Package 'gamCopula'

May 8, 2026

Type Package

Title Generalized Additive Models for Bivariate Conditional Dependence Structures and Vine Copulas

Version 0.0-8

Encoding UTF-8

Date 2025-04-02

Maintainer Thibault Vatter <thibault.vatter@gmail.com>

Depends R (>= 3.1.0)

Imports graphics, stats, utils, VineCopula (>= 2.0.0), mgcv, MASS, gsl, numDeriv, methods, copula, igraph (>= 1.0.0), parallel, foreach, doParallel

Description Implementation of various inference and simulation tools to apply generalized additive models to bivariate dependence structures and non-simplified vine copulas.

License GPL-3

LazyLoad yes

BugReports https://github.com/tvatter/gamCopula/issues

URL https://github.com/tvatter/gamCopula

RoxygenNote 7.3.2

NeedsCompilation no

Author Thomas Nagler [aut], Thibault Vatter [aut, cre]

Repository CRAN

Date/Publication 2025-04-02 12:30:07 UTC

Contents

`gamCopula-package` *gamCopula: Generalized Additive Models for Bivariate Conditional Dependence Structures and Vine Copulas*

Description

This package implements inference and simulation tools to apply generalized additive models to bivariate dependence structures and vine copulas.

Details

More details can be found in Vatter and Chavez-Demoulin (2015) and Vatter and Nagler (2016).

Package:	gamCopula
Type:	Package
Version:	0.0-8
Date:	2025-04-02
License:	GPL-3

Author(s)

Thibault Vatter and Thomas Nagler

Maintainer: Thibault Vatter <thibault.vatter@gmail.com>

References

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

Brechmann, E. C., C. Czado, and K. Aas (2012) Truncated regular vines in high dimensions with applications to financial data. *Canadian Journal of Statistics*, 40(1), 68–85.

Dissmann, J. F., E. C. Brechmann, C. Czado, and D. Kurowicka (2013) Selecting and estimating regular vine copulae and application to financial returns. *Computational Statistics & Data Analysis*, 59(1), 52–69.

Vatter, T. and V. Chavez-Demoulin (2015) Generalized Additive Models for Conditional Dependence Structures. *Journal of Multivariate Analysis*, 141, 147–167.

Vatter, T. and T. Nagler (2016) Generalized additive models for non-simplified pair-copula constructions. https://arxiv.org/abs/1608.01593

Wood, S. N. (2004) Stable and efficient multiple smoothing parameter estimation for generalized additive models. *Journal of the American Statistical Association*, 99, 673–686.

Wood, S. N. (2006) Generalized Additive Models: an introduction with R. Chapman and Hall/CRC.

See Also

The present package heavily relies on the **mgcv** and **VineCopula** packages, as it basically extends and mixes both of them.

