbelfrank, bb1frank, bb1gumbel, t.
`iprfn`	indicator for printing of function and gradient (within Newton-Raphson iterations)

Value

nllk, grad, hess (gradient and hessian included)

Examples

```
grsize = c(4,4,3)
d = sum(grsize)
n = 500
mgrp = length(grsize)
par_bi = c(seq(1.4,3.4,0.2),seq(2.0,1.7,-0.1),seq(1.9,1.6,-0.1),rep(1.4,3))
set.seed(333)
udat_obj = rbifactor(n,grsize,cop=4,par_bi)
udat = udat_obj$data
summary(udat_obj$v0)
summary(udat_obj$vg)
zdat = qnorm(udat)
rmat = cor(zdat)
round(rmat,3)
# run bifactor_fa to get bi-factor correlation structure
bifa = bifactor_fa(grsize,start=c(rep(0.8,d),rep(0.2,d)),cormat=rmat,n=n,prlevel=0)
loading1 = bifa$parmat[,1] # correlations
pcor = bifa$parmat[,2] # partial correlations given global latent
pcor2 = pcor;  pcor2[pcor<0]=0.05  # for cases with only positive dependence
# starting values for different cases
# convert loading/pcor to Frank, Gumbel and BB1 parameters etc
# Frank for conditional given global latent can allow for conditional negative dependence
start_frk1 = frank_rhoS2cpar(loading1)
start_frk2 = frank_rhoS2cpar(pcor)
start_frk = c(start_frk1,start_frk2)
start_gum1 = gumbel_rhoS2cpar(loading1)
start_gum2 = gumbel_rhoS2cpar(pcor2)
start_gum = c(start_gum1,start_gum2)
start_tnu = c(loading1,pcor)
```

```
tau = bvn_cpar2tau(c(loading1))
# order of BB1 parameters has all thetas and then all deltas (different from 1-factor)
start_bb1 = bb1_tau2eqtd(tau)
start_bb1 = c(start_bb1[,1:2])
start_gumfrk = c(start_gum1,start_frk2)
start_bb1frk = c(start_bb1,start_frk2)
start_bb1gum = c(start_bb1,start_gum2)
#
gl = gaussLegendre(25)
npar = 2*d
dstrfrk = list(data=udat,copname="frank",quad=gl,repar=0,grsize=grsize,pdf=0)
dstrfrk1 = list(data=udat,copname="frank",quad=gl,repar=0,grsize=grsize,pdf=1)
obj1 = bifactorcop_nllk(start_frk,dstrfrk1) # nllk only
obj = bifactorcop_nllk(start_frk,dstrfrk) # nllk, grad, hess
print(obj1$fnval)
print(obj$grad)
ml_frk = posDefHessMinb(start_frk,bifactorcop_nllk,ifixed=rep(FALSE,npar),
dstrfrk, LB=rep(-20,npar), UB=rep(30,npar), mxiter=30, eps=5.e-5,iprint=TRUE)
dstrgum = list(data=udat,copname="gumbel",quad=gl,repar=0,grsize=grsize,pdf=0)
ml_gum = posDefHessMinb(start_gum,bifactorcop_nllk,ifixed=rep(FALSE,npar),
dstrgum, LB=rep(1,npar), UB=rep(20,npar), mxiter=30, eps=5.e-5,iprint=TRUE)
dstrgumfrk = list(data=udat,copname="gumbelfrank",quad=gl,repar=0,grsize=grsize,pdf=0)
ml_gumfrk = posDefHessMinb(start_gumfrk,bifactorcop_nllk,ifixed=rep(FALSE,npar),
dstrgumfrk, LB=c(rep(1,d),rep(-20,d)), UB=rep(25,npar), mxiter=30, eps=5.e-5,iprint=TRUE)
dstrtnu = list(data=udat,copname="t",quad=gl,repar=0,grsize=grsize,nu=c(10,20),pdf=0)
# slow because of many qt() calculations
# numerical issues because data does not have both upper and lower taildep
ml_tnu = posDefHessMinb(start_tnu,bifactorcop_nllk,ifixed=rep(FALSE,npar),
dstrtnu, LB=rep(-1,npar), UB=rep(1,npar), mxiter=30, eps=5.e-5,iprint=TRUE)
npar3 = 3*d
dstrbb1frk = list(data=udat,copname="bb1frank",quad=gl,repar=0,grsize=grsize,pdf=0)
ml_bb1frk = posDefHessMinb(start_bb1frk,bifactorcop_nllk,ifixed=rep(FALSE,npar3),
dstrbb1frk, LB=c(rep(0,d),rep(1,d),rep(-20,d)), UB=rep(20,npar3), mxiter=30, eps=5.e-5,iprint=TRUE)
dstrbb1gum = list(data=udat,copname="bb1gumbel",quad=gl,repar=0,grsize=grsize,pdf=0)
ml_bb1gum = posDefHessMinb(start_bb1gum,bifactorcop_nllk,ifixed=rep(FALSE,npar3),
dstrbb1gum, LB=c(rep(0,d),rep(1,2*d)), UB=rep(20,npar3), mxiter=30, eps=5.e-5,iprint=TRUE)
#
cat(ml_frk$fnval,ml_gum$fnval,ml_gumfrk$fnval,ml_tnu$fnval,ml_bb1frk$fnval,ml_bb1gum$fnval,"\n")
# -2256.602 -2574.16 -2509.274 -6793.963 -2514.124 -2581.214
cat(ml_frk$iter, ml_gum$iter, ml_gumfrk$iter, ml_tnu$iter, ml_bb1frk$iter, ml_bb1gum$iter, "\n")
# 5 6 5 23 15 12
# bi-factor t(10)/t(20) failed because some parameters approached the
# upper bound of 1 in which case the numerical integration is inaccurate.
```

`bifactorEstWithProxy` *Sequential parameter estimation for bi-factor copula with estimated latent variables using VineCopula::BiCopSelect*

Description

Sequential parameter estimation for bi-factor copula with estimated latent variables

Usage

```
bifactorEstWithProxy(udata,vglobal,vgroup,grsize,
  famset_global, famset_group, iprint=FALSE)
```

Arguments

`udata`	nxd matrix with values in (0,1)
`vglobal`	n-vector is estimated global latent variables (or test with known values)
`vgroup`	n*mgrp matrix with estimated group-based latent variables
`grsize`	G-vector with group sizes with mgrp=G=length(grsize)groups
`famset_global`	codes for allowable copula families for d global linking copulas
`famset_group`	codes for allowable copula families for d global linking copulas VineCopula: current choices to cover a range of tail behavior are: 1 = Gaussian/normal; 2 = t; 4 = Gumbel; 5 = Frank; 7 = BB1; 10 = BB8; 14 = survival Gumbel; 17 = survival BB1; 20 = survival BB8.
`iprint`	if TRUE print intermediate results

Details

It is best if variables have been oriented to be positively related to the latent variable

Value

list with fam = 2*d vector of family codes chosen via BiCopSelect;dx2 matrix global_par; dx2 matrix group_par; these contain par,par2 for the selected copula families in the 2-truncated vine rooted at the latent variables.

References

1. Krupskii P and Joe H (2013). Factor copula models for multivariate data. Journal of Multivariate Analysis, 120, 85-101. 2. Fan X and Joe H (2024). High-dimensional factor copula models with estimation of latent variables Journal of Multivariate Analysis, 201, 105263.

Examples

```
# BB1/Frank bi-factor copula
set.seed(2024)
th1_range = c(0.3,1)
th2_range = c(1.1,2.5)
th3_range = c(8.5,18.5)
grsize = rep(10,3)
mgrp = length(grsize)
d = sum(grsize)
```

```
parbi = c(runif(d,th1_range[1],th1_range[2]),  # BB1 theta
runif(d,th2_range[1],th2_range[2]),  # BB1 delta
runif(d,th3_range[1],th3_range[2]))  # Frank
n = 500
data = rbifactor(n,grsize=grsize,cop=7,parbi)
udata = data$data
vlat = cbind(data$v0,data$vg)
fam_true = c(rep(7,d),rep(5,d))
#
guess = c(rep(0.7,d),rep(0.5,d))
bif_obj = bifactorScore(udata, start=guess, grsize, prlev=1)
proxy_init = bif_obj$proxies
# selection of linking copula families and estimation of their parameters
select1 = bifactorEstWithProxy(udata,proxy_init[,1],proxy_init[,-1], grsize,
famset_global=c(1,4,5,7,10,17,20), famset_group=c(1,2,4,5))
fam1 = select1$fam
parglo1 = select1$global_par
pargrp1 = select1$group_par
print(fam1)  # 7,17,1 for global; 5 for group
print(parglo1)
print(pargrp1)
plot(parbi[(2*d+1):(3*d)],pargrp1[,1])
cor(parbi[(2*d+1):(3*d)],pargrp1[,1])
#
condExpProxy = latentUpdateBifactor(udata=udata, cparvec=c(parglo1,pargrp1),
grsize=grsize, family=fam1, nq=25)
proxy_improved = cbind(condExpProxy$v0, condExpProxy$vg)
par(mfrow=c(2,2))
for(j in 1:(mgrp+1))
{ plot(proxy_improved[,j],vlat[,j])
  print(cor(proxy_improved[,j],vlat[,j]))
}
rmse_values = sapply(1:ncol(proxy_improved),
function(i) sqrt(mean((proxy_improved[,i] - vlat[,i])^2)))
round(rmse_values,3)
#
# With improved proxies,
# selection of linking copula families and estimation of their parameters
select2 = bifactorEstWithProxy(udata,proxy_improved[,1],proxy_improved[,-1],
grsize, famset_global=c(1,4,5,7,10,17,20), famset_group=c(1,2,4,5))
parglo2=select2$global_par
pargrp2=select2$group_par
fam2 = select2$fam  # 7,17 for global; 5 for local
cbind(fam_true,fam1,fam2)
plot(parbi[(2*d+1):(3*d)],pargrp2[,1])
cor(parbi[(2*d+1):(3*d)],pargrp2[,1]) # higher correlation than pargrp1
```

`bifactorScore` *Proxies for bi-factor copula model based on Gaussian bi-factor score*

Description

Proxies (in (0,1)) for bi-factor copula model based on Gaussian bi-factor score

Usage

```
bifactorScore(udata, start, grsize, prlev=1)
```

Arguments

`udata`	nxd matrix in (0,1); n is sample size, d is dimension
`start`	starting values for fitting the bi-factor Gaussian model
`grsize`	G-vector with group sizes with G groups
`prlev`	printlevel in call to nlm

Value

list with Aloadmat=estimated loading matrix after N(0,1) transform; proxies = nx(G+1) matrix with stage 1 proxies for the latent variables (for global latent in column 1, and then for group latent); weight = weight matrix based on the correlation matrix of normal score and the loading matrix.

