```
#Installing packages
install.packages("mnormt")
install.packages("pcaPP")
install.packages("rgl")
install.packages("QRM")
install.packages("VGAM")
install.packages("copula")
install.packages("fCopulae")
require(mnormt)
require(pcaPP)
require(rgl)
require("QRM")
require("VGAM")
require("copula")
require("fCopulae")


##############################################################
####
#Functions
##############################################################
####

######## Color palette functions Functions #############

###Displaying the Rainbow colour palette for n colours (ranks)
#Code is adapted from http://stat.ethz.ch/R-manual/R-
devel/library/grDevices/html/palettes.html
demo.pal <- function(n, border = if (n<32) "light gray" else
NA,
           main = paste("Rainbow palette"))
{
    rank <- 1:n; j <- n; d <- j/6; dy <- 2*d
    plot(rank,rank+d, type="n", yaxt="n", ylab="", main=main)
    rect(rank-.5, dy, rank+.4, j,
             col = rainbow(n), border = border)
    text(2*j, j +dy/4, rainbow(n))
}


######## Copula functions #############
###Empirical copula function
###Uses Equation reference (2.2)

#Produces a length(v1)-vector 'ans' s.t.
ans[i]=C_n(v1[i],v2[i]) with u1 and u2 being the inputted
data.
#v1 and v2 can be vectors or numbers, but have to be the same
length
empCopula<-function(v1,v2,u1,u2,n) {
      #Stops process if v1 and v2 are different lengths or if
n!=length(u1)!=length(u2)
      if(length(v1)!=length(v2)|n!=length(u1)|n!=length(u2)) {
           stop("length(v1)!=length(v2) or n!=length(u1) or n!
=length(u2)")
           }
      #Creates ans vector
```

```
      ans<-numeric(length(v1))
      #Applies Equation reference (2.10) to each entry in
(v1,v2) with u1 and u2 being the iid sample of U
      for(i in 1:length(v1)) {
            ans[i]<-
(1/n)*sum(as.numeric(u1<=v1[i])*as.numeric(u2<=v2[i]))
            }
      #Returns ans vector
      return(ans)}

#Produces a length(v1)xlength(v2) matrix ans s.t.
ans[i,j]=C_n(v1[i],v2[j]) with u1 and u2 being the inputted
variables.
empCopulaSur<-function(v1,v2,u1,u2,n) {
      #Stops process if n!=length(u1)!=length(u2)
      if(n!=length(u1)|n!=length(u2)) {
            stop("n=length(u1)=length(u2) does not hold")
            }
      #Creates ans matrix
      ans<-matrix(0,length(v1),length(v2))
      #Applies Equation reference (2.10) to each entry in v1 x
v2 with u1 and u2 being the iid sample of U
      for(i in 1:length(v1)) {
            for(j in 1:length(v2)) {
                  ans[i,j]<-
(1/n)*sum(as.numeric(u1<=v1[i])*as.numeric(u2<=v2[j]))
                  }
            }
      #Returns ans vector
      return(ans)}

###Gaussian copula function

#Produces a length(v1)-vector 'ans' s.t.
ans[i]=C^{Ga}_R(v1[i],v2[i]) where R_12=sin(pi*tau/2)
#v1 and v2 can be vectors or numbers, but have to be the same
length
gauCopula<-function(v1,v2,tau) {
      #Stops process if v1 and v2 are different lengths
      if(length(v1)!=length(v2)) {
            stop("length of v1 and v2 differ")
            }
      #Calculates R_12 and names it R12
      R12<-sin(pi*tau/2)
      #Checks to see if R_12!=1, stops process if not true
      if(R12==1) {
            stop("Not valid Gaussian copula as R_12=1")
            }
      #Calculates R matrix and names it covmat
      covmat<-matrix(c(1,R12,R12,1),2)
      #Creates ans vector
      ans<-numeric(length(v1))
      #Applies equation from Definition reference (2.3.1) with
u=v1[i], v=v2[i] and R=covmat
      for(i in 1:length(v1)) {
            ans[i]<-
pmnorm(c(qnorm(v1[i]),qnorm(v2[i])),mean=rep(0,2),varcov=covma
```

```
t)
      }
      #Returns ans vector
      return(ans)
}

#Produces a length(v1)xlength(v2) matrix ans s.t.
ans[i,j]=C^{Ga}_R(v1[i],v2[j]) where R_12=sin(pi*tau/2)
gauCopulaSur<-function(v1,v2,tau) {
      #Calculates R_12 and names it R12
      R12<-sin(pi*tau/2)
      #Checks to see if R_12!=1, stops process if not true
      if(R12==1) {
            stop("Not valid Gaussian copula as R_12=1")
            }
      #Calculates R matrix and names it covmat
      covmat<-matrix(c(1,R12,R12,1),2)
      #Creates ans matrix
      ans<-matrix(0,length(v1),length(v2))
      #Applies Equation from Definition reference (2.3.1) with
u=v1[i], v=v2[j] and R=covmat
      for(i in 1:length(v1)) {
            for(j in 1:length(v2)) {
                  ans[i,j]<-
pmnorm(c(qnorm(v1[i]),qnorm(v2[j])),mean=rep(0,2),varcov=covma
t)
            }
      }
      #Returns ans matrix
      return(ans)
}

###Gumbel copula function

#Produces a length(v1)-vector 'ans' s.t.