Examples

```
##### A gamBiCop example
require(copula)
require(mgcv)
set.seed(0)

## Simulation parameters (sample size, correlation between covariates,
## Gaussian copula family)
n <- 5e2
rho <- 0.5
fam <- 1

### A calibration surface depending on three variables
```

```
eta0 <- 1
calib.surf <- list(
  calib.quad <- function(t, Ti = 0, Tf = 1, b = 8) {
    Tm <- (Tf - Ti) / 2
    a <- -(b / 3) * (Tf^2 - 3 * Tf * Tm + 3 * Tm^2)
    return(a + b * (t - Tm)^2)
  },
  calib.sin <- function(t, Ti = 0, Tf = 1, b = 1, f = 1) {
    a <- b * (1 - 2 * Tf * pi / (f * Tf * pi +
      cos(2 * f * pi * (Tf - Ti))
      - cos(2 * f * pi * Ti)))
    return((a + b) / 2 + (b - a) * sin(2 * f * pi * (t - Ti)) / 2)
  },
  calib.exp <- function(t, Ti = 0, Tf = 1, b = 2, s = Tf / 8) {
    Tm <- (Tf - Ti) / 2
    a <- (b * s * sqrt(2 * pi) / Tf) * (pnorm(0, Tm, s) - pnorm(Tf, Tm, s))
    return(a + b * exp(-(t - Tm)^2 / (2 * s^2)))
  }
)
### Display the calibration surface
par(mfrow = c(1, 3), pty = "s", mar = c(1, 1, 4, 1))
u <- seq(0, 1, length.out = 100)
sel <- matrix(c(1, 1, 2, 2, 3, 3), ncol = 2)
jet.colors <- colorRamp(c(
  "#00007F", "blue", "#007FFF", "cyan", "#7FFF7F",
  "yellow", "#FF7F00", "red", "#7F0000"
))
jet <- function(x) {
  rgb(jet.colors(exp(x / 3) / (1 + exp(x / 3))),
    maxColorValue = 255
  )
}
for (k in 1:3) {
  tmp <- outer(u, u, function(x, y) {
    eta0 + calib.surf[[sel[k, 1]]](x) + calib.surf[[sel[k, 2]]](y)
  })
  persp(u, u, tmp,
    border = NA, theta = 60, phi = 30, zlab = "",
    col = matrix(jet(tmp), nrow = 100),
    xlab = paste("X", sel[k, 1], sep = ""),
    ylab = paste("X", sel[k, 2], sep = ""),
    main = paste("eta0+f", sel[k, 1],
      "(X", sel[k, 1], ") +f", sel[k, 2],
      "(X", sel[k, 2], ")",
      sep = ""
    )
  )
}
### 3-dimensional matrix X of covariates
covariates.distr <- mvdc(normalCopula(rho, dim = 3),
  c("unif"), list(list(min = 0, max = 1)),
  marginsIdentical = TRUE
)
```

```
X <- rMvdc(n, covariates.distr)
### U in [0,1]x[0,1] with copula parameter depending on X
U <- condBiCopSim(fam, function(x1, x2, x3) {
  eta0 + sum(mapply(function(f, x) {
    f(x)
  }, calib.surf, c(x1, x2, x3)))
}, X[, 1:3], par2 = 6, return.par = TRUE)
### Merge U and X
data <- data.frame(U$data, X)
names(data) <- c(paste("u", 1:2, sep = ""), paste("x", 1:3, sep = ""))
### Display the data
dev.off()
plot(data[, "u1"], data[, "u2"], xlab = "U1", ylab = "U2")
### Model fit with a basis size (arguably) too small
### and unpenalized cubic spines
pen <- FALSE
basis0 <- c(3, 4, 4)
formula <- ~ s(x1, k = basis0[1], bs = "cr", fx = !pen) +
  s(x2, k = basis0[2], bs = "cr", fx = !pen) +
  s(x3, k = basis0[3], bs = "cr", fx = !pen)
system.time(fit0 <- gamBiCopFit(data, formula, fam))
### Model fit with a better basis size and penalized cubic splines (via min GCV)
pen <- TRUE
basis1 <- c(3, 10, 10)
formula <- ~ s(x1, k = basis1[1], bs = "cr", fx = !pen) +
  s(x2, k = basis1[2], bs = "cr", fx = !pen) +
  s(x3, k = basis1[3], bs = "cr", fx = !pen)
system.time(fit1 <- gamBiCopFit(data, formula, fam))
### Extract the gamBiCop objects and show various methods
(res <- sapply(list(fit0, fit1), function(fit) {
  fit$res
}))
metds <- list("logLik" = logLik, "AIC" = AIC, "BIC" = BIC, "EDF" = EDF)
lapply(res, function(x) sapply(metds, function(f) f(x)))
### Comparison between fitted, true smooth and spline approximation for each
### true smooth function for the two basis sizes
fitted <- lapply(res, function(x) {
  gamBiCopPredict(x, data.frame(x1 = u, x2 = u, x3 = u),
    type = "terms"
  )$calib
})
true <- vector("list", 3)
for (i in 1:3) {
  y <- eta0 + calib.surf[[i]](u)
  true[[i]]$true <- y - eta0
  temp <- gam(y ~ s(u, k = basis0[i], bs = "cr", fx = TRUE))
  true[[i]]$approx <- predict.gam(temp, type = "terms")
  temp <- gam(y ~ s(u, k = basis1[i], bs = "cr", fx = FALSE))
  true[[i]]$approx2 <- predict.gam(temp, type = "terms")
}
### Display results
par(mfrow = c(1, 3), pty = "s")
yy <- range(true, fitted)
```

```
yy[1] <- yy[1] * 1.5
for (k in 1:3) {
  plot(u, true[[k]]$true,
    type = "l", ylim = yy,
    xlab = paste("Covariate", k), ylab = paste("Smooth", k)
  )
  lines(u, true[[k]]$approx, col = "red", lty = 2)
  lines(u, fitted[[1]][, k], col = "red")
  lines(u, fitted[[2]][, k], col = "green")