Examples

```
#See example in bifactorEstWithProxy()
```

`bifactor_fa` *Gaussian bi-factor structure correlation matrix*

Description

Gaussian bi-factor structure correlation matrix with quasi-Newton

Usage

```
bifactor_fa(grsize,start,data=1,cormat=NULL,n=100,prlevel=0,mxiter=100)
```

Arguments

`grsize`	vector of group sizes for bi-factor model
`start`	starting point should have dimension 2*d
`data`	nsize x d data set to compute the correlation matrix if correlation matrix (cormat) not given
`cormat`	dxd empirical correlation matrix
`n`	sample size
`prlevel`	print.level for nlm()
`mxiter`	maximum number of iterations for nlm()

Value

a list with$nllk, $parmat= dx2 matrix of correlations and partial correlations

Examples

```
data(rainstorm)
rmat = rainstorm$cormat
n = nrow(rainstorm$zprecip)
d = ncol(rmat)
grsize = rainstorm$grsize
fa1 = pfactor_fa(1,start=rep(0.8,d),cormat=rmat,n=n,prlevel=1)
fa2 = pfactor_fa(2,start=rep(0.8,2*d),cormat=rmat,n=n,prlevel=1)
st3 = c(rep(0.7,grsize[1]),rep(0.1,d),rep(0.7,grsize[2]),rep(0.1,d),rep(0.7,grsize[3]))
fa3 = pfactor_fa(3,start=st3,cormat=rmat,n=n,prlevel=1)
names(fa3)
loadmat_rotated = fa3$loading
# compare factanal
fa3b = factanal(factors=3,covmat=rmat)
compare = cbind(loadmat_rotated,fa3b$loadings)
print(round(compare,3)) # order of factors is different but interpretation similar
#
bifa = bifactor_fa(grsize,start=c(rep(0.8,d),rep(0.2,d)),cormat=rmat,n=n,prlevel=1)
mgrp = length(grsize)
# oblique factor model is much more parsimonious than bi-factor
obfa = oblique_fa(grsize,start=rep(0.7,d+mgrp), cormat=rmat, n=n, prlevel=1)
#
cat(fa1$nllk, fa2$nllk, fa3$nllk, bifa$nllk, obfa$nllk,"\n")
```

`bifactor_nllk` *log-likelihood Gaussian bi-factor structure correlation matrix*

Description

log-likelihood Gaussian bi-factor structure correlation matrix

Usage

```
bifactor_nllk(rhvec,grsize,Robs,nsize)
```

Arguments

`rhvec`	vector of length d*2 for partial correlation representation of loadings,
`grsize`	vector of group sizes for bi-factor model
`Robs`	dxd empirical correlation matrix
`nsize`	sample size

Value

negative log-likelihood and gradient for Gaussian bi-factor model

`bvnSemiCor` *Semi-correlation for bivariate normal/Gaussian distribution*

Description

semicorrelation assuming bivariate normal/Gaussian copula

Usage

```
bvnSemiCor(rho)
```

Arguments

`rho`	correlation in (-1,1)

Value

Cor(Z1,Z2| Z1>0,Z2>0) when (Z1,Z2)~bivariate standard normal(rho)

References

Joe (2014), Dependence Modeling with Copulas, Chapman&Hall/CRC; p 71

`bvn_cpar2tau` *Kendall's tau for bivariate normal*

Description

Kendall's tau for bivariate normal

Usage

```
bvn_cpar2tau(rho)
```

Arguments

`rho`	in (-1,1)

Value

Kendall's tau = 2*arcsin(rho)/pi

corDis — *Discrepancy of model-based and observed correlation matrices based on Gaussian log-likelihood*

Description

Discrepancy of model-based and observed correlation matrices

Usage

```
corDis(Rmodel,Rdata,n=0,npar=0)
```

Arguments

Rmodel	model-based correlation matrix
Rdata	empirical correlation matrix (could be observed or polychoric)
n	sample size (if positive integer)
npar	#parameters in the correlation structure

Value

vector with discrepancy Dfit, and also nllk2 (wice negative log-likelihood), BIC, AIC if n and npar are inputted

Examples

```
Rmodel = matrix(c(1,.3,.4,.4,.3,1,.5,.6,.4,.5,1,.7,.4,.6,.7,1),4,4)
print(Rmodel); print(chol(Rmodel))
Rdata = matrix(c(1,.32,.38,.41,.32,1,.53,.61,.38,.53,1,.67,.41,.61,.67,1),4,4)
print(corDis(Rmodel,Rdata))
print(corDis(Rmodel,Rdata,n=400,npar=3))
```

corvec2mat — *Convert from correlations in vector form to a correlation matrix*

Description

Convert from correlations in vector form to a correlation matrix

Usage

```
corvec2mat(rvec)
```

Arguments

`rvec`	correlations in vector form of length d*(d-1)/2 in the order r12,r13,r23,r14,... r[d-1,d]

Value

dxd correlation matrix

Examples

```
rvec = c(0.3,0.4,0.5,0.4,0.6,0.7)
Rmat = corvec2mat(rvec)
print(Rmat) # column 1 has 1, 0.3, 0.4, 0.4
```

`cparBounds` *lower and upper bounds for copula parameters (1-parameter, 2-parameter families)*

Description

lower and upper bounds for copula parameters for use in min negative log-likelihood

Usage

```
cparBounds(familyvec)
```

Arguments

`familyvec`	vector of family codes linking copula families

Value

lower bound LB1/LB2 and upper bound LB2/UB2 for par1 and par2

Examples

```
famvec = c(1,4,5,14,2,7,17,10,20, 4,5,1)
out = cparBounds(famvec)
print(out)
print(out$LB1)
```

`d1factcop`	*Integrand for 1-factor copula with 1-parameter bivariate linking copula families; or for m-parameter bivariate linking copulas*

Description

Integrand for 1-factor copula

Usage

```
d1factcop(u0,uvec,dcop,param)
```

Arguments

`u0`	latent variable for integrand
`uvec`	vector of length d, components in (0,1)
`dcop`	name of function of bivariate copula density (common for all variables), dcop accepts input of form of d-vector or dxm matrix
`param`	d-vector or mxd matrix, parameter of dcop

Value

integrand for 1-factor copula density

`DJ20142016gf`	*GARCH-filtered log returns for Dow Jones stocks 2014-2016*

Description

R workspace file with GARCH-filtered log returns for Dow Jones stocks 2014-2016

Three objects:

1. dj1416gf is a list with $filter, $uscore $zscore $uscmodel $zscmodel $sigmat $coefficient; dimensions of matrices $uscore, $zscore are 764 x 30 (n x d).

filter: GARCH filtered data nxd, before transform

uscore: empirical uniform scores (nxd)

zscore: empirical normal scores (nxd)

uscmodel: model-based uniform scores (nxd) from the GARCH fit

zscmodel: model-based normal scores (nxd) from the GARCH fit

sigmat: matrix of estimated volatilities (nxd)

coef: matrix of GARCH parameters (6xd or 5xd depending on where AR(1) was used for GARCH model; the parameters are mu, (ar1), omega, alpha1, beta1, shape.

2. dateindex is an object with the dates for the rows of $uscore, $zscore

3. lab is an object with the ticker names for the 30 stocks in dj1416gf

Usage

```
data(DJ20142016gf)
```

`euro07` *log returns and GARCH-filtered log returns for some Euro markets 2007*

Description

Log returns and GARCH filtered values of log returns for OSEAX FTSE AEX FCHI SSMI GDAXI ATX (market indexes in Norway, UK, Holland, France, Switzerland, Germany, Austria). This is a small data set with n=239 that can be used for illustration of functions for fitting vine and factor copulas. The original source is http://quote.yahoo.com

euro07gf has two objects: (i) euro07names has the above labels for the markets and (ii) euro07lr is a list with several matrices given below.

Usage

```
data(euro07gf) # objects euro07names and euro07gf
```

Format

The following are components.

filter 239x7 matrix of GARCH filtered returns

uscore 239x7 matrix of empirical U(0,1) scores of GARCH filtered returns

zscore 239x7 matrix of empirical normal scores of GARCH filtered returns

uscmodel 239x7 matrix of U(0,1) scores of GARCH filtered returns based on assuming standardized Student t distributions for the innovations

zscmodel 239x7 matrix of normal scores of GARCH filtered returns based on assuming standardized Student t distributions for the innovations

sigmat 239x7 matrix of volatilities

coef 5x7 matrix of GARCH parameter estimates rows are mu, omega, alpha1, beta1, shape, where 'shape' is the shape or degree of freedom parameter for the Student t innovations.

factor1trvine_nllk *negative log-likelihood with gradient and Hessian computed in f90 for copula from 1-factor/1-truncated vine (tree for residual dependence conditional on a latent variable); models included are BB1 for latent with Frank or Gaussian(bvncop) for truncated vine residual dependence*

Description

negative log-likelihood for 1-factor with tree for residual dependence

Usage

```
factor1trvine_nllk(param,dstruct,iprint=FALSE)
```

Arguments

`param`	parameter vector (length 2*d+(d-1)); BB1 parameters for links of observed variable U_j to latent V, j=1,...,d; Frank or Gaussian parameters for edge_k=(node_k[1],node_k[2]) observed variables nodek[1],nodek[2] given V, k=1,...,d-1, where the d-1 edges form a tree. 2*d BB1 parameters (theta_1,delta_1,...,theta_d,delta_d); d-1 Frank or BVN parameters for residual dependence.
`dstruct`	list with nxd data set $data on (0,1)^d ; $quad is list with quadrature weights and nodes; $edg1, $edg2 (d-1)-vectors for node labels of edges 1,...,d-1; for the tree for residual dependence; $fam for copula family label, where $fam=1 for Frank copula for residual dependence conditional on latent; $fam=2 for bivariate Gaussian/normal copula.
`iprint`	indicator for printing of function and gradient

Value

nllk, grad, hess (negative log-likelihood, gradient, Hessian at MLE)

References

Joe (2018). Parsimonious graphical dependence models constructed from vines. Canadian Journal of Statistics 46(4), 532-555.

Examples

```
data(DJ20142016gf)
udat = dj1416gf$uscore
d = ncol(udat)
loadings = c(0.53,0.68,0.68,0.62,0.69,0.58,0.60,0.67,0.73,0.72,
0.64,0.70,0.65,0.69,0.75,0.56,0.56,0.79,0.59,0.66,
0.57,0.60,0.60,0.71,0.57,0.73,0.70,0.56, 0.52,0.61)
# starting values for BB1 that have roughly dependence of bivariate Gaussian
tau = bvn_cpar2tau(loadings)
```

```
par0 = bb1_tau2eqtd(tau)
par0 = par0[,c(1,2)]
par0=c(t(par0))
# from gauss1f1t()
edg1 = c(4,4,4,2,5,10,10,16,9,14,1,3,12,13,8,11,
17,19,4,10,16,16,22,3,4,21,16,23,6)
edg2 = c(6,7,9,10,13,14,15,17,18,19,20,20,20,20,21,21,
21,22,23,23,23,24,25,26,26,27,28,29,30)
pc = c(0.2986594,0.1660604,0.213082,0.2975893,0.2534793,-0.1485211,
0.6179934,0.2207242,0.1877172,0.2625087,0.1414915,-0.1171917,
0.1396517,0.2632831,0.1964068,0.2850865,0.1679997,0.3683564,
-0.1666632,-0.2291848,0.3600637,0.1737616,0.2022859,0.2136862,
0.1948507,0.1731044,0.186075,0.2177455,0.6606208)
# convert above 29 rho parameters from above to Frank parameters with roughly
# the same Spearman rhos, e.g. frank_rhoS2cpar()
print(frank_rhoS2cpar(pc))
parfrk = c(1.87, 1.00, 1.30, 1.86, 1.57, -0.90, 4.67, 1.35, 1.14, 1.63,
0.85, -0.70, 0.84, 1.63, 1.20, 1.78, 1.02, 2.37, -1.01, -1.41,
2.30, 1.05, 1.23, 1.31, 1.19, 1.05, 1.13, 1.33, 5.23)
# parameters for tree 1 (latent) and tree 2 (residual dependence)
gl = gaussLegendre(21)
bffxvar = rep(FALSE,d-1)
ifixed = c(rep(FALSE,2*d),bffxvar)
LB = c(rep(c(0.01,1.01),d),rep(-15,d-1))
UB = c(rep(c(10,10),d),rep(20,d-1))
st_bb1frk = c(par0,parfrk)
dstrbb1frk = list(data=udat,quad=gl, edg1=edg1, edg2=edg2, fam=1)
tem_frk = factor1trvine_nllk(st_bb1frk,dstrbb1frk)
print(tem_frk$fnval)
# -6600.042
ml_bb1frk = posDefHessMinb(st_bb1frk, factor1trvine_nllk, ifixed= ifixed,
dstrbb1frk, LB, UB, mxiter=30, eps=5.e-5, bdd=5, iprint=TRUE)
#1 -6600.042 0.537193
#2 -6645.547 0.06360556
#3 -6645.968 0.0008600067
#4 -6645.968 6.065411e-07
cat("BB1frk: parmin (mle for udata), SEs\n")
print(ml_bb1frk$fnval)
#  -6645.968
print(ml_bb1frk$parmin)
print(sqrt(diag(ml_bb1frk$invh)))
#
dstrbb1gau = list(data=udat,quad=gl, edg1=edg1, edg2=edg2, fam=2)
LB = c(rep(c(0.01,1.01),d),rep(-1,d-1))
UB = c(rep(c(10,10),d),rep(1,d-1))
st_bb1gau = c(par0,pc)
tem_gau = factor1trvine_nllk(st_bb1gau,dstrbb1gau)
print(tem_gau$fnval)
# -6587.514
ml_bb1gau = posDefHessMinb(st_bb1gau, factor1trvine_nllk, ifixed= ifixed,
dstrbb1gau, LB, UB, mxiter=30, eps=5.e-5, bdd=5, iprint=TRUE)
#1 -6587.514 0.1732503
#2 -6621.988 0.03526498
```

```
#3 -6622.375 0.000806789
#4 -6622.375 7.05569e-07
cat("BB1gau: parmin (mle for udata), SEs\n")
print(ml_bb1gau$fnval)
# -6622.375
print(ml_bb1gau$parmin)
print(sqrt(diag(ml_bb1gau$invh)))
```