ans[i]=C^{Gum}_theta(v1[i],v2[i]) where theta=(1-tau)^(-1)
#v1 and v2 can be vectors or numbers, but have to be the same
length
gumCopula<-function(v1,v2,tau) {
      #Stops process if v1 and v2 are different lengths
      if(length(v1)!=length(v2)) {
            stop("length of v1 and v2 differ")
            }
      #Calculates theta
      theta<-1/(1-tau)
      #Checks to see if Inf>theta>=1, stops process if not true
      if(theta<1|theta==Inf) {
            stop("Not valid Gumbel copula as theta<1 or
theta==Inf")
            }
      #Creates ans vector
      ans<-numeric(length(v1))
      #Applies Equation reference (2.7) with u=v1[i], v=v2[i]
and theta
      for(i in 1:length(v1)) {
            ans[i]<-exp(-((-log(v1[i]))^theta+(-
log(v2[i]))^theta)^(1/theta))
```

```
	}
	#Returns ans vector
	return(ans)
}

#Produces a length(v1) x length(v2) matrix 'ans' s.t.
ans[i,j]=C^{Gum}_theta(v1[i],v2[j]) where theta=(1-tau)^-1
gumCopulaSur<-function(v1,v2,tau) {
	#Calculates theta
	theta<-1/(1-tau)
	#Checks to see if Inf>theta>=1, stops process if not
	if(theta<1|theta==Inf) {
		stop("Not valid Gumbel copula as theta<1 or
theta==In")
	}
	#Creates ans matrix
	ans<-matrix(0,length(v1),length(v2))
	#Applies equation (2.20) with u=v1[i], v=v2[j] and theta
	for(i in 1:length(v1)) {
		for(j in 1:length(v2)) {
			ans[i,j]<-exp(-((-log(v1[i]))^theta+(-
log(v2[j]))^theta)^(1/theta))
		}
	}
	#Returns ans matrix
	return(ans)
}

###Clayton copula function
###

#Produces a length(v1)-vector 'ans' s.t.
ans[i]=C^{cla}_theta(v1[i],v2[i]) where theta =2*tau/(1-tau)
#v1 and v2 can be vectors or numbers, but have to be the same
length
claCopula<-function(v1,v2,tau) {
	#Stops process if v1 and v2 are different lengths
	if(length(v1)!=length(v2)) {
		stop("length of v1 and v2 differ")
		}
	#Calculates theta
	theta<-2*tau/(1-tau)
	#Checks to see if Inf>theta>0, stops process if not true
	if(theta<=0|theta==Inf) {
		stop("Not valid clayton copula as theta<1 or
theta==Inf or theta==0")
		}
	#Creates ans vector
	ans<-numeric(length(v1))
	#Applies Equation reference -- with u=v1[i], v=v2[i] and
theta
	for(i in 1:length(v1)) {
		ans1=max(v1[i]^(-theta)+v2[i]^(-theta)-1,0)
		ans[i]<-ans1^(-1/theta)
	}
	#Returns ans vector
	return(ans)
```

```
}

#Produces a length(v1) x length(v2) matrix 'ans' s.t.
ans[i,j]=C^{cla}_theta(v1[i],v2[j]) where theta =2*tau/(1-tau)

claCopulaSur<-function(v1,v2,tau) {
      #Calculates theta
      theta<-2*tau/(1-tau)
      #Checks to see if Inf>theta>0, stops process if not true
      if(theta<=0|theta==Inf) {
            stop("Not valid clayton copula as theta==Inf or
theta<=0")
            }
      #Creates ans matrix
      ans<-matrix(0,length(v1),length(v2))
      #Applies equation -- with u=v1[i], v=v2[j] and theta
      for(i in 1:length(v1)) {
            for(j in 1:length(v2)) {
                  ans1=max(v1[i]^(-theta)+v2[j]^(-theta)-1,0)
                  ans[i,j]<-ans1^(-1/theta)
                  }
      }
      #Returns ans matrix
      return(ans)
}


###Two-component copula function
###Uses Equation in Remark 3.3

#Produces an length(v1)-vector 'ans' s.t.