  lines(u, true[[k]]$approx2, col = "green", lty = 2)
  legend("bottomleft",
    cex = 0.6, lty = c(1, 1, 2, 1, 2),
    c("True", "Fitted", "Appox 1", "Fitted 2", "Approx 2"),
    col = c("black", "red", "red", "green", "green")
  )
}
###### A gamVine example
set.seed(0)
###  Simulation parameters
## Sample size
n <- 1e3
## Copula families
familyset <- c(1:2, 301:304, 401:404)
## Define a 4-dimensional R-vine tree structure matrix
d <- 4
Matrix <- c(2, 3, 4, 1, 0, 3, 4, 1, 0, 0, 4, 1, 0, 0, 0, 1)
Matrix <- matrix(Matrix, d, d)
nnames <- paste("X", 1:d, sep = "")
### A function factory
eta0 <- 1
calib.surf <- list(
  calib.quad <- function(t, Ti = 0, Tf = 1, b = 8) {
    Tm <- (Tf - Ti) / 2
    a <- -(b / 3) * (Tf^2 - 3 * Tf * Tm + 3 * Tm^2)
    return(a + b * (t - Tm)^2)
  },
  calib.sin <- function(t, Ti = 0, Tf = 1, b = 1, f = 1) {
    a <- b * (1 - 2 * Tf * pi / (f * Tf * pi +
      cos(2 * f * pi * (Tf - Ti))
      - cos(2 * f * pi * Ti)))
    return((a + b) / 2 + (b - a) * sin(2 * f * pi * (t - Ti)) / 2)
  },
  calib.exp <- function(t, Ti = 0, Tf = 1, b = 2, s = Tf / 8) {
    Tm <- (Tf - Ti) / 2
    a <- (b * s * sqrt(2 * pi) / Tf) * (pnorm(0, Tm, s) - pnorm(Tf, Tm, s))
    return(a + b * exp(-(t - Tm)^2 / (2 * s^2)))
  }
)
###  Create the model
## Define gam-vine model list
count <- 1
model <- vector(mode = "list", length = d * (d - 1) / 2)
sel <- seq(d, d^2 - d, by = d)
```

```
## First tree
for (i in 1:(d - 1)) {
  # Select a copula family
  family <- sample(familyset, 1)
  model[[count]]$family <- family
  # Use the canonical link and a randomly generated parameter
  if (is.element(family, c(1, 2))) {
    model[[count]]$par <- tanh(rnorm(1) / 2)
    if (family == 2) {
      model[[count]]$par2 <- 2 + exp(rnorm(1))
    }
  } else {
    if (is.element(family, c(401:404))) {
      rr <- rnorm(1)
      model[[count]]$par <- sign(rr) * (1 + abs(rr))
    } else {
      model[[count]]$par <- rnorm(1)
    }
    model[[count]]$par2 <- 0
  }
  count <- count + 1
}
## A dummy dataset
data <- data.frame(u1 = runif(1e2), u2 = runif(1e2), matrix(runif(1e2 * d), 1e2, d))
## Trees 2 to (d-1)
for (j in 2:(d - 1)) {
  for (i in 1:(d - j)) {
    # Select a copula family
    family <- sample(familyset, 1)
    # Select the conditiong set and create a model formula
    cond <- nnames[sort(Matrix[(d - j + 2):d, i])]
    tmpform <- paste("~", paste(paste("s(", cond, ", k=10, bs='cr')",
      sep = ""
    ), collapse = " + "))
    l <- length(cond)
    temp <- sample(3, l, replace = TRUE)
    # Spline approximation of the true function
    m <- 1e2
    x <- matrix(seq(0, 1, length.out = m), nrow = m, ncol = 1)
    if (l != 1) {
      tmp.fct <- paste("function(x){eta0+",
        paste(sapply(1:l, function(x) {
          paste("calib.surf[[", temp[x], "]](x[", x, "])",
            sep = ""
          )
        }), collapse = "+"), "}",
        sep = ""
      )
      tmp.fct <- eval(parse(text = tmp.fct))
      x <- eval(parse(text = paste0("expand.grid(",
        paste0(rep("x", l), collapse = ","), ")",
        collapse = ""
      )))
```

```
      y <- apply(x, 1, tmp.fct)
    } else {
      tmp.fct <- function(x) eta0 + calib.surf[[temp]](x)
      colnames(x) <- cond
      y <- tmp.fct(x)
    }
    # Estimate the gam model
    form <- as.formula(paste0("y", tmpform))
    dd <- data.frame(y, x)
    names(dd) <- c("y", cond)
    b <- gam(form, data = dd)
    # plot(x[,1],(y-fitted(b))/y)
    # Create a dummy gamBiCop object
    tmp <- gamBiCopFit(data = data, formula = form, family = 1, n.iters = 1)$res
    # Update the copula family and the model coefficients
    attr(tmp, "model")$coefficients <- coefficients(b)
    attr(tmp, "model")$smooth <- b$smooth
    attr(tmp, "family") <- family
    if (family == 2) {
      attr(tmp, "par2") <- 2 + exp(rnorm(1))
    }
    model[[count]] <- tmp
    count <- count + 1
  }
}
## Create the gamVineCopula object
GVC <- gamVine(Matrix = Matrix, model = model, names = nnames)
print(GVC)
#
## Not run:
### Simulate and fit the model
sim <- gamVineSimulate(n, GVC)
fitGVC <- gamVineSeqFit(sim, GVC, verbose = TRUE)
fitGVC2 <- gamVineCopSelect(sim, Matrix, verbose = TRUE)
### Plot the results
par(mfrow = c(3, 4))
plot(GVC, ylim = c(-2.5, 2.5))
plot(fitGVC, ylim = c(-2.5, 2.5))
plot(fitGVC2, ylim = c(-2.5, 2.5))

## End(Not run)
```