---

frank_beta2cpar *Frank: Blomqvist's beta to copula parameter*

Description

Frank: Blomqvist's beta to copula parameter, vectorized

Usage

```
frank_beta2cpar(beta, cpar0=0,mxiter=20,eps=1.e-8,iprint=FALSE)
```

Arguments

beta	vector of Blomqvist's beta values, -1<beta<1
cpar0	starting point for Newton-Raphson iterations
mxiter	maximum number of iterations, default 20
eps	tolerance for convergence, default 1.e-8
iprint	print flag for iterations, default FALSE

Details

Solve equation to get cpar given Blomqvist's beta, Newton-Raphson iterations; vectorized input beta is OK, beta=0 fails

Value

vector of Frank copula parameters with the given betas

Examples

```
b = seq(-0.2,0.5,0.1)
frank_beta2cpar(b)
frank_beta2cpar(b,iprint=TRUE)
```

`frank_rhoS2cpar` *Frank: Spearman rho to copula parameter*

Description

Frank: Spearman rho to copula parameter

Usage

```
frank_rhoS2cpar(rho)
```

Arguments

`rho`	vector of Spearman values, -1<rho<1

Value

vector of Frank copula parameters with the given rho

Examples

```
rho = seq(-0.2,0.6,0.1)
frank_rhoS2cpar(rho)
```

`gauss1f1t` *Compute correlation matrix according to 1-factor + 1-truncated vine (residual dependence) model*

Description

Compute correlation matrix according to a 1-factor + 1-truncated vine (for residual dependence) model

Usage

```
gauss1f1t(cormat, start_loading, iter=10, est="mle", plots=TRUE, trace=TRUE)
```

Arguments

`cormat`	dxd correlation matrix
`start_loading`	dx1 loading vector (for latent factor)
`iter`	number of iterations for modified EM
`est`	"mle" or "mom"
`plots`	flag that is TRUE to show plots of EM steps
`trace`	flag that is TRUE to print every 100th integer for iter

Details

A modified EM algorithm is used – first step of the M-step performed either by MLE or by method of moments – the second step assumes the moment estimator has been used

Value

components:loading = final estimate for loading vector; R = correlation matrix (from MLE for 1F1T structure); Psi = vector of residual variances; loadings = matrix where ith row has the ith iteration; Rmats = list of correlation matrices; Rmats[[i]] has the ith iteration; Rstart = starting value of R based on starting values; dists = vector of distance measures as GOF criterion, ith entry for ith iteration; incls = iter x d*(d-1)/2 matrix, ith row for ith iteration columns are whether edge 12, 13, 23, 14, ... (d-1,d) are in residual tree; partcor = dxd matrix of partial correlations given latent variable; loglik = vector of loglik values, ith entry for ith iteration.

References

Brechmann EC and Joe H (2014). Parsimonious parameterization of correlation matrices using truncated vines and factor analysis. Computational Statistics and Data Analysis, 77, 233-251.

Examples

```
library(igraph)
data(DJ20142016gf)
zdat = dj1416gf$zscore # GARCH-filtered returns that have been transformed to N(0,1)
rzmat = cor(zdat)
d = ncol(zdat)
cat("\n1-factor start for 1f1t\n")
fa = factanal(factors=1,covmat=rzmat)
start = c(fa$loading)
# fitting 1Factor 1Truncated vine residual dependence structure
out1f1t = gauss1f1t(rzmat,start_loading=start,iter=20,plots=FALSE)
print(out1f1t$loading)
cat("\nedges for tree of residual dependence\n")
i = 1:d
nn = i*(i-1)/2
niter = 21  # above iteration bound +1
incls = out1f1t$incls[niter,]
for(j in 2:d)
{ for(k in 1:(j-1))
  { if(incls[nn[j-1]+k])
    { cat("edge ", k,j," "); cat(out1f1t$partcor[k,j],"\n") }
  }
}
# extract three columns of this output to use with factor1trvine_nllk
# See example in cop1f1t()
```

`gaussLegendre` *R interface for Gauss-Legendre quadrature*

Description

Gauss-Legendre quadrature nodes and weights

Usage

```
gaussLegendre(nq)
```

Arguments

`nq`	number of quadrature points

Details

links to C code translation of jacobi.f in Stroud and Secrest (1966)

Value

structures with

Examples

```
out = gaussLegendre(15)
# same as statmod::gauss.quad.prob(15,dist="uniform") in library(statmod)
print(sum(out$weights)) # should be 1
print(sum(out$weights*out$nodes)) # should be 0.5  = E(U), U~Uniform(0,1)
print(sum(out$weights*out$nodes^2)) # should be 1/3 = E(U^2)
```

`gumbel_beta2cpar` *Gumbel: Blomqvist's beta to copula parameter*

Description

Gumbel: Blomqvist's beta to copula parameter, vectorized

Usage

```
gumbel_beta2cpar(beta)
```

Arguments

`beta`	vector of Blomqvist's beta values, 0<beta<1

Value

vector of Gumbel copula parameters with the given betas

Examples

```
b = seq(0.1,0.5,0.1)
gumbel_beta2cpar(b)
```

---

gumbel_rhoS2cpar *Gumbel: Spearman rho to copula parameter*

Description

Gumbel: Spearman rho to copula parameter

Usage

```
gumbel_rhoS2cpar(rho)
```

Arguments

`rho`	vector of Spearman values, 0<rho<1

Value

vector of Gumbel copula parameters with the given rho

Examples

```
rho = seq(0.1,0.5,0.1)
gumbel_rhoS2cpar(rho)
```

---

isPosDef *Check if a square symmetric matrix is positive definite*

Description

Check if a square symmetric matrix is positive definite

Usage

```
isPosDef(amat)
```

Arguments

`amat`	symmetric matrix

Value

TRUE if amat is positive definite, FALSE otherwise

Examples

```
a1 = matrix(c(1,.5,.5,1),2,2)
a2 = matrix(c(1,1.5,1.5,1),2,2)
t1 = try(chol(a1))
t2 = try(chol(a2))
print(isPosDef(a1))
print(isPosDef(a2))
```

latentUpdate1factor *Compute new proxies for 1-factor copula based on the mean of observations*

Description

Compute new proxies for 1-factor copula for 1-parameter or 2-parameter linking copulas

Usage

```
latentUpdate1factor(cpar_est,udata,nq,family)
```

Arguments

cpar_est	estimated parameters (based on complete likelihood with latent variables known or estimated)
udata	nxd matrix of data in (0,1)
nq	number of nodes for Gaussian-Legendre quadrature
family	vector of code for d linking copula families (choices 1,2,4,5,7,10,14,17,20)

Value

latent_est: proxies as estimates of latent variables

Examples

```
# See examples in onefactorEstWithProxy()
```

`latentUpdate1factor1` *Compute new proxies for 1-factor copula based on the mean of observations*

Description

Compute new proxies for 1-factor copula for 1-parameter linking copulas

Usage

```
latentUpdate1factor1(cpar_est,udata,nq,family)
```

Arguments

`cpar_est`	estimated parameters (based on complete likelihood with latent variables known or estimated)
`udata`	nxd matrix of data in (0,1)
`nq`	number of nodes for Gaussian-Legendre quadrature
`family`	vector of code for d linking copula families (choices 1,4,5,14)

Value

latent_est: proxies as estimates of latent variables

Examples

```
# See examples in onefactorEstWithProxy()
```

`latentUpdateBifactor` *Conditional expectation proxies for bi-factor copula models with linking copulas in different copula families*

Description

Conditional expectation proxies for bi-factor copula models with linking copulas in different copula families

Usage

```
latentUpdateBifactor(udata,cparvec,grsize,family, nq)
```

Arguments

`udata`	nxd matrix with valies in (0,1)
`cparvec`	parameters for linking copulas; order is global_par1, global_par2, local_par1, local_par2
`grsize`	group size vector of length mgrp
`family`	codes for linking copula (VineCopula)
`nq`	number of Gaussian-Legendre points

Value

v0: proxies of the global latent variable andvg: proxies of the local latent variables

Examples

```
# See example in bifactorEstWithProxy()
```

`ml1factor` *max likelihood (min negative log-likelihood) for 1-factor copula model*

Description

min negative log-likelihood for 1-factor copula model

Usage

```
ml1factor(nq,start,udata,dcop,LB=0,UB=1.e2,prlevel=0,mxiter=100)
```

Arguments

`nq`	number of quadrature points
`start`	starting point (d-vector or m*d vector, e.g. 2*d vector for BB1)
`udata`	nxd matrix of uniform scores
`dcop`	name of function for a bivariate copula density (common for all variables)
`LB`	lower bound on parameters (scalar or same dimension as start)
`UB`	upper bound on parameters (scalar or same dimension as start)
`prlevel`	printlevel for nlm()
`mxiter`	maximum number of iteration for nlm()

Value

nlm object with minimum, estimate, hessian at MLE

Examples

```
# See example in r1factor()
```

`ml1factor_f90` *min negative log-likelihood for 1-factor copula with nlm()*

Description

min negative log-likelihood for 1-factor copula with nlm()

Usage

```
ml1factor_f90(nq,start,udata,copname,LB=0,UB=40,ihess=FALSE,prlevel=0,
  mxiter=100,nu=3)
```

Arguments

`nq`	number of quadrature points
`start`	starting point (d-vector or m*d vector, e.g. 2*d vector for BB1)
`udata`	nxd matrix of uniform scores
`copname`	name of copula family such as "gumbel", "frank", "bb1", "t" (copname common for all variables)
`LB`	lower bound on parameters (scalar or same dimension as start)
`UB`	upper bound on parameters (scalar or same dimension as start)
`ihess`	flag for hessian option in nlm()
`prlevel`	printlevel for nlm()
`mxiter`	max number of iterations for nlm()
`nu`	degree of freedom parameter if copname ="t"

Value

MLE as nlm object (estimate, Hessian, SEs, nllk)

Examples

```
# See example in r1factor()
```

ml1factor_v2 — *min negative log-likelihood for 1-factor copula model (some parameters can be fixed)*

Description

min negative log-likelihood (nllk) for 1-factor copula model (some parameters can be fixed)

Usage

```
ml1factor_v2(nq,start,ifixed,udata,dcop,LB=0,UB=1.e2,prlevel=0,mxiter=100)
```

Arguments

`nq`	number of quadrature points
`start`	starting point (d-vector or m*d vector, e.g. 2*d vector for BB1)
`ifixed`	vector of length(param) of True/False, such that ifixed[i]=TRUE iff param[i] is fixed at the given value start[j]
`udata`	nxd matrix of uniform scores
`dcop`	name of function for a bivariate copula density (common for all variables)
`LB`	lower bound on parameters (scalar or same dimension as start)
`UB`	upper bound on parameters (scalar or same dimension as start)
`prlevel`	printlevel for nlm()
`mxiter`	maximum number of iteration for nlm()

Value

nlm object with nllk value, estimate, hessian at MLE

mvtBifact — *MLE for multivariate normal/t with a bi-factor or nested factor correlation structure*

Description

MLE for the bi-factor or nested factor structure for multivariate normal/t

Usage

```
mvtBifact(tdata,start,grsize,df,prlevel=2,model="bifactor",mxiter=100)
```

Arguments

`tdata`	nxd matrix of t-scores or z-scores
`start`	vector of length 2*d with starting values of partial correlations values for correlations of observed Z_[gj] and common latent V_0 go first, then partial correlations of Z_[gj] and V_g given V_0 (j in group g)
`grsize`	vector of group sizes for bi-factor model
`df`	degrees of freedom parameter >0
`prlevel`	print.level for nlm()
`model`	"bifactor" or "nestfactor" nested-factor is reduced model with fewer parameters
`mxiter`	maximum number of iterations for nlm()

Details

Note the minimum nllk can be the same for different parameter vectors if some group size values are 1 or 2.