ans[i]=C^{Tc}_(alpha1,alpha2)(v1[i],v2[i])
#alpha1 and alpha2 have to be positive, otherwise gamma cdf is
not valid
#v1 and v2 can be vectors or numbers, but have to be the same
length
tcCopula<-function(v1,v2,alpha1,alpha2) {
      #Stops process if length(v1)!=length(v2) or if alpha1<=0
or alpha2<=0
      if(alpha1<=0 | alpha2<=0) {
            stop("alpha1 and alpha2 are not positive")}
      if(length(v1)!=length(v2)) {
            stop("length of v1 and v2 differ")
      }
      #Creates ans vector
      ans<-numeric(length(v1))
      #Applies Equation in Theorem reference 3.2.1 with u_1
=v1[i], u_2=v2[i]
      for(i in 1:length(v1)) {
            if(v1[i]==0 && v2[i]<1){
                  ans[i]=0
            }else if(v1[i]<1 && v2[i]==0) {
                  ans[i]=0
            }else if(v2[i]==1){
                  ans[i]=v1[i]
            }else if(v1[i]==1){
                  ans[i]=v2[i]
```

```
            } else{
                  f<-function(x) {
                        pgamma(x/((1-
v1[i])^(-1/alpha1)-1),shape=alpha1,scale=1)*pgamma(x/((1-
v2[i])^(-1/alpha2)-1),shape=alpha2,scale=1)*exp(-x)
                  }
                  ans[i]<-v1[i]+v2[i]-1+integrate(f,lower=
0,upper=Inf)$value
            }
      }
      #Returns ans vector
      return(ans)
}

#Produces a length(v1) x length(v2) matrix 'ans' s.t.
ans[i,j]=C^{Tc}_(alpha1,alpha2)(v1[i],v2[j])
#alpha1 and alpha2 have to be positive, otherwise gamma cdf is
not valid
tcCopulaSur<-function(v1,v2,alpha1,alpha2) {
      #Stops process if alpha1<=0 or alpha2<=0
      if(alpha1<=0 | alpha2<=0) {
            stop("alpha1 and alpha2 are not positive")
      }
      #Creates ans matrix
      ans<-matrix(0,length(v1),length(v2))
      #Applies Equation in Theorem reference 3.2.1 with u_1
=v1[i], u_2=v2[i]
      for(i in 1:length(v1)) {
            for(j in 1:length(v2)) {
                  if(v1[i]==0 && v2[j]<1){
                        ans[i,j]=0
                  }else if(v1[i]<1 && v2[j]==0) {
                        ans[i,j]=0
                  }else if(v2[j]==1){
                        ans[i,j]=v1[i]
                  }else if(v1[i]==1){
                        ans[i,j]=v2[j]
                  } else{
                        f<-function(x) {
                              pgamma(x/((1-
v1[i])^(-1/alpha1)-1),shape=alpha1,scale=1)*pgamma(x/((1-
v2[j])^(-1/alpha2)-1),shape=alpha2,scale=1)*exp(-x)
                        }
                        ans[i,j]<-v1[i]+v2[j]-1
+integrate(f,lower=0,upper=Inf)$value
                  }
            }
      }
      #Returns ans matrix
      return(ans)
}

######## Approximation function for Two-component copula Upper
tail dependence #############

#As there is no explicit form for Lambda_U for the Two-
component Copula, this function estimated by Z(u,u)/u, without
```

```
the limit)
#This function evaluates Lambda_U(t) fo a particular value t
(t is not a vector).
lambdaTcZ<-function(t,alpha1,alpha2) {
      #Check t is not a vector, if not stop process
      if(length(t)!=1) {
            stop("t has a length greater than 1")
      }
      #Integrand of Z
      function1 <- function(x) {
            ans<-
pgamma(x/((t)^(-1/alpha1)-1),shape=alpha1,scale=
1)*pgamma(x/((t)^(-1/alpha2)-1),shape=alpha2,scale=1)*exp(-x)
                  return(ans)}
      #Integral of Z intergrand
      intfunct<-integrate(function1,lower=0,upper=Inf)$value
      lambdaTcZAns<-intfunct/t
      return(lambdaTcZAns)}

#Evaluates lambdaTcZ(t) (Z(t,t)/t) for multiple values (t can
be a vector)
lambdaTcZVector<-function(t,alpha1,alpha2) {
      ans<-numeric(length=length(t))
      for(i in 1:length(t)){
            ans[i]<-lambdaTcZ(t[i],alpha1,alpha2)
            }
      return(ans)
      }


######## Two-component copula analysis Functions ############

### z: The double derivative of the monotone decreasing
transformation of the Two-component copula - Survival copula
derivative

#Using Equation reference (3.5)
DecTwoCompz<-function(u,v,alpha1,alpha2) {
  ans<-(u^(-(1/alpha1+1)))*(v^(-(1/alpha2+
1)))*((u^(-1/alpha1)-1)^alpha2)*((v^(-1/alpha2)-1)^alpha1)
  ans<-ans/((alpha1+alpha2+1)*beta(alpha1+1,alpha2+
1)*((u^(-1/alpha1))*(v^(-1/alpha2))-1)^(alpha1+alpha2+1))
  return(ans)
}



## z: Surface function (Equation reference 3.3)
#Produces a length(v1) x length(v2) matrix 'ans' s.t.