`AIC.gamBiCop` *Akaike's An Information Criterion for a gamBiCop Object*

Description

Function calculating Akaike's 'An Information Criterion' (AIC) for an object of the class gamBiCop (note that the models are usually fitted by penalized likelihood maximization).

Usage

```
## S4 method for signature 'gamBiCop'
AIC(object, ..., k = 2)
```

Arguments

object	An object of the class gamBiCop.
...	un-used in this class
k	numeric, the penalty per parameter to be used; the default k = 2 is the classical AIC.

Value

A numeric value with the corresponding AIC.

See Also

AIC and BIC.

BIC.gamBiCop *Schwarz's Bayesian Information Criterion for a gamBiCop Object*

Description

Function calculating the Schwarz's Bayesian Information Criterion (BIC) for an object of the class gamBiCop (note that the models are usually fitted by penalized likelihood maximization).

Usage

```
## S4 method for signature 'gamBiCop'
BIC(object, ...)
```

Arguments

object	An object of the class gamBiCop.
...	un-used in this class

Value

A numeric value with the corresponding BIC.

See Also

AIC and BIC.

`BiCopEta2Par` *Copula Parameter of a Bivariate Copula for a Given Value of the Calibration Function*

Description

Computes the (first) copula parameter of a bivariate copula for a given value of the calibration function (eta).

Usage

```
BiCopEta2Par(family, eta)
```

Arguments

`family`	A copula family: `1` Gaussian, `2` Student t, `301` Double Clayton type I (standard and rotated 90 degrees), `302` Double Clayton type II (standard and rotated 270 degrees), `303` Double Clayton type III (survival and rotated 90 degrees), `304` Double Clayton type IV (survival and rotated 270 degrees), `401` Double Gumbel type I (standard and rotated 90 degrees), `402` Double Gumbel type II (standard and rotated 270 degrees), `403` Double Gumbel type III (survival and rotated 90 degrees), `404` Double Gumbel type IV (survival and rotated 270 degrees).
`eta`	The calibration function.

Value

The value of the first copula parameter, depending on the copula parameter and family as:

- `1` Gaussian, `f(x) = tanh(x/2)`
- `2` Student t, `f(x) = tanh(x/2)`
- `301` Double Clayton type I (standard and rotated 90 degrees), `f(x) = x`
- `302` Double Clayton type II (standard and rotated 270 degrees), `f(x) = x`
- `303` Double Clayton type III (survival and rotated 90 degrees), `f(x) = x`
- `304` Double Clayton type IV (survival and rotated 270 degrees), `f(x) = x`
- `401` Double Gumbel type I (standard and rotated 90 degrees), `f(x) = x*(1+abs(x))/abs(x)`
- `402` Double Gumbel type II (standard and rotated 270 degrees), `f(x) = x*(1+abs(x))/abs(x)`
- `403` Double Gumbel type III (survival and rotated 90 degrees), `f(x) = x*(1+abs(x))/abs(x)`
- `404` Double Gumbel type IV (survival and rotated 270 degrees) `f(x) = x*(1+abs(x))/abs(x)`.

See Also

`BiCopPar2Eta`, `BiCopPar2Tau`, and `BiCopTau2Par` from the **VineCopula** package.

`BiCopPar2Eta` *Calibration Function of a Bivariate Copula for a Given Parameter's Value*

Description

Computes the calibration function (eta) of a bivariate copula for a given value of the (first) copula parameter.