Value

nlm object with ($code,$estimate,$gradient,$iterations,$minimum)

Examples

```
data(rainstorm)
udat = rainstorm$uprecip
d = ncol(udat)
grsize = rainstorm$grsize
df = 10
tdata = qt(udat,df)
bif = mvtBifact(tdata, c(rep(0.8,d),rep(0.2,d)), grsize, df=df,
prlevel=1, model="bifactor", mxiter=100)
#
# nested-factor: parameters for group latent linked to global latent
# come in the first tree of the 2-truncated vine.
nestf = mvtBifact(tdata, c(0.7,0.7,0.7,rep(0.85,d)), grsize, df=df,
prlevel=1, model="nestfactor", mxiter=100)
# doesn't converge properly, group2 latent matches global latent
#
st1 = rep(0.7,d)
out1t = mvtPfact(tdata,st1,pfact=1,df=df,prlevel=1)
st2 = rep(0.7,2*d)
out2t = mvtPfact(tdata,st2,pfact=2,df=df,prlevel=1)
st3 = c(rep(0.7,grsize[1]),rep(0.1,d),rep(0.7,grsize[2]),rep(0.1,d),rep(0.7,grsize[3]))
out3t = mvtPfact(tdata,st3,pfact=3,df=df,prlevel=1)
cat(bif$minimum, nestf$minimum, out1t$minimum, out2t$minimum, out3t$minimum,"\n")
```

`mvtBifact_nllk` *negative log-likelihood for the bi-factor Gaussian/t model*

Description

negative log-likelihood in the bi-factor Gaussian or t model

Usage

```
mvtBifact_nllk(rhovec,grsize,tdata,df)
```

Arguments

`rhovec`	vector of length d*2 for partial correlation representation of loadings, first d correlations with common factor, then partial correlations with group factor given common factor
`grsize`	vector of group sizes for bi-factor model
`tdata`	nxd data set of scores for t df (e.g., qt(u,df))
`df`	degree of freedom parameter (positive)

Value

negative log-likelihood (nllk) of copula for mvt bi-factor model

`mvtPfact` *MLE in a MVt model with a p-factor correlation structure*

Description

MLE in a MVt model with a p-factor correlation structure

Usage

```
mvtPfact(tdata,start,pfact,df,prlevel=0,mxiter=100)
```

Arguments

`tdata`	nxd matrix of t-scores
`start`	vector of length p*d with starting values for partial correlation representation of loadings (algberaically independent)
`pfact`	number of factors (such as 1,2,3,...)
`df`	degree of freedom parameter >0
`prlevel`	print.level for nlm()
`mxiter`	maximum number of iterations for nlm()

Value

nlm object with ($code,$estimate,$gradient,$iterations,$minimum) Note the minimum nllk can be the same for different parameter vectors because of invariance of the loading matrix to rotations

Examples

```
# See example in mvtBifact()
```

---

mvtPfact_nllk *negative log-likelihood for the p-factor Gaussian/t model*

Description

negative log-likelihood in the p-factor Gaussian or t model

Usage

```
mvtPfact_nllk(rhvec,tdata,df)
```

Arguments

rhvec	vector of length d*p with partial correlation representation of loadings
tdata	nxd data set of t(df)-scores
df	degree of freedom parameter >0

Value

negative log-likelihood of copula for mvt p-factor model

---

nestfactorcop_nllk *negative log-likelihoods of nested factor structured factor copula and derivatives computed in f90 for input to posDefHessMinb*

Description

negative log-likelihoods of nested factor structured factor copula and derivatives

Usage

```
nestfactorcop_nllk(param,dstruct,iprfn=FALSE)
```

Arguments

`param`	parameter vector; parameters for copulas linking U_ij and V_j go *at the end* (i's with j=1 then j=2 etc) parameters for copulas linking V_j and V_0 go *first* (j=1,2 etc). For BB1 linking copulas for the global latent, the order is theta1,..,theta[d],delta1, ...,delta[d]; V_0 is the global/common latent variable that loads on all variables; V_j is a latent variable that loads only for variables in group j (by group g=1,2,..,mgrp etc).
`dstruct`	list with data set $data, copula name $copname, $quad is a Gauss-Legendre quadrature object, $repar is a flag for reparametrization (for Gumbel, BB1), $nu is a scalar or 2-vector for degree of freedom parameter(s), $grsize is a vector with group sizes; if dstruct$pdf == 1 the function evaluates nllk only (and returns zero gradient and hessian). Options for copname are: frank, gumbel, frankgumbel, frankbb1, gumbelbb1, tbb1, t. For tbbb1, nu is a scalar. For t, nu is a 2-vector.
`iprfn`	indicator for printing of function and gradient (within Newton-Raphson iterations)

Value

nllk, grad, hess (gradient and hessian included)

Examples

```
grsize = c(4,4,3)
d = sum(grsize)
n = 500
mgrp = length(grsize)
set.seed(222)
par_nest = c(rep(1.7,3),seq(1.7,3.7,0.2))
udat_obj = rnestfactor(n,grsize,cop=4,par_nest)
udat = udat_obj$data
summary(udat_obj$v0)
summary(udat_obj$vg)
zdat = qnorm(udat)
rmat  = cor(zdat)
round(cor(zdat),3)
# run oblique_fa to get oblqiue factor correlation matrix
obfa = oblique_fa(grsize,start=c(rep(0.8,d),rep(0.5,mgrp)),cormat=rmat,n=n,prlevel=0)
loading1 = rowSums(obfa$loadings)
corlat = obfa$cor_lat # correlations of group latent variables
fa1 = factanal(covmat=corlat,factors=1)
loadlat = c(fa1$loadings)
print(loadlat)
# starting values for different cases
# convert loading/latcor to Frank, Gumbel and BB1 parameters etc
start_frk1 = frank_rhoS2cpar(loading1)
start_frk0 = frank_rhoS2cpar(loadlat)
start_frk = c(start_frk0,start_frk1)
start_gum1 = gumbel_rhoS2cpar(loading1)
start_gum0 = gumbel_rhoS2cpar(loadlat)
```

```
start_gum = c(start_gum0,start_gum1)
start_frkgum = c(start_frk0,start_gum1)
start_tnu = c(loadlat,loading1)
tau = bvn_cpar2tau(loading1)
start_bb1 = bb1_tau2eqtd(tau)
start_bb1 = c(start_bb1[,1:2]) # all thetas and then all deltas
start_frkbb1 = c(start_frk0,start_bb1)
start_gumbb1 = c(start_gum0,start_bb1)
start_tnubb1 = c(loadlat,start_bb1)
#
gl = gaussLegendre(25)
npar = mgrp+d
dstrfrk = list(data=udat,copname="frank",quad=gl,repar=0,grsize=grsize)
out = nestfactorcop_nllk(start_frk, dstrfrk)
print(out$fnval)
print(out$grad)
ml_frk = posDefHessMinb(rep(3,npar),nestfactorcop_nllk, ifixed=rep(FALSE,npar),
dstrfrk, LB=rep(-20,npar), UB=rep(30,npar), mxiter=30, eps=5.e-5, iprint=TRUE)
dstrgum = list(data=udat,copname="gumbel",quad=gl,repar=0,grsize=grsize)
ml_gum = posDefHessMinb(start_gum,nestfactorcop_nllk, ifixed=rep(FALSE,npar),
dstrgum, LB=rep(1,npar), UB=rep(20,npar), mxiter=30, eps=5.e-5, iprint=TRUE)
dstrfrkgum = list(data=udat,copname="frankgumbel",quad=gl,repar=0,grsize=grsize)
ml_frkgum = posDefHessMinb(start_frkgum,nestfactorcop_nllk, ifixed=rep(FALSE,npar),
dstrfrkgum, LB=c(rep(-20,mgrp),rep(1,d)), UB=rep(25,npar), mxiter=30,
eps=5.e-5, iprint=TRUE)
dstrtnu = list(data=udat,copname="t",quad=gl,repar=0,grsize=grsize, nu=c(10,20))
ml_tnu = posDefHessMinb(start_tnu,nestfactorcop_nllk, ifixed=rep(FALSE,npar),
dstrtnu, LB=c(rep(-1,npar)), UB=rep(1,npar), mxiter=30, eps=5.e-5, iprint=TRUE)
# diverges with parameters approaching 1
#
npar2 = mgrp+2*d
dstrfrkbb1 = list(data=udat,copname="frankbb1",quad=gl,repar=0,grsize=grsize)
out = nestfactorcop_nllk(start_frkbb1,dstrfrkbb1)
print(out$fnval)
print(out$grad)
ml_frkbb1 = posDefHessMinb(start_frkbb1,nestfactorcop_nllk, ifixed=rep(FALSE,npar2),
dstrfrkbb1, LB=c(rep(-20,mgrp),rep(0,d),rep(1,d)), UB=rep(20,npar2),
mxiter=30, eps=5.e-5, iprint=TRUE)
dstrgumbb1 = list(data=udat,copname="gumbelbb1",quad=gl,repar=0,grsize=grsize)
ml_gumbb1 = posDefHessMinb(start_gumbb1,nestfactorcop_nllk, ifixed=rep(FALSE,npar2),
dstrgumbb1, LB=c(rep(1,mgrp),rep(0,d),rep(1,d)), UB=rep(20,npar2),
mxiter=30, eps=5.e-5, iprint=TRUE)
dstrtnubb1 = list(data=udat,copname="tbb1",quad=gl,repar=0,grsize=grsize, nu=20)
ml_tnubb1 = posDefHessMinb(start_tnubb1,nestfactorcop_nllk, ifixed=rep(FALSE,npar2),
dstrtnubb1, LB=c(rep(-1,mgrp),rep(0,d),rep(1,d)), UB=c(rep(1,mgrp),rep(20,2*d)),
mxiter=30, eps=5.e-5, iprint=TRUE)
#
# compare nllk and number of iterations
cat(ml_frk$fnval, ml_gum$fnval, ml_frkgum$fnval, ml_tnu$fnval,
ml_frkbb1$fnval, ml_gumbb1$fnval, ml_tnubb1$fnval, "\n")
# -1438.187 -1760.851 -1725.286 -5555.964 -1729.274 -1764.83 -1746.629
cat(ml_frk$iter, ml_gum$iter, ml_frkgum$iter, ml_tnu$iter,
ml_frkbb1$iter, ml_gumbb1$iter, ml_tnubb1$iter, "\n")
```

```
# 7 8 6 23 15 16 16
# nested-factor t(10)/t(20) failed because some parameters approached the
# upper bound of 1 in which case the numerical integration is inaccurate.
```

nscore *Rank-based normal scores transform*

Description

Rank-based normal scores transform

Usage

```
nscore(data)
```

Arguments

data	dataframe or matrix, or vector, of reals

Value

matrix or vector of normal scores

oblique_fa *Gaussian oblique factor structure correlation matrix*

Description

MLE of parameters in the Gaussian oblique factor model for d variables and m groups,

Usage

```
oblique_fa(grsize, start, data=1, cormat=NULL,
                          n=100, prlevel=0, mxiter=100)
```

Arguments

grsize	vector of group sizes (variables ordered by group)
start	starting point should have dimension d+m*(m-1)/2
data	nxd data set to compute the correlation matrix if cormat not given
cormat	dxd empirical correlation matrix (of normal scores)
n	sample size, if available
prlevel	print.level for nlm()
mxiter	maximum number of iterations for nlm()

Value

list with nllk: negative log-likelihood; rhovec: the estimated mle; loadings: loading matrix; cor_lat: correlation matrix of the latent variables; Rmod: the correlation matrix with optimized parameters.

Examples

```
 # See example in bifact_fa() for a comparison with a data example
# Simpler example below
rhpar = c(0.81,0.84,0.84, 0.54,0.57,0.49, 0.51,0.54,0.55,0.70,  0.53,0.56,0.53,0.67,0.70)
cormat = corvec2mat(rhpar)
grsize = c(3,3)
mgrp = length(grsize)
d = sum(grsize)
start = rep(0.7,d+mgrp*(mgrp-1)/2)
res = oblique_fa(grsize, start, cormat=cormat, n=100, prlevel=1)
# iteration = 18
# Parameter:
# [1] 0.9005171 0.9064707 0.9270258 0.8202611 0.8480912 0.8243349 0.7039589
# Function Value
# [1] 622.5533
# Gradient:
# [1]  1.796252e-05  1.464286e-04  9.424639e-05 -8.344614e-05 -9.640644e-05
# [6] -9.799805e-05 -1.705303e-05
#
# Relative gradient close to zero.
# Current iterate is probably solution.
```

---

oblique_grad_fa *Gaussian oblique factor structure correlation matrix*

Description

MLE of parameters in the Gaussian oblique factor model for d variables and m groups,

Usage

```
oblique_grad_fa(grsize, start, data=1, cormat=NULL,
                               n=100, prlevel=0, mxiter=100)
```

Arguments

grsize	vector of group sizes (variables ordered by group)
start	starting vector of length d + m*(m-1)/2; d loading parameters followed by m*(m-1)/2 entries in correlation matrix of latent variables (lower triangle by row)
data	n x d data set to compute the correlation matrix if correlation matrix (cormat) not given

`cormat`	dxd (empirical) correlation matrix of normal scores
`n`	sample size, if available
`prlevel`	print.level for nlm()
`mxiter`	maximum number of iterations for nlm()

Value

list with nllk: negative log-likeilihood;rhovec: the estimated mle; loadings: loading matrix; cor_lat: correlation matrix of the latent variables; Rmod: the correlation matrix with optimized parameters.