ans[i,j]=DecTwoCompz(v1[i],v2[j],alpha1,alpha2)
DecTwoCompzSur<-function(v1,v2,alpha1,alpha2) {
      #Creates ans matrix
      ans<-matrix(0,length(v1),length(v2))
      for(i in 1:length(v1)) {
            for(j in 1:length(v2)) {
                  ans[i,j]<-
DecTwoCompz(v1[i],v2[j],alpha1,alpha2)
```

```
        }
    }
    #Returns ans matrix
    return(ans)
}


## c: The double derivative of the Two-component copula (uses
Remark 3.3 - derivating survival copula)
TwoCompc<-function(u,v,alpha1,alpha2) {
  ans<-DecTwoCompz(1-u,1-v,alpha1,alpha2)
  return(ans)
}

## c: Surface function (uses Remark 3.3 - derivating survival
copula)
#Produces a length(v1) x length(v2) matrix 'ans' s.t.
ans[i,j]=DecTwoCompz(1-v1[i],1-v2[j],alpha1,alpha2)
TwoCompcSur<-function(v1,v2,alpha1,alpha2) {
  #Creates ans matrix
  ans<-matrix(0,length(v1),length(v2))
  for(i in 1:length(v1)) {
    for(j in 1:length(v2)) {
      ans[i,j]<-TwoCompc(v1[i],v2[j],alpha1,alpha2)
    }
  }
  #Returns ans matrix
  return(ans)
}


##############################################################
####
#Model Construction
##############################################################
####

#Constructing Two-component model using method in chapter 3.1
xi<-runif(2,min=0,max=1) #Generate random xi value for X_1 and
X_2
alpha1<-1/xi[1]
alpha2<-1/xi[2]
sigma1<-1
sigma2<-0.9
n<-1000 #Number of iid observations
w<-rexp(n,rate=1) #n iid observations of W
y1<-rgamma(n,alpha1, rate=1) #n iid observations of (Y_1)^(-1)
y2<-rgamma(n,alpha2, rate=1) #n iid observations of (Y_2)^(-1)
y1<-1/y1 #n iid observations of Y_1
y2<-1/y2 #n iid observations of Y_2
x1<-sigma1*w*y1 #n iid observations of X_1 using Equation
reference (3.1)
x2<-sigma2*w*y2 #n iid observations of X_2 using Equation
reference (3.2)
```

```
#################################################################
####
#Correlation measures
#################################################################
####
#Finding correlation coefficients
rho<-cor(x1,x2, method="pearson")
rhos<-cor(x1,x2,method="spearman")
tau<-cor.fk(x1,x2)
rho #Pearson's correlation
rhos #Spearman's rho
tau #Kendall's tau

#################################################################
####
#Goodness of Fit tests
#################################################################
####
#Create transformed sample using Equation reference (2.3)
Fn1<-ecdf(x1)
Fn2<-ecdf(x2)
u1<-(n/(n+1))*Fn1(x1)
u2<-(n/(n+1))*Fn2(x2)
#tau2<-cor.fk(u1,u2) #Do not need to calculate tau again as
Kendall's tau is invariant under strictly increasing
transformations, so cor.fk(u1,u2)=cor.fk(x1,x2)


###Goodness of fit test for the Gaussian Copula
#Get 'time' to check the processing time of the test at the
end.
time<-proc.time()
R<-sin(pi*tau/2) #Get R_12 value and name it R
#Check if R_12 is valid for a Gaussian copula, continue with
test only if it is. If it is not valid end test and print
reason
if(R!=1) {
      #Estimate test statistic using Equation reference (2.4)
      rhoHatGau<-sum((empCopula(u1,u2,u1,u2,n)-
gauCopula(u1,u2,tau))^2)
      k=1000 #Number of iterations in Step 5 of the bootstrap
method
      covmat<-matrix(c(1,R,R,1),2) #Make R matrix and name it
covmat
      rhoGau<-numeric(k) #A vector to contain the test
statistics from Step 5 of the bootstrap method
      #Step 5 in bootstrap method (parametric bootstrap)
      for(i in 1:k) {
            real<-rcopula.gauss(n,Sigma=covmat) #Get n iid
samples from the best fitting Gaussian copula

            #Calculate the associate transformed sample of
'real'
            Fn01<-ecdf(real[,1])
            Fn02<-ecdf(real[,2])
            u01<-(n/(n+1))*Fn01(real[,1])
            u02<-(n/(n+1))*Fn02(real[,2])
```

```
            #Estimate Kendall's tau for the transformed sample
of 'real'
            tau0<-cor.fk(u01,u02)
            #Check to see if tau0 will estimate a valid Gaussian
distribution. If tau0 is valid estimate the test statistic of
this iteration and store it in ith position of rhoGau, if it
is not valid store 'NA' in the ith position of rhoGau.
            if(1>tau0) {
                  rhoGau[i]<-sum((empCopula(u01,u02,u01,u02,n)-
gauCopula(u01,u02,tau0))^2)
            }else{
                  rhoGau[i]<-NA
            }
      }
      val<-length(which(!is.na(rhoGau))) #Number of valid
iterations
      #Estimate the pvalue of the test using Step 6 of the
Bootstrap method
      pvalueGau<-(1/(val+1))*sum(as.numeric(rhoGau[which(!