Usage

```
BiCopPar2Eta(family, par)
```

Arguments

`family`	A copula family: `1` Gaussian, `2` Student t, `301` Double Clayton type I (standard and rotated 90 degrees), `302` Double Clayton type II (standard and rotated 270 degrees), `303` Double Clayton type III (survival and rotated 90 degrees), `304` Double Clayton type IV (survival and rotated 270 degrees), `401` Double Gumbel type I (standard and rotated 90 degrees), `402` Double Gumbel type II (standard and rotated 270 degrees), `403` Double Gumbel type III (survival and rotated 90 degrees), `404` Double Gumbel type IV (survival and rotated 270 degrees).
`par`	The (first) copula parameter

Value

The value of the calibration function, depending on the copula parameter and family as:

- `1` Gaussian, `f(x) = 2*atanh(x)`
- `2` Student t, `f(x) = 2*atanh(x)`
- `301` Double Clayton type I (standard and rotated 90 degrees), `f(x) = x`
- `302` Double Clayton type II (standard and rotated 270 degrees), `f(x) = x`
- `303` Double Clayton type III (survival and rotated 90 degrees), `f(x) = x`
- `304` Double Clayton type IV (survival and rotated 270 degrees), `f(x) = x`
- `401` Double Gumbel type I (standard and rotated 90 degrees), `f(x) = x*(1-1/abs(x))`
- `402` Double Gumbel type II (standard and rotated 270 degrees), `f(x) = x*(1-1/abs(x))`
- `403` Double Gumbel type III (survival and rotated 90 degrees), `f(x) = x*(1-1/abs(x))`
- `404` Double Gumbel type IV (survival and rotated 270 degrees) `f(x) = x*(1-1/abs(x))`.

See Also

`BiCopEta2Par`, `BiCopPar2Tau`, and `BiCopTau2Par` from the **VineCopula** package.

condBiCopSim *Simulation from a Conditional Bivariate Copula*

Description

Simulates from a conditional bivariate copula, where each copula parameter takes a different value, depending on the calibration function and covariates.

Usage

```
condBiCopSim(family, calib.fnc, X, par2 = 0, return.par = TRUE, tau = TRUE)
```

Arguments

family	family A copula family: 1 Gaussian, 2 Student t, 3 Clayton, 4 Gumbel, 5 Frank, 13 Survival Clayton, 14 Survival Gumbel, 23 Rotated (90 degrees) Clayton, 24 Rotated (90 degrees) Gumbel, 33 Rotated (270 degrees) Clayton and 34 Rotated (270 degrees) Gumbel.
calib.fnc	A calibration function.
X	A vector (if calib.fnc takes a single argument) or matrix (if calib.fnc takes multiple arguments) of covariates values.
par2	The second copula parameter (for the Student t), default par2 = 0.
return.par	Should the parameter (and calibration function) be returned as well (default return.par = TRUE)?
tau	Should the calibration function (and the model) be specified for the copula parameter or Kendall's tau (default tau = TRUE)?

Value

If return.par = TRUE, then the function returns a list with:

- data, a matrix with two columns containing the simulated data,
- par, a vector containing the values of the copula parameter,
- and eta, a vector containing the values of the calibration function.

If return.par = FALSE, then the function simply returns data, a matrix with two columns containing the simulated data.

See Also

gamBiCopFit and gamBiCopSimulate.

Examples

```
require(copula)
set.seed(0)

## Simulation parameters (sample size, correlation between covariates,
## Gaussian copula family)
n <- 2e2
rho <- 0.5
fam <- 1


## A calibration surface depending on three variables
eta0 <- 1
calib.surf <- list(
  calib.quad <- function(t, Ti = 0, Tf = 1, b = 8) {
    Tm <- (Tf - Ti) / 2
    a <- -(b / 3) * (Tf^2 - 3 * Tf * Tm + 3 * Tm^2)
    return(a + b * (t - Tm)^2)
  },
  calib.sin <- function(t, Ti = 0, Tf = 1, b = 1, f = 1) {
    a <- b * (1 - 2 * Tf * pi / (f * Tf * pi +
      cos(2 * f * pi * (Tf - Ti))
      - cos(2 * f * pi * Ti)))
    return((a + b) / 2 + (b - a) * sin(2 * f * pi * (t - Ti)) / 2)
  },
  calib.exp <- function(t, Ti = 0, Tf = 1, b = 2, s = Tf / 8) {
    Tm <- (Tf - Ti) / 2
    a <- (b * s * sqrt(2 * pi) / Tf) * (pnorm(0, Tm, s) - pnorm(Tf, Tm, s))
    return(a + b * exp(-(t - Tm)^2 / (2 * s^2)))
  }
)

## Display the calibration surface
par(mfrow = c(1, 3), pty = "s", mar = c(1, 1, 4, 1))
u <- seq(0, 1, length.out = 100)
sel <- matrix(c(1, 1, 2, 2, 3, 3), ncol = 2)
jet.colors <- colorRamp(c(
  "#00007F", "blue", "#007FFF