`oblique_grad_nllk` *log-likelihood Gaussian oblique factor structure correlation matrix*

Description

log-likelihood Gaussian oblique factor structure correlation matrix with gradient for d variables and m groups,

Usage

```
oblique_grad_nllk(theta, grsize, Robs, nsize=100)
```

Arguments

`theta`	vector of length d + m*(m-1)/2; d loading parameters followed by m*(m-1)/2 entries in correlation matrix of latent variables (lower triangle by row)
`grsize`	vector of group sizes (variables ordered by group)
`Robs`	dxd empirical correlation matrix
`nsize`	sample size if available

Value

negative log-likelihood and gradient for Gaussian p-factor model

`oblique_nllk` *log-likelihood Gaussian oblique factor structure correlation matrix*

Description

negative log-likelihood of the Gaussian oblique factor model for d variables and m groups,

Usage

```
oblique_nllk(theta, grsize, Robs, nsize=100)
```

Arguments

`theta`	vector of length d + m*(m-1)/2; d loading parameters followed by m*(m-1)/2 entries in correlation matrix of latent variables (lower triangle by row)
`grsize`	vector of group sizes (variables ordered by group)
`Robs`	dxd (empirical) correlation matrix of normal scores
`nsize`	sample size used to get Robs if available

Value

negative log-likelihood value of the oblique Gaussian factor modelwith fixed group size at MLE

Examples

```
 rhpar = c(0.81,0.84,0.84, 0.54,0.57,0.49, 0.51,0.54,0.55,0.70, 0.53,0.56,0.53,0.67,0.70)
cormat = corvec2mat(rhpar)
print(cormat)
#    [,1] [,2] [,3] [,4] [,5] [,6]
#[1,] 1.00 0.81 0.84 0.54 0.51 0.53
#[2,] 0.81 1.00 0.84 0.57 0.54 0.56
#[3,] 0.84 0.84 1.00 0.49 0.55 0.53
#[4,] 0.54 0.57 0.49 1.00 0.70 0.67
#[5,] 0.51 0.54 0.55 0.70 1.00 0.70
#[6,] 0.53 0.56 0.53 0.67 0.70 1.00
grsize = c(3,3)
mgrp = length(grsize)
d = sum(grsize)
theta = c(rep(0.3,d+mgrp*(mgrp-1)/2))
ml_obl = oblique_nllk(theta=theta, grsize, Robs=cormat)
print(ml_obl)
# 806.7432
```

`oblique_par2load` *oblique factor correlation structure for d variables and m groups*

Description

For oblique factor correlation structure for d variables and m groups, convert the vector of parameters theta into the loading matrix and correlation matrix of latent variables The variables are assumed ordered by group.

Usage

```
oblique_par2load(theta,grsize)
```

Arguments

`theta`	vector of length d + m*(m-1)/2; d loading parameters followed by m*(m-1)/2 entries in correlation matrix of latent variables (lower triangle by row)
`grsize`	vector of group sizes (variables ordered by group)

Value

loadings: loading matrix; cor_lat: correlation matrix of latent variables; Rmod: correlation matrix based on theta for oblique factor model

Examples

```
 theta = c(0.6,0.7,0.8,0.7,0.6,0.5,0.5)
oblique_par2load(theta,grsize=c(3,3))
#$loadings
#[,1] [,2]
#[1,]  0.6  0.0
#[2,]  0.7  0.0
#[3,]  0.8  0.0
#[4,]  0.0  0.7
#[5,]  0.0  0.6
#[6,]  0.0  0.5
#
#$cor_lat
#[,1] [,2]
#[1,]  1.0  0.5
#[2,]  0.5  1.0
#
#$Rmod
#[,1]  [,2] [,3]  [,4] [,5]  [,6]
#[1,] 1.00 0.420 0.48 0.210 0.18 0.150
#[2,] 0.42 1.000 0.56 0.245 0.21 0.175
#[3,] 0.48 0.560 1.00 0.280 0.24 0.200
#[4,] 0.21 0.245 0.28 1.000 0.42 0.350
#[5,] 0.18 0.210 0.24 0.420 1.00 0.300
#[6,] 0.15 0.175 0.20 0.350 0.30 1.000
```

oblique_pp_par2load *oblique factor correlation structure for d variables and m groups include determinant and inverse*

Description

For oblique factor correlation structure for d variables and m groups, convert the vector of parameters theta into the loading matrix and correlation matrix of latent variables The variables are assumed ordered by group.

Usage

```
oblique_pp_par2load(theta,grsize, icheck=FALSE)
```

Arguments

theta	vector of length d + m*(m-1)/2; d loading parameters followed by m*(m-1)/2 entries in correlation matrix of latent variables (lower triangle by row)
grsize	vector of group sizes (variables ordered by group)
icheck	flag, if TRUE checks are made

Value

loadings: loading matrix; cor_lat: correlation matrix of latent variables; Rmod: correlation matrix based on theta for oblique factor model

Examples

```
 theta = c(0.6,0.7,0.8,0.7,0.6,0.5,0.5)
oblique_pp_par2load(theta,grsize=c(3,3))
#$loadings
#[,1] [,2]
#[1,]  0.6  0.0
#[2,]  0.7  0.0
#[3,]  0.8  0.0
#[4,]  0.0  0.7
#[5,]  0.0  0.6
#[6,]  0.0  0.5
#
#$cor_lat
#[,1] [,2]
#[1,]  1.0  0.5
#[2,]  0.5  1.0
#
#$Rmod
#[,1]  [,2] [,3]  [,4] [,5]  [,6]
#[1,] 1.00 0.420 0.48 0.210 0.18 0.150
#[2,] 0.42 1.000 0.56 0.245 0.21 0.175
#[3,] 0.48 0.560 1.00 0.280 0.24 0.200
```

```
#[4,] 0.21 0.245 0.28 1.000 0.42 0.350
#[5,] 0.18 0.210 0.24 0.420 1.00 0.300
#[6,] 0.15 0.175 0.20 0.350 0.30 1.000
```

onefactorcop_nllk *negative log-likelihood of 1-factor copula for input to posDefHessMin and posDefHessMinb*

Description

negative log-likelihood (nllk) of 1-factor copula for input to posDefHessMin and posDefHessMinb

Usage

```
onefactorcop_nllk(param,dstruct,iprfn=FALSE)
```

Arguments

param	parameter vector
dstruct	list with data set $data, copula name $copname, $quad is list with quadrature weights and nodes, $repar is code for reparametrization (for Gumbel, BB1), $nu is positive degree of freedom parameter (for t) (linking copula is common for all variables). Options for copname are: frank, gumbel, bb1, t. For reflected gumbel or bb1, use something like dstruct$dat = 1-udata
iprfn	print flag for function and gradient (within Newton-Raphson) iterations) for BB1, param is 2*d-vector with th1,de1,th2,de2,...

Details

linked to Fortran 90 code for speed

Value

nllk, grad (gradient), hess (hessian) at MLE

Examples

```
cpar_gum = seq(1.9,3.7,0.2)
d = length(cpar_gum)
n = 300
param = c(rbind(cpar_gum,rep(0,d)))  # second par2 is 0 for VineCopula
set.seed(111)
gum_obj = r1factor(n,d,param,famvec=rep(4,d)) # uses VineCopula
udat = gum_obj$udata
zdat = qnorm(udat)
rmat = cor(zdat)
print(round(rmat,3))
# run factanal to get loading (rho in normal scale close to Spearman rho)
```

```
fa1 = factanal(covmat=rmat,factors=1)
loadings = c(fa1$loading)
# convert loadings to Frank, Gumbel and BB1 parameters
start_frk = frank_rhoS2cpar(loadings)
start_gum = gumbel_rhoS2cpar(loadings)
tau = bvn_cpar2tau(loadings)
start_bb1 = bb1_tau2eqtd(tau)
start_bb1 = c(t(start_bb1[,1:2]))
gl = gaussLegendre(25)
dstrfrk1 = list(copname="frank",data=udat,quad=gl,repar=0, pdf=1)
dstrfrk = list(copname="frank",data=udat,quad=gl,repar=0, pdf=0)
obj1 = onefactorcop_nllk(start_frk,dstrfrk1) #nllk only
obj = onefactorcop_nllk(start_frk,dstrfrk) # nllk, grad, hess
print(obj1$fnval)
print(obj$grad)
#
ml_frk = posDefHessMinb(start_frk,onefactorcop_nllk,ifixed=rep(FALSE,d),
  dstruct=dstrfrk, LB=rep(-30,d),UB=rep(30,d),iprint=TRUE,eps=1.e-5)
dstrgum = list(copname="gumbel",data=udat,quad=gl,repar=0)
ml_gum = posDefHessMinb(start_gum,onefactorcop_nllk,ifixed=rep(FALSE,d),
  dstruct=dstrgum, LB=rep(-30,d),UB=rep(30,d),iprint=TRUE,eps=1.e-5)
dstrgumr = list(copname="gumbel",data=1-udat,quad=gl,repar=0)
ml_gumr = posDefHessMinb(start_gum,onefactorcop_nllk,ifixed=rep(FALSE,d),
  dstruct=dstrgumr, LB=rep(-30,d),UB=rep(30,d),iprint=TRUE,eps=1.e-5)
dstrtnu = list(copname="t",data=udat,quad=gl,repar=0,nu=10)
ml_tnu = posDefHessMinb(loadings,onefactorcop_nllk,ifixed=rep(FALSE,d),
  dstruct=dstrtnu, LB=rep(-1,d),UB=rep(1,d),iprint=TRUE,eps=1.e-5)
dstrbb1 = list(copname="bb1",data=udat,quad=gl,repar=0)
ml_bb1 = posDefHessMinb(start_bb1,onefactorcop_nllk,ifixed=rep(FALSE,2*d),
  dstruct=dstrbb1, LB=rep(c(0,1),d),UB=rep(20,2*d),iprint=TRUE,eps=1.e-5)
dstrbb1r = list(copname="bb1",data=1-udat,quad=gl,repar=0)
ml_bb1r = posDefHessMinb(start_bb1,onefactorcop_nllk,ifixed=rep(FALSE,2*d),
  dstruct=dstrbb1r, LB=rep(c(0,1),d),UB=rep(20,2*d),iprint=TRUE,eps=1.e-5)
cat(ml_frk$fnval, ml_gum$fnval, ml_gumr$fnval, ml_tnu$fnval, ml_bb1$fnval, ml_bb1r$fnval, "\n")
# -1342.936 -1560.391 -1198.837 -1454.907 -1560.45 -1549.935
cat(ml_frk$iter, ml_gum$iter, ml_gumr$iter, ml_tnu$iter, ml_bb1$iter, ml_bb1r$iter, "\n")
# 4 4 5 4 16 6
```

onefactorEstWithProxy *Parameter estimation for 1-factor copula with estimated latent variables using VineCopula::BiCopSeelct*

Description

Parameter estimation for 1-factor copula with estimated latent variables

Usage

```
onefactorEstWithProxy(udata,vlatent, famset, iprint=FALSE)
```

Arguments

`udata`	nxd matrix with values in (0,1)
`vlatent`	vector is estimated latent variables (or test with known values)
`famset`	2*d vector of codes for copula families for d global linking copulas and d group-based linking copulas, using those from VineCopula: current choices to cover a range of tail behavior are: 1 = Gaussian/normal; 2 = t; 4 = Gumbel; 5 = Frank; 7 = BB1; 10 = BB8; 14 = survival Gumbel; 17 = survival BB1; 20 = survival BB8.
`iprint`	if TRUE print intermediate results

Details

It is best if variables have been oriented to be positively related to the latent variable

Value

list with fam = d-vector of family codes chosen via BiCopSelect;par1 = d-vector; par2 = d-vector of parameters for the selected copula families in the 1-truncated vine rooted at the latent variable,

References

1. Krupskii P and Joe H (2013). Factor copula models for multivariate data. Journal of Multivariate Analysis, 120, 85-101. 2. Fan X and Joe H (2024). High-dimensional factor copula models with estimation of latent variables Journal of Multivariate Analysis, 201, 105263.