is.na(rhoGau))]>=rhoHatGau))
      pvalueGau
} else {
      paste("R_12 estimate does not produce a valid Gaussian
distribution")
}
#Display processing time of Gaussian GoF test
proc.time()-time



###Goodness of fit test for the Gumbel Copula
time<-proc.time()
theta<-1/(1-tau) #Estimate theta value of best fitting Gumbel
copula

#Check if theta is valid for a Gumbel copula, continue with
test only if it is. If it is not valid end test and print
reason.
if(theta>=1&&theta<Inf) {
      #Estimate test statistic using Equation reference (2.4)
      rhoHatGum<-sum((empCopula(u1,u2,u1,u2,n)-
gumCopula(u1,u2,tau))^2)
      k=1000
      rhoGum<-numeric(k) #A vector to contain the test
statistics from Step 5 of the bootstrap method
      #Step 5 in bootstrap method (parametric bootstrap)
      for(i in 1:k) {
            real<-rCopula(n,gumbelCopula(theta,2)) #Get n iid
samples from the best fitting Gumbel copula

            #Calculate the associate transformed sample of
'real'
            Fn01<-ecdf(real[,1])
            Fn02<-ecdf(real[,2])
            u01<-(n/(n+1))*Fn01(real[,1])
            u02<-(n/(n+1))*Fn02(real[,2])
            #Estimate Kendall's tau for the transformed sample
```

```
of 'real'
            tau0<-cor.fk(u01,u02)
            #Check to see if tau0 will estimate a valid Gumbel
distribution. If tau0 is valid estimate the test statistic of
this iteration and store it in ith position of rhoGum, if it
is not valid store 'NA' in the ith position of rhoGum.
            if(1>tau0&&tau0>=0) {
                  rhoGum[i]<-sum((empCopula(u01,u02,u01,u02,n)-
gumCopula(u01,u02,tau0))^2)
            }else{
                  rhoGum[i]<-NA
            }
      }
      val<-length(which(!is.na(rhoGum))) #Number of valid
iterations
      #Estimate the pvalue of the test using Step 6 of the
Bootstrap method
      pvalueGum<-(1/(val+1))*sum(as.numeric(rhoGum[which(!
is.na(rhoGum))]>=rhoHatGum))
      pvalueGum
} else {
      paste("theta estimate does not produce a valid Gumbel
distribution")
}
#Display processing time of Gumbel GoF test
proc.time()-time



###Goodness of fit test for the Clayton Copula
time<-proc.time()
theta<-2*tau/(1-tau) #Estimate theta value of best fitting
Clayton copula

#Check if theta is valid for a Clayton copula, continue with
test only if it is. If it is not valid end test and print
reason.
if(theta>0&&theta<Inf) {
      #Estimate test statistic using Equation reference (2.4)
      rhoHatCla<-sum((empCopula(u1,u2,u1,u2,n)-
claCopula(u1,u2,tau))^2)
      k=1000
      rhoCla<-numeric(k) #A vector to contain the test
statistics from Step 5 of the bootstrap method
      #Step 5 in bootstrap method (parametric bootstrap)
      for(i in 1:k) {
            real<-rCopula(n,claytonCopula(theta,2)) #Get n iid
samples from the best fitting Clayton copula

            #Calculate the associate transformed sample of
'real'
            Fn01<-ecdf(real[,1])
            Fn02<-ecdf(real[,2])
            u01<-(n/(n+1))*Fn01(real[,1])
            u02<-(n/(n+1))*Fn02(real[,2])
            #Estimate Kendall's tau for the transformed sample
of 'real'
```

```
            tau0<-cor.fk(u01,u02)
            #Check to see if tau0 will estimate a valid clayton
distribution. If tau0 is valid estimate the test statistic of
this iteration and store it in ith position of rhoGum, if it
is not valid store 'NA' in the ith position of rhoGum.
            if(1>tau0&&tau0>=0) {
                  rhoCla[i]<-sum((empCopula(u01,u02,u01,u02,n)-
claCopula(u01,u02,tau0))^2)
            }else{
                  rhoCla[i]<-NA
            }
      }
      val<-length(which(!is.na(rhoCla))) #Number of valid
iterations
      #Estimate the pvalue of the test using Step 6 of the
Bootstrap method
      pvalueCla<-(1/(val+1))*sum(as.numeric(rhoCla[which(!