Examples

```
## Not run:
# simulate data from 1-factor model with all Frank copulas
n = 500
d = 40
set.seed(20)
cpar = runif(d,4.2,18.5)
param = c(rbind(cpar,rep(0,d))) #Kendall's tau 0.4 to 0.8
data = r1factor(n,d,param,fam=rep(5,d))
vlat = data$vlatent # latent variables
udata = data$udata
proxyMean = uscore(apply(udata,1,mean)) # mean proxy
# RMSE of estimated latent variables
print(sqrt(mean((proxyMean-vlat)^2)))
# first estimation of 1-factor copula parameters
# allow for Frank, gaussian, t linking copulas
est1 = onefactorEstWithProxy(udata,proxyMean, famset=c(1,2,5))
print(est1$fam) # check choices , all 5s (Frank) in this case
print(est1$par1)
# estimation with only Frank copula as choice
est0 = onefactorEstWithProxy(udata,proxyMean, famset=c(5))
print(summary(abs(est0$par1-cpar)))  # same as $est1$par1
# improved conditional expectation proxies
# latentUpdate1factor allows for estimated linking copula with 2-parameters
# latentUpdate1factor1 can be used if estimated linking copulas all have par2=0
condExpProxy = latentUpdate1factor(c(rbind(est1$par1,est1$par2)),
udata=udata,nq=25,family=rep(5,d))
# improved estimation of 1-factor copula parameters
est2 = onefactorEstWithProxy(udata,condExpProxy, famset=est1$fam)
print(est2$par1)
# simple version of update for 1-parameter linking copulas
condExpProxy1 = latentUpdate1factor1(est0$par1,
udata=udata,nq=25,family=rep(5,d))
summary(condExpProxy-condExpProxy1)  # 0 because family was chosen as 5
print(summary(abs(est2$par1-cpar)))
# rmse of estimated latent variables
print(sqrt(mean((condExpProxy-vlat)^2)))
# smaller rmse than initial proxies

## End(Not run)
```

pcor2load *Partial correlation representation to loadings for p-factor*

Description

Partial correlation representation to loadings for p-factor

Usage

```
pcor2load(rhomat)
```

Arguments

rhomat	dxp matrix with correlations for factor 1 in column 1, and partial correlations with factor k given previous factors in column k

Details

Partial correlation representation to loadings for p-factor

Value

loading matrix

Examples

```
grsize = c(5,4,3)
# bi-factor parameters: 13 correlations with global latent and then
# 5 partial correlations for group1 latent given global,
# 4 partial correlations for group2 latent given global,
# 3 partial correlations for group3 latent given global
par_bifact = c(0.84,0.63,0.58,0.78,0.79,  0.87,0.80,0.74,0.71,  0.83,0.77,0.80,
0.67,0.58,0.15,0.70,0.47,  0.32,0.27,0.73,0.19,  0.35,0.23,0.53)
```

```
mgrp = length(grsize)
d = sum(grsize)
pcmat = matrix(0,d,mgrp+1) # bi-factor structure has mgrp+1 factors
pcmat[,1] = par_bifact[1:d]
iend = cumsum(grsize)
ibeg = iend+1; ibeg = c(1,1+iend[-mgrp])
for(g in 1:mgrp)
{ pcmat[ibeg[g]:iend[g],g+1] = par_bifact[d+ibeg[g]:iend[g]] }
print(pcmat)
aload = pcor2load(pcmat)
print(aload)
```

`pfactor_fa` *Gaussian p-factor structure correlation matrix*

Description

Gaussian p-factor structure correlation matrix with quasi-Newton

Usage

```
pfactor_fa(factors,start,data=1,cormat=NULL,n=100,prlevel=0,mxiter=100)
```

Arguments

`factors`	p = #factors
`start`	starting point should have dimension 2*d
`data`	nsize x d data set to compute the correlation matrix if correlation matrix (cormat) not given
`cormat`	dxd empirical correlation matrix
`n`	sample size
`prlevel`	print.level for nlm()
`mxiter`	maximum number of iterations for nlm()

Value

a list with $nllk, $rhmat = dxp matrix of partial correlations, $loading = dxp loading matrix after varimax, $rotmat = pxp rotation matrix used by varimax

Examples

```
# See example in bifactor_fa()
```

`pfactor_nllk` *log-likelihood Gaussian p-factor structure correlation matrix*

Description

log-likelihood Gaussian p-factor structure correlation matrix with gradient

Usage

```
pfactor_nllk(rhvec,Robs,nsize)
```

Arguments

`rhvec`	vector of length d*p with partial corr representation of loadings
`Robs`	dxd empirical correlation matrix
`nsize`	sample size

Value

negative log-likelihood and gradient for Gaussian p-factor model

`posDefHessMin` *Minimization with modified Newton-Raphson iterations, Hessian is modified to be positive definite at each step. Algorithm and code produced by Pavel Krupskii (2013) see PhD thesis Krupskii (2014), UBC and Section 6.2 of # Joe (2014) Dependence Models with Copulas. Chapman&Hall/CRC*

Description

modified Newton-Raphson minimization with positive Hessian

Usage

```
posDefHessMin(param,objfn,dstruct,LB,UB,mxiter=30,eps=1.e-6,bdd=5,iprint=FALSE)
```

Arguments

`param`	starting point for minimization
`objfn`	function to be minimized with gradient and Hessian
`dstruct`	list with data set and other variables used by objfn
`LB`	lower bound vector
`UB`	upper bound vector
`mxiter`	max number of iterations

`eps`	tolerance for Newton-Raphson iterations
`bdd`	bound on difference of 2 consecutive iterations (useful is starting point is far from solution and func is far from convex)
`iprint`	control on amount of printing, FALSE for no printing of iterations and TRUE for printing x^(k) on each iteration.

Value

list withfnval = function value at minimum; parmin = param for minimum; invh = inverse Hessian; iconv = 1 if converged, -1 for a boundary point, 0 otherwise; iter = number of iterations.

Examples

```
# See examples in onefactorcop_nllk(), bifactorcop_nllk(), nestfactorcop_nllk()
```

`posDefHessMinb` *Version with ifixed as argument*

Description

modified Newton-Raphson minimization with positive Hessian

Usage

```
posDefHessMinb(param,objfn,ifixed,dstruct,LB,UB,mxiter=30,eps=1.e-6,
  bdd=5,iprint=FALSE)
```

Arguments

`param`	starting point for minimization
`objfn`	function to be minimized with gradient and Hessian
`ifixed`	vector of length(param) of TRUE/FALSE, such that ifixed[i]=TRUE iff param[i] is fixed at the given value
`dstruct`	list with data set and other variables used by objfn
`LB`	lower bound vector
`UB`	upper bound vector
`mxiter`	max number of iterations
`eps`	tolerance for Newton-Raphson iterations
`bdd`	bound on difference of 2 consecutive iterations (useful is starting point is far from solution and func is far from convex)
`iprint`	control on amount of printing, FALSE for no printing of iterations and TRUE for printing x^(k) on each iteration.

Value

list withfnval = function value at minimum; parmin = param for minimum; invh = inverse Hessian; iconv = 1 if converged, -1 for a boundary point, 0 otherwise; iter = number of iterations.

Examples

```
# See examples in onefactorcop_nllk(), bifactorcop_nllk(), nestfactorcop_nllk()
```

---

qcondbvtcop *C_[2|1]^[-1](p|u) for bivariate Student t copula*

Description

bivariate Student t copula conditional quantile

Usage

```
qcondbvtcop(p,u,cpar)
```

Arguments

p	0<p<1, could be a vector
u	0<u<1, could be a vector
cpar	copula parameter: 2-vector with -1<rho<1, df>0

Value

conditional quantiles of bivariate Student t copula

---

qcondFrank *C_[2|1]^[-1](p|u) for bivariate Frank copula*

Description

Frank bivariate copula conditional quantile

Usage

```
qcondFrank(p,u,cpar)
```

Arguments

p	0<p<1, could be a vector
u	0<u<1, could be a vector
cpar	copula parameter: cpar>0 or cpar<0; cpar=0 input will not work

Details

1-exp(-cpar) becomes 1 in double precision for cpar>37.4; any argument can be a vector, but all vectors must have same length. Form of inputs not checked (for readability of code).

Value

conditional quantiles of bivariate Frank copula

`r1factor` *simulate from 1-factor copula model with different linking copula families*

Description

simulate from 1-factor copula model and include corresponding latent variables

Usage

```
r1factor(n,d,param,famvec,vlatent=NULL)
```

Arguments

n	sample size
d	dimension
param	copula parameter vector of dimension 2*d (par[j],par2[j], j=1,...,d) for d linking copulas, for one-parameter families set par2=0
famvec	family vector for d linking copulas same index with VineCopula package, for a range of tail properties, select 1:Gaussian; 2:t; 4:Gumbel; 14:survival Gumbel; 5:Frank; 7:BB1; 17:survival BB1; 10:BB8; 20:survival BB8;
vlatent	given n-vector of latent variavles in U(0,1); use for simulating from a group for oblique factor copula (in this case, apply this function G times for G groups, extracting dependent latent variables from a nxG matrix)

Value

list with udata: nxd matrix in (0,1) andvlatent n-vector of corresponding latent variables in (0,1).

Examples

```
 # Example 1
cpar_frk = c(12.2,3.45,4.47,4.47,5.82)
d = length(cpar_frk)
cpar2_frk = rep(0,d)
param = c(rbind(cpar_frk,cpar2_frk))
n = 300
set.seed(123)
frk_obj = r1factor(n,d,param,famvec=rep(5,d)) # uses VineCopula
```

```
frkdat = frk_obj$udata
print(cor(frkdat))
print(summary(frk_obj$vlatent))
dfrank = function(u,v,cpar)
{ t1 = 1.-exp(-cpar); tem1 = exp(-cpar*u); tem2 = exp(-cpar*v);
  pdf = cpar*tem1*tem2*t1; tem = t1-(1.-tem1)*(1.-tem2);
  pdf = pdf/(tem*tem);
  pdf
}
cat("\nFrank 1-factor MLE: standalone R\n")
out1_frk = ml1factor(nq=21,cpar_frk,frkdat,dfrank,LB=-30,UB=30,prlevel=1,mxiter=100)
cat("\nFrank 1-factor MLE: nlm with f90 code\n")
out2_frk = ml1factor_f90(nq=21,cpar_frk,frkdat,copname="frank",LB=-30,UB=30,prlevel=1,mxiter=100)
cat("\nFrank 1-factor MLE: posDefhessMinb with f90 code\n")
gl21 = gaussLegendre(21)
dstrfrk = list(copname="frank",data=frkdat,quad=gl21,repar=0)
out3_frk = posDefHessMinb(cpar_frk,onefactorcop_nllk,ifixed=rep(FALSE,d),
dstruct=dstrfrk, LB=rep(-30,d),UB=rep(30,d),iprint=TRUE,eps=1.e-5)
cat(out1_frk$minimum, out2_frk$minimum, out3_frk$fnval,"\n")
print(cbind(out1_frk$estimate, out2_frk$estimate, out3_frk$parmin))
print(sqrt(diag(out3_frk$invh))) # SEs
#
# Example 2 (oblique factor with 3 groups)
n = 500
d = 10
ltd1 = c(0.3,0.4,0.6,0.7,0.6); utd1 = c(0.5,0.6,0.7,0.5,0.4)
ltd2 = c(0.3,0.4,0.6,0.5,0.4); utd2 = c(0.6,0.3,0.5,0.7,0.6)
ltd3 = c(0.5,0.4,0.5,0.5); utd3 = c(0.5,0.4,0.5,0.5)
grsize = c(5,5,4)
rmat = toeplitz(c(1,0.5,0.5))  # for Gaussian copula parameter of  latent
cpar1 = bb1_td2cpar(cbind(ltd1,utd1))
cpar2 = bb1_td2cpar(cbind(ltd2,utd2))
cpar3 = bb1_td2cpar(cbind(ltd3,utd3))
set.seed(205)
zmat = rmvn(n,rmat); vmat = pnorm(zmat)
# Any vine copula could be used for latent variables, besides multivariate normal
data1g = r1factor(n=500,grsize[1],param=c(t(cpar1)),
famvec=rep(7,grsize[1]), vlatent=vmat[,1])
data2g = r1factor(n=500,grsize[2],param=c(t(cpar2)),
famvec=rep(7,grsize[2]), vlatent=vmat[,2])
data3g = r1factor(n=500,grsize[3],param=c(t(cpar3)),
famvec=rep(7,grsize[3]), vlatent=vmat[,3])
udata_oblf = cbind(data1g$udata,data2g$udata,data3g$udata)
rr_oblf = cor(udata_oblf,method="spearman")
print(round(rr_oblf,2))
```

`rainstorm` *Precipitation by rainstorm at 28 stations*

Description

R workspace file with transformed precipitation by rainstorm at 28 stations

There are 4 components:

1. $uprecip: 256x28 matrix with total precipitation in mm by storm at 28 stations, after rank transform to U(0,1);

2. $zprecip: 256x28 matrix with total precipitation in mm by storm at 28 stations, after rank transform to N(0,1);

3. $cormat: the correlation matrix of $zprecip;

4. $grsize: vector (6,12,10) for sizes of 3 goups of stations found by a variable clustering method.

Usage

```
data(rainstorm)
```

`rbifactor` *simulate from bi-factor copula model*

Description

simulate from bi-factor copula model and include corresponding latent variables

Usage

```
rbifactor(n, grsize, cop=5, param)
```

Arguments

`n`	sample size
`grsize`	G-vector of group sizes for G groups
`cop`	code for copula families 1: Gaussian/Gaussian; 2: t/t; 4: Gumbel/Gumbel; 5: Frank/Frank; 7: BB1/Frank; 14: survival Gumbel, 17: survivalBB1 /Frank if cop = 1, data have standard normal marginals if cop = 2, data have t marginals if cop > 2, data have uniform(0,1) marginals
`param`	vector of parameters (those for the common factor go first) The order in param is the same as in start for mvtbifct(full=T) function. For BB1/Frank: BB1thetas then BB1deltas, then Frank parameters – the order in param is the same as in start for mvtbifct(full=F) function

Details

The user can modify this code to get other linking copulas.

Value

list with data: nxd data set with U(0,1) or N(0,1) or t(df) margin; v0: n-vector of corresponding global latent variables; and vg: nxG matrix of corresponding local (group) latent variables.

Examples

```
grsize = c(4,3)
cop = 4; param4 = c(seq(1.5,2.1,0.1),  rep(1.1,7))
cop = 14; param14 = c(seq(1.5,2.1,0.1),  rep(1.1,7))
cop = 5; param5 = c(seq(1.5,2.1,0.1),  rep(1.1,7))
cop = 1; param1 = c(0.5,0.6,0.7,0.8,0.9,0.4,0.5,  rep(1.1,7))
cop = 2; param2 = c(0.5,0.6,0.7,0.8,0.9,0.4,0.5,  rep(1.1,7), 7)
cop = 7; param7 = c(seq(0.5,1.1,0.1), 1.5,1.6,1.7,1.8,1.9,1.4,1.5, rep(1.1,7))
cop = 17; param17 = c(seq(0.5,1.1,0.1), 1.5,1.6,1.7,1.8,1.9,1.4,1.5, rep(1.1,7))
set.seed(123)
gumdat = rbifactor(n=10, grsize=grsize, cop=4, param=param4)    # U(0,1)
gumrdat = rbifactor(n=10, grsize=grsize, cop=14, param=param14) # U(0,1)
frkdat = rbifactor(n=10, grsize=grsize, cop=5, param=param5)    # U(0,1)
gaudat = rbifactor(n=10, grsize=grsize, cop=1, param=param1)    # N(0,1)
bvtdat = rbifactor(n=10, grsize=grsize, cop=2, param=param2)    # t_7
bb1frkdat = rbifactor(n=10, grsize=grsize, cop=7, param=param7)    # U(0,1)
bb1rfrkdat = rbifactor(n=10, grsize=grsize, cop=17, param=param17) # U(0,1)
summary(bb1frkdat$data)
summary(bb1frkdat$v0)
summary(bb1frkdat$vg)
```

`residDep` *correlation matrix for 1-factor plus 1-truncated vine (for residual dependence)*

Description

correlation matrix for 1-factor plus 1-truncated vine (for residual dependence)

Usage

```
residDep(cormat,loading)
```

Arguments

`cormat`	dxd correlation matrix
`loading`	d-dimensional loading vector (for latent factor), -1<loading[j]<1

Details

MST algorithm with weights log(1-rho^2), rho's are partial correlations fiven the latent variable. not exported

Value

list with R = correlation matrix for structure of 1-factor+Markov tree residual dependence;incl = d*(d-1)/2 binary vector: indicator of edges [1,2], [1,3], [2,3], [1,4], ...[d-1,d] edges in tree with d-1 edges; partcor = conditional correlation matrix given the latent variable.

rhoS *Spearman's rho for bivariate copula with parameter cpar*

Description

Spearman's rho for bivariate copula with parameter cpar

Usage

```
rhoS(cpar,cop, zero=0, icond=FALSE, tol=0.0001)
```

Arguments

cpar	copula parameter
cop	function name of joint cdf or conditional cdf C_2|1
zero	0 or something like 1.e-6 (to avoid endpoint problems)
icond	icond flag for using condition cdf of copula; if icond = TRUE, cop = conditional cdf pcondcop if icond = FALSE, cop = joint cdf pcop default is to integrate on [zero,1-zero]^2 for icond=TRUE.
tol	accuracy for 2-dimensional integration

Value

Spearman rho value for copula

Examples

```
# Bivariate margin of 1-factor copula
# using conditional cdf via VineCopula::BiCopHfunc2
# cpar1 = (par,par2) for fam1 for first variable
# cpar2 = (par,par2) for fam2 for second variable
# family codes 1:Gaussian, 2:t, 4:Gumbel, 5:Frank, 7:BB1, 14:survGumbel, 17:survBB1
p1factbiv = function(u1,u2,cpar1,cpar2,fam1,fam2,nq)
{ if(length(u1)==1) u1 = rep(u1,nq)
  if(length(u2)==1) u2 = rep(u2,nq)
  gl = gaussLegendre(nq)
  wl = gl$weights; vl = gl$nodes
  a1 = VineCopula::BiCopHfunc2(u1,vl,family=fam1,par=cpar1[1], par2=cpar1[2])
  a2 = VineCopula::BiCopHfunc2(u2,vl,family=fam2,par=cpar2[1], par2=cpar2[2])
  sum(wl*a1*a2)
}
#
# Spearman rho
rhoS_1factor = function(cpar1,cpar2,fam1,fam2,nq)
{ gl = gaussLegendre(nq)
  wl = gl$weights; xl = gl$nodes
  pcint1 = rep(0,nq); pcint2 = rep(0,nq)
  for(iq in 1:nq)
```

```
 { a1 = VineCopula::BiCopHfunc2(xl,rep(xl[iq],nq),family=fam1,par=cpar1[1],par2=cpar1[2])
   a2 = VineCopula::BiCopHfunc2(xl,rep(xl[iq],nq),family=fam2,par=cpar2[1],par2=cpar2[2])
     pcint1[iq] = sum(wl*a1)
     pcint2[iq] = sum(wl*a2)
   }
   tem = sum(wl*pcint1*pcint2)
   12*tem-3
}
#
# Tests for Spearman rho with BB1 linking copulas to latent variable
param = matrix(c(0.5,1.5,0.6,1.2),2,2,byrow=TRUE)
# Gauss-Legendre quadrature
rho_1factbb1_gl = rhoS_1factor(param[1,],param[2,],fam1=7,fam2=7,nq=21)
# reflected/survival BB1
rho_1factbb1r_gl = rhoS_1factor(param[1,],param[2,],fam1=17,fam2=17,nq=21)
cat(rho_1factbb1_gl, rho_1factbb1r_gl, "\n")
# 0.3401764 0.339885
#
pcop1fact_bb1 = function(u1,u2,param)
{ p1factbiv(u1,u2,param[c(1,3)],param[c(2,4)],fam1=7,fam2=7,nq=21) }
#
pcop1fact_bb1r = function(u1,u2,param)
{ p1factbiv(u1,u2,param[c(1,3)],param[c(2,4)],fam1=17,fam2=17,nq=21) }
#
# Using function rhoS() based on adaptive integration
library(cubature)
rho_1factbb1_ai = rhoS(c(param),pcop1fact_bb1,zero=0.00001,tol=0.00001)
rho_1factbb1r_ai = rhoS(c(param),pcop1fact_bb1r,zero=0.00001,tol=0.00001)
cat(rho_1factbb1_ai, rho_1factbb1r_ai, "\n")
# 0.3400871 0.3397855
# same to 3 decimal places
```

---

rmvn *Random multivariate normal (standard N(0,1) margins)*

Description

Random multivariate normal (standard N(0,1) margins)

Usage

```
rmvn(n,rmat)
```

Arguments

n	simulation sample size
rmat	correlation matrix

Value

nxd matrix, where d=nrow(rmat)

`rmvt` *Random multivariate t (standard t(nu) margins)*

Description

Random multivariate t (standard t(nu) margins)

Usage

```
rmvt(n,rmat,nu)
```

Arguments

`n`	simulation sample size
`rmat`	correlation matrix
`nu`	degree of freedom parameter

Value

nxd matrix, where d=nrow(rmat)

`rnestfactor` *Simulate data from nested copula or Gaussian model*

Description

Simulate data from nested copula or Gaussian model

Usage

```
rnestfactor(n,grsize,cop,param)
```

Arguments

`n`	sample size
`grsize`	G-vector of group sizes for G groups
`cop`	number code: 1: Gaussian; 2: t; 4: Gumbel; 14: survival Gumbel; 5: Frank; 7: Gumbel+BB1
`param`	vector of parameters, length is d+mgrp(+1 for cop==2):

Details

The user can modify this code to get other linking copulas.

Value

d-dimensional random sample with U(0,1) margins or N(0,1) or t(df) margin; v0: n-vector of corresponding global latent variables; and vg: nxG matrix of corresponding local (group) latent variables

Examples

```
grsize = c(3,3,3)
cop = 4; param4 = c(1.1,1.1,1.1,  seq(1.5,2.3,0.1))
cop = 14; param14 = c(1.1,1.1,1.1,  seq(1.5,2.3,0.1))
cop = 5; param5 = c(1.1,1.1,1.1,  seq(1.5,2.3,0.1))
cop = 1; param1 = c(0.4,0.4,0.4, 0.5,0.6,0.7,0.8,0.9,0.4,0.5,0.6,0.7 )
cop = 2; param2 = c(0.4,0.4,0.4, 0.5,0.6,0.7,0.8,0.9,0.4,0.5,0.6,0.7, 7 )
cop = 7; param7 = c(1.5,1.5,1.5, seq(0.5,1.3,0.1), 1.5,1.6,1.7,1.8,1.9,1.4,1.5,1.6,1.7)
set.seed(123)
gumdat = rnestfactor(n=10, grsize=grsize, cop=4, param=param4)
gumrdat = rnestfactor(n=10, grsize=grsize, cop=14, param=param14)
frkdat = rnestfactor(n=10, grsize=grsize, cop=5, param=param5)
gaudat = rnestfactor(n=10, grsize=grsize, cop=1, param=param1)
bvtdat = rnestfactor(n=10, grsize=grsize, cop=2, param=param2)
gumbb1dat = rnestfactor(n=10, grsize=grsize, cop=7, param=param7)
summary(gumbb1dat$data)
round(cor(gumbb1dat$data),2)
summary(gumbb1dat$v0)
summary(gumbb1dat$vg)
```

RVtrunc2cor *compute correlation matrix from 2-truncated R-vine*

Description

compute correlation matrix from 2-truncated R-vine

Usage

```
RVtrunc2cor(RVobj)
```

Arguments

RVobj	R-vine object with vine array and partial correlation matrix variable 1 is the latent variable; list with $Varray (d+1)x(d+1), $PCor 2x(d+1)