is.na(rhoCla))]>=rhoHatCla))
      pvalueCla
} else {
      paste("theta estimate does not produce a valid Clayton
distribution")
}
#Display processing time of Clayton GoF test
proc.time()-time


###Goodness of fit test for the Two-component copula
time<-proc.time()
#Get alpha parameter estimates (only alpha is needed in
copula)
threshold = 0
gdata = data.frame(y = x1)
fit = vglm(y ~ 1, gpd(threshold = threshold), gdata, trace =
TRUE) #Get the best fitting GPD of X1 from dataset x1
alphaEst1<-1/Coef(fit)[2] #Get MLE of alpha_1
gdata = data.frame(y = x2)
fit = vglm(y ~ 1, gpd(threshold = threshold), gdata, trace =
TRUE) #Get the best fitting GPD of X1 from dataset x1
alphaEst2<-1/Coef(fit)[2] #Get MLE of alpha_2

#Check if alpha1 and alpha2 are valid for a Two-component
copula, continue with test only if they are. If they are not
valid end test and print reason
if(alphaEst1>0 && alphaEst2>0) {
      #Estimate test statistic using Equation reference (2.12)
      rhoHatTc<-sum((empCopula(u1,u2,u1,u2,n)-
tcCopula(u1,u2,alphaEst1,alphaEst2))^2)

      sigmaEst1<-1 #Arbitrary sigma_1 for parametric bootstrap
data - arbitrary as Two-component copula is independent of
sigma_1
      sigmaEst2<-1 #Arbituary sigma_2 for parametric bootstrap
data - arbitrary as Two-component copula is independent of
sigma_2
      k=1000
      rhoTc<-numeric(k) #A vector to contain the test
```

```
statistics from Step 5 of the bootstrap method
      alphaEstGof1<-numeric(k) #A vector to contain the alpha1 
values from which iteration in Step 5 of the bootstrap method
      alphaEstGof2<-numeric(k) #A vector to contain the alpha2 
values from which iteration in Step 5 of the bootstrap method
      #Step 5 in bootstrap method (parametric bootstrap)
      for(i in 1:k) {
            #Generate n idd samples from the best fitting Two-
component copula. This is done by using the Two-component 
model to produce data (x01,x02) that has the required Two-
component copula.
            w0<-rexp(n,rate=1)
            y01<-rgamma(n,alphaEst1, rate=1)
            y02<-rgamma(n,alphaEst2, rate=1)
            y01<-1/y01
            y02<-1/y02
            x01<-sigmaEst1*w0*y01
            x02<-sigmaEst2*w0*y02
            Fn01<-ecdf(x01)
            Fn02<-ecdf(x02)
            #Transform the sample using Equation reference 
(2.11)
            u01<-(n/(n+1))*Fn01(x01)
            u02<-(n/(n+1))*Fn02(x02)

            #Estimate the alpha parameters for this new dataset
            gdata = data.frame(y = x01)
            fit = vglm(y ~ 1, gpd(threshold = threshold), gdata, 
trace = TRUE)
            alphaEst01<-1/Coef(fit)[2]
            gdata = data.frame(y = x02)
            fit = vglm(y ~ 1, gpd(threshold = threshold), gdata, 
trace = TRUE)
            alphaEst02<-1/Coef(fit)[2]
            alphaEstGof1[i]<-alphaEst01
            alphaEstGof2[i]<-alphaEst02


            #Check to see if alpha parameters will estimate a 
valid Two component copula distribution. If they are valid 
estimate the test statistic of this iteration and store it in 
ith position of rhoTc, if they are not valid store 'NA' in the 
ith position of rhoTc.
            if(alphaEst01>0 && alphaEst02>0) {
                  rhoTc[i]<-sum((empCopula(u01,u02,u01,u02,n)-
tcCopula(u01,u02,alphaEst01,alphaEst02))^2)
            }else{
                  rhoTc[i]<-NA
            }

      }
      val<-length(which(!is.na(rhoTc))) #Number of valid 
iterations
      #Estimate the pvalue of the test using Step 6 of the 
Bootstrap method
      pvalueTc<-(1/(val+1))*sum(as.numeric(rhoTc[which(!
is.na(rhoTc))]>=rhoHatTc))
```

```
      pvalueTc
} else {
      paste("alpha estimates are not positive - marginal
distributions are not Pareto Type II")
}
#Display processing time of Two-component copula GoF test
proc.time()-time

#Pvalues and Histograms of statistic distribution.
pvalueGau
pvalueGum
pvalueCla
pvalueTc

#A plot containing 4 histograms. The histograms show the
distribution of the test statistic under the null hypothesis
of each goodness of fit test respectively, and have the
observed test statistic from the dataset marked with a red
line.
par(mfrow=c(4,1))
hist(rhoGau,xlim=c((min(rhoGau[which(!
is.na(rhoGau))],rhoHatGau)-0.1),(max(rhoGau[which(!
is.na(rhoGau))],rhoHatGau)+0.1)),main="Histogram of Test
Statistics obtained in the Gaussian Copula GoF test",
xlab="Test statistic value")
abline(v=rhoHatGau, col=2)
hist(rhoGum,xlim=c((min(rhoGum[which(!
is.na(rhoGum))],rhoHatGum)-0.1),(max(rhoGum[which(!
is.na(rhoGum))],rhoHatGum)+0.1)),main="Histogram of Test
Statistics obtained in the Gumbel Copula GoF test", xlab="Test
statistic value")
abline(v=rhoHatGum, col=2)
hist(rhoCla,xlim=c((min(rhoCla[which(!
is.na(rhoCla))],rhoHatCla)-0.1),(max(rhoCla[which(!
is.na(rhoCla))],rhoHatCla)+0.1)),main="Histogram of Test
Statistics obtained in the Clayton Copula GoF test",
xlab="Test statistic value")
abline(v=rhoHatCla, col=2)
hist(rhoTc, xlim=c((min(rhoTc[which(!
is.na(rhoTc))],rhoHatTc)-0.1),(max(rhoTc[which(!
is.na(rhoTc))],rhoHatTc)+0.1)),main="Histogram of Test
Statistics obtained in the Two-component Copula GoF test",
xlab="Test statistic value")
abline(v=rhoHatTc, col=2)


##############################################################
####
#3D surface plots of copula fits
##############################################################
####
#Set the view of the 3D plot we wish to observe (two
perspectives used in this study, uncomment the usermatrix of
the perspective we would like to use)
#userMatrix<-matrix(c(0.6968547, 0.3058737, -0.6487178,
0, -0.7170246, 0.3178042, -0.6203839, 0, 0.01640616,
0.89746410, 0.44078234, 0, 0, 0, 0, 1),4)
```

```
userMatrix<-matrix(c(0.7143918, -0.2573741,
0.6506941,0, -0.6997451, -0.2639421, 0.6638458, 0,
0.000888928, -0.929565966, -0.368654341, 0, 0, 0, 0, 1),4)
view3d(userMatrix=userMatrix) #Set the perspective

#Create 3d plot showing the empirical copula and the best
fitting Gaussian copula of our data set
x<-seq(0.001,0.999,by=0.001) #x-axis values
y<-x #y-axis values
zemp<-empCopulaSur(x,y,u1,u2,n) #Get the empirical copula
values
zgau<-gauCopulaSur(x,y,tau) #Get the best fitting Gaussian
copula values
plot3d(x,y,x,type="n",xlabel="x",ylabel="y",zlab="copula",
main="Gaussian Copula fit") #Create an empty plot of the unit
cube with a suitable title and axis labels
surface3d(x,y,zemp,col="green") #Plot the empirical copula
surface3d(x,y,zgau,col="blue") #Plot the best fitting Gaussian
copula

#Create 3d plot showing the empirical copula and the best
fitting Gumbel copula of our data set
#x<-seq(0.001,0.999,by=0.001)
#y<-x
#zemp<-empCopulaSur(x,y,u1,u2,n)
zgum<-gumCopulaSur(x,y,tau) #Get the best fitting Gumbel
copula values
plot3d(x,y,x,type="n",xlabel="x",ylabel="y",zlab="copula",main
="Gumbel Copula fit") #Create an empty plot of the unit cube
with a suitable title and axis labels
surface3d(x,y,zemp,col="green") #Plot the empirical copula
surface3d(x,y,zgum,col="blue") #Plot the best fitting Gumbel
copula

#Create 3d plot showing the empirical copula and the best
fitting Clayton copula of our data set
#x<-seq(0.001,0.999,by=0.001)
#y<-x
#zemp<-empCopulaSur(x,y,u1,u2,n)
zcla<-claCopulaSur(x,y,tau) #Get the best fitting Clayton
copula values
plot3d(x,y,x,type="n",xlabel="x",ylabel="y",zlab="copula",main
="Clayton Copula fit") #Create an empty plot of the unit cube
with a suitable title and axis labels
surface3d(x,y,zemp,col="green") #Plot the empirical copula
surface3d(x,y,zcla,col="blue") #Plot the best fitting Clayton
copula

#Create 3d plot of the Two component model Copula
#x<-seq(0.001,0.999,by=0.001)
#y<-x
#zemp<-empCopulaSur(x,y,u1,u2,n)
#Get alpha parameter estimates (only alpha is needed in
copula) - for an explanation of the code see Two-component GoF
test
threshold = 0
gdata = data.frame(y = x1)
```

```
fit = vglm(y ~ 1, gpd(threshold = threshold), gdata, trace =
TRUE)
alphaEst1<-1/Coef(fit)[2]
gdata = data.frame(y = x2)
fit = vglm(y ~ 1, gpd(threshold = threshold), gdata, trace =
TRUE)
alphaEst2<-1/Coef(fit)[2]

ztc<-tcCopulaSur(x,y,alphaEst1,alphaEst2) #Get the best
fitting Two-component copula values
plot3d(x,y,x,type="n",xlabel="x",ylabel="y",zlab="copula",main
="Two-Component Model Copula") #Create an empty plot of the
unit cube with a suitable title and axis labels
surface3d(x,y,zemp,col="green") #Plot the empirical copula
surface3d(x,y,ztc,col="blue") #Plot the best fitting Gaussian
copula



##############################################################
###
#Extreme Value test
##############################################################
###
evTestK(matrix(c(x1,x2),n,2),method="fsample") #Apply the
extreme value test to the dataset


##############################################################
####
#Upper Dependence Value
##############################################################
####

#Gumbel Copula
lambdaGum<-2-2^(1-tau) #Estimated Lambda_U for Gumbel Copula
lambdaGum
#Two-component Copula - As we do not have an explicit form for
Lambda_U, we will plot the function Z(t,t)/t in a range close
to 0
alphalam1<-alpha1 #Alpha1 value #1.320427
alphalam2<-alpha2 #Alpha2 value #5.478712
taxis<-seq(0,0.01, by=0.00001)
lambda_Uaxis<-lambdaTcZVector(taxis,alphalam1,alphalam2)
plot(taxis,lambda_Uaxis,type="l",main=paste("Plot to estimate
Lambda_U for the Two-component copula with (alpha1,alpha2)=(",
alphalam1,",",alphalam2,")"),xlab="t",ylab="lambda_U(t)")

##############################################################
####
#Lower Dependence Value
##############################################################
####

#Clayton Copula
lambda_LCla<-2^(0.5*(1-tau)) #Estimated Lambda_U for Gumbel
Copula
```

```
lambda_LCla


####################################################################
#Analysis of the two-component copula
####################################################################

####################################
######  Check integration of z (Equation reference 3.3) and c
(double deribative of two component copula) over unit square
equals 1 - this is a requirement as z and c are also the joint
density functions of two uniform random variables with copula
Z and C respectively, evaulated on (0,1).