Details

not exported

Value

correlation matrix for 2-truncated vine structure based on partial correlations in tree 2

semiCor *Semi-correlations for two variables*

Description

semi-correlations (lower and upper) applied to data after normal scores transform

Usage

```
semiCor(bivdat,inscore=FALSE)
```

Arguments

`bivdat`	nx2 data set
`inscore`	TRUE if bivdat has already been converted to normal scores, default FALSE

Value

3-vector with rhoN = correlation of normal scores (vander Waerden correlation) and lower/ upper semi-correlations

Examples

```
# See example in semiCorTable()
```

semiCorTable *Semi-correlation table for a multivariate data set*

Description

Semi-correlation table for several variables

Usage

```
semiCorTable(mdat, varnames, inscore=FALSE)
```

Arguments

`mdat`	nxd multivariate data set with d>=2 columns
`varnames`	d-vector of (abbreviated) variable names
`inscore`	TRUE if mdat has already been converted to normal scores, default FALSE

Value

d*(d-1)/2 by 8-column dataframe with columns j1,j2,ncor,lcor,ucor,bvnsemic, varnames[j1] varnames[j2]for 2 variable indices, correlation of normal scores, lower semi-correlation, upper semi-correlation and BVN semi-correlation assuming Gaussian copula Stronger than Gaussian dependence in upper tail if ucor is larger than bvn semicor

Examples

```
rmat = toeplitz(c(1,.7,.4))
print(rmat)
set.seed(1234)
zdat = rmvn(n=500,rmat)
set.seed(12345)
tdat = rmvt(n=500,rmat,nu=5)
vnames=c("V1","V2","V3")
zsemi = semiCorTable(zdat,vnames)
tsemi = semiCorTable(tdat,vnames)
cat("trivariate normal\n")
print(zsemi)
cat("trivariate t(5)\n")
print(tsemi)
```

`tailDep` *Tail dependence parameter estimation*

Description

Tail dependence parameter estimate based on extrapolating zeta(alpha)

Usage

```
tailDep(u1,u2, lowertail=FALSE, eps=0.1, semictol=0.1, rank=TRUE,
  iprint=FALSE)
```

Arguments

`u1`	nx1 vector with values (in (0,1) if rank=FALSE)
`u2`	nx1 vector with values (in (0,1) if rank=FALSE)
`lowertail`	TRUE if lower tail-weighted dependence measure, default is FALSE
`eps`	tolerance (default 0.1) for rate parameter in method 2; non-linear (use method 2) if rate < 1-eps
`semictol`	tolerance (default 0.1) for exceedance of normal semicorrelation to treat as tail dependent; use something like semicol=-0.5 if normal scores plot suggest tail dependence
`rank`	TRUE (default) if data matrix needs to be converted to uniform scores in (0,1)
`iprint`	TRUE for intermediate prints

Value

upper tail dependence parameter based on extrapolation of zeta(alpha) for large alpha

References

Lee D, Joe H, Krupskii P (2018). J Nonparametric Statistics, 30(2), 262-290

Examples

```
mytest = function(qcond,cpar,n=500,seed=123)
{ set.seed(seed)
  u1 = runif(n)
  u2 = qcond(runif(n),u1,cpar)
  #convert to uniform scores (marginals are usually not known)
  u1 = (rank(u1)-0.5)/n
  u2 = (rank(u2)-0.5)/n
  alp = c(1,5,10:20)
  zetaL = zetaDep(cbind(u1,u2),alp,rank=FALSE,lowertail=FALSE)
  zetaU = zetaDep(cbind(u1,u2),alp,rank=FALSE,lowertail=TRUE)
  print(cbind(alp,zetaL,zetaU))
  utd = tailDep(u1,u2, lowertail=FALSE, eps=0.1, semictol=0.1, rank=FALSE, iprint=TRUE)
  ltd = tailDep(u1,u2, lowertail=TRUE, eps=0.1, semictol=0.1, rank=FALSE, iprint=TRUE)
  cat(ltd,utd,"\n")
  utd = tailDep(u1,u2, lowertail=FALSE, eps=0.1, semictol=0.1, rank=FALSE, iprint=FALSE)
  ltd = tailDep(u1,u2, lowertail=TRUE, eps=0.1, semictol=0.1, rank=FALSE, iprint=FALSE)
  cat(ltd,utd,"\n")
  #par(mfrow=c(2,1))
  #zetaPlot(cbind(u1,u2),alp,ylim=c(0,1),inverse=FALSE)
  #zetaPlot(cbind(u1,u2),alp,ylim=c(0,1),inverse=TRUE)
  0
}
mytest(qcondFrank,3)
mytest(qcondbvtcop,c(0.6,5))
```

`uscore` *Rank-based uniform scores transform*

Description

Rank-based uniform scores transform

Usage

```
uscore(data,aunif=-0.5)
```

Arguments

`data`	dataframe or matrix, or vector, of reals
`aunif`	adjustment 'a' for (rank+a)/(n+1+2*a) as scores. n=sample size , default is -0,5

Value

matrix or vector of uniform scores

`zetaDep` *Empirical version of zeta(alpha) tail-weighted dependence measure*

Description

Empirical version of zeta(alpha) tail-weighted dependence measure

Usage

```
zetaDep(dat,alpha,rank=TRUE,lowertail=FALSE)
```

Arguments

`dat`	nx2 data matrix with values (in (0,1) if rank=FALSE)
`alpha`	vector of alpha>0 for zeta measure
`rank`	TRUE (default) if to convert data matrix to uniform scores in (0,1)
`lowertail`	TRUE if lower tail-weighted dependence measure, default is FALSE

Details

This is a central dependence measure if alpha =1 and upper tail-weighted is alpha»1

Value

Dependence measure zeta(alpha)

References

Lee D, Joe H, Krupskii P (2018). J Nonparametric Statistics, 30(2), 262-290

Examples

```
data(euro07gf)
udat = euro07gf$uscore
euro07names = colnames(udat)
d = ncol(udat)
for(j2 in 2:d)
{ for(j1 in 1:(j2-1))
  { zetaU = zetaDep(udat[,c(j1,j2)],alpha=15,rank=FALSE,lowertail=FALSE)
    zetaL = zetaDep(udat[,c(j1,j2)],alpha=15,rank=FALSE,lowertail=TRUE)
    zeta1 = zetaDep(udat[,c(j1,j2)],alpha=1,rank=FALSE,lowertail=FALSE)
   cat(j1,j2,round(zeta1,3),round(zetaL,3),round(zetaU,3),euro07names[j1],euro07names[j2],"\n")
  }
}
```

zetaDepC *Upper Tail-weighted dependence measure zeta(C,alpha)*

Description

Upper Tail-weighted dependence measure zeta(C,alpha)

Usage

```
zetaDepC(cpar,pcop,alpha,zero=0,iGL=FALSE,gl=0)
```

Arguments

`cpar`	copula parameter (vector) of pcop
`pcop`	copula cdf C, assume this takes vectorized form for u,v
`alpha`	scalar alpha>0 for zeta measure
`zero`	tolerance to use for zero if integrate is used, e.g., 0.0001
`iGL`	TRUE to use Gauss-Legendre quadrature
`gl`	Gauss-Legendre quadratureobject with nodes/weights if iGL=TRUE

Details

zeta(alpha)=2+alpha-alpha/integral integral int_0^1 C(x^(1/alpha), x^(1/alpha)) dx. This is a central dependence of measure if alpha =1 and upper tail-weighted is alpha»1.

Value

zeta(C,alpha) ; this is upper tail-weighted is alpha»1

References

Lee D, Joe H, Krupskii P (2018). J Nonparametric Statistics, 30(2), 262-290

Examples

```
# Bivariate margin of 1-factor copula
# using conditional cdf via VineCopula::BiCopHfunc2
# cpar1 = par,par2 for fam1 for first variable
# cpar2 = par,par2 for fam2 for second variable
# family codes 1:Gaussian, 2:t, 4:Gumbel, 5:Frank, 7:BB1, 14:survGumbel 17:survBB1
p1factbiv = function(u1,u2,cpar1,cpar2,fam1,fam2,nq)
{ if(length(u1)==1) u1 = rep(u1,nq)
  if(length(u2)==1) u2 = rep(u2,nq)
  gl = gaussLegendre(nq)
  wl = gl$weights
  vl = gl$nodes
  a1 = VineCopula::BiCopHfunc2(u1,vl,family=fam1,par=cpar1[1], par2=cpar1[2])
  a2 = VineCopula::BiCopHfunc2(u2,vl,family=fam2,par=cpar2[1], par2=cpar2[2])
  sum(wl*a1*a2)
}
#
# Version of zeta(C) for bivariate margin of 1-factor copula
zetaDepC_1factor = function(cpar1,cpar2,fam1,fam2,nq,alpha,zero=0,iGL=FALSE,gl=0)
{ a1 = 1/alpha
  gfn = function(x)
  { nn = length(x)
    gval = rep(0,nn)
    # need this form because of nesting in p1factbiv()
    for(ii in 1:nn)
    { xx = x[ii]^a1
      gval[ii] = p1factbiv(xx,xx,cpar1,cpar2,fam1,fam2,nq)
    }
    gval
  }
  if(iGL)
  { xq = gl$nodes
    wq = gl$weight
    tem = sum(wq*gfn(xq))
  }
  else
  { tem = integrate(gfn,zero,1-zero)
  tem = tem$value
  }
  zeta = 2+alpha-alpha/tem
  zeta
}
# Tests for zeta
param = matrix(c(0.5,1.5,0.6,1.2),2,2,byrow=TRUE)
# Create BB1 copula cdf pbb1 and  BB1 surival copula cdf pbb1r using BiCopCDF
pbb1_VC = function(u,v,cpar) { VineCopula::BiCopCDF(u,v,family=7,par=cpar[1],par2=cpar[2]) }
pbb1r_VC = function(u,v,cpar) { VineCopula::BiCopCDF(u,v,family=17,par=cpar[1],par2=cpar[2]) }
gl21 = gaussLegendre(21)
zeta1u_bb1 = zetaDepC(param[1,],pbb1_VC,alpha=10,zero=0.00001,iGL=TRUE,gl=gl21)
zeta1l_bb1 = zetaDepC(param[1,],pbb1r_VC,alpha=10,zero=0.00001,iGL=TRUE,gl=gl21)
cat(zeta1u_bb1,zeta1l_bb1,"\n")
# 0.4504351 0.4776329
zeta2u_bb1 = zetaDepC(param[2,],pbb1_VC,alpha=10,zero=0.00001,iGL=TRUE,gl=gl21)
zeta2l_bb1 = zetaDepC(param[2,],pbb1r_VC,alpha=10,zero=0.00001,iGL=TRUE,gl=gl21)
cat(zeta2u_bb1,zeta2l_bb1,"\n")
# 0.2825654 0.419787
# Bivariate margin of 1-factor copula: linking BB1(param1) and BB1(param2)
# Upper tail
zetau_ai = zetaDepC_1factor(param[1,],param[2,],fam1=7,fam2=7,nq=21,alpha=10,
zero=0.00001,iGL=FALSE,gl=0)
zetau_gl = zetaDepC_1factor(param[1,],param[2,],fam1=7,fam2=7,nq=21,alpha=10,
zero=0.00001,iGL=TRUE,gl=gl21)
cat(zetau_ai,zetau_gl,"\n")
# 0.1766584 0.1775621
# Lower tail
zetal_ai = zetaDepC_1factor(param[1,],param[2,],fam1=17,fam2=17,nq=21,alpha=10,
zero=0.00001,iGL=FALSE,gl=0)
zetal_gl = zetaDepC_1factor(param[1,],param[2,],fam1=17,fam2=17,nq=21,alpha=10,
zero=0.00001,iGL=TRUE,gl=gl21)
cat(zetal_ai,zetal_gl,"\n")
# 0.2664287 0.26737
# Ordering is expected based on the individual BB1(param1) and BB1(param2)
```

`zetaPlot` *Plot zeta(alpha) against alpha*

Description

Plot zeta(alpha) against alpha

Usage

```
zetaPlot(dat,alpha,ylim=c(0,1),inverse=FALSE)
```

Arguments

`dat`	nx2 data matrix with values (u-data in (0,1))
`alpha`	vector of alpha>0 for zeta measure
`ylim`	limits for yaxis to pass to plot
`inverse`	if TRUE, plot zeta against 1/alpha

Value

nothing is returned, but a plot is produced

Index