DenAlpha1<-1 #Arbituary alpha1 value
DenAlpha2<-1 #Arbituary alpha2 value
#Create function to represent the joint density function (on
(0,1)) of two uniform random variables with copula Z with
alpha1=DenAlpha1 and alpha2=DenAlpha2 (uses Equation reference
3.6)
func<-function(z) {
      ans<-DecTwoCompz(z[1],z[2],DenAlpha1,DenAlpha2)
      return(ans)
      }
#Integrate density over the unit square
adapt(2,lo=c(0,0),up=c(1,1), functn=func)


#Create function to represent the joint density function (on
(0,1)) of two uniform random variables with copula C with
alpha1=DenAlpha1 and alpha2=DenAlpha2 (uses Equation reference
(3.6))
func<-function(z) {
  ans<-TwoCompc(z[1],z[2],DenAlpha1,DenAlpha2)
  return(ans)}
#Integrate density over the unit square
adapt(2,lo=c(0,0),up=c(1,1), functn=func)


################################
####Plotting Density function

u<-seq(0.001,0.999,by=0.001) #u-axis values
v<-u #v-axis values

##Plotting the colour palette of the colours used in colouring
the graphs.
#In this function n=number of points plotted in graph, so
n=length(u)*length(v)
demo.pal(length(u)*length(v))


##Plotting c(u,v) with alpha1 =a and alpha2=b (Equation
reference 3.6)
a=0.5
```

```
b=0.7
cvalues<-TwoCompcSur(u,v,a,b)
col<-rainbow(length(cvalues))[rank(cvalues)]
plot3d(u,v,
cvalues,type="n",xlabel="u",ylabel="v",zlab="c(u,v)",main=past
e("Plot of c(u,v) (alpha1=",a,", alpha2=",b,")"))
surface3d(u,v, cvalues,col=col)

###Plotting density c(u,v)<1 with alpha1 =a and alpha2=b
(Equation reference 3.6)
cvalues2<-cvalues
cvalues2[which(cvalues2>1)]<-NA
col<-rainbow(length(cvalues2))[rank(cvalues2)]
#usermatrix<-par3d("userMatrix") ###Saves userMatrix for plot
that you have hand rotated into the right position
#Change perspective of plot
usermatrix<-matrix(c(0.7122523, 0.2930976, -0.6378013,
0, -0.7016941, 0.3205379, -0.6363025, 0, 0.01794076,
0.90074944, 0.43396875, 0,0,0,0,1),4)
view3d(userMatrix=usermatrix)
#Create empty plot with suitable title and axis labels
plot3d(u,v,u,type="n",xlabel="x",ylabel="y",zlab="copula",
main=paste("Plot of the function c(u,v) contained in the unit
cube (alpha1=",a,", alpha2=",b,")"), axes=FALSE)
axes3d(c('x--','y--','z-+')) #Changes the position of the axis
labels to improve the look of the plot
surface3d(u,v, cvalues2,col=col) #plots the graph


###################################
#####Plotting the contour plots of density function

u<-seq(0.001,0.999,by=0.001) #u-axis values
v<-u #v-axis values


##Plotting c(u,v) with alpha1 =a and alpha2=b (Equation
reference 3.6)
a=1
b=35
cvalues<-TwoCompcSur(u,v,a,b) #Calcules c values (density
function)
zvalues<-DecTwoCompzSur(u,v,a,b) #Calcules z values (survival
copula function)
#contlevels=c(seq(0,2,by=0.2),3,4)
contlevels=c(seq(0,2,by=0.2),seq(0,6,by=1))
contour(u,v, cvalues, levels=contlevels,
plot.title=title(main=paste("Plot of c(u,v) (alpha1=",a,",
alpha2=",b,")"),xlab="u",ylab="v"))
contour(u,v, zvalues, levels=contlevels,
plot.title=title(main=paste("Plot of z(u,v) (alpha1=",a,",
alpha2=",b,")"),xlab="u",ylab="v"))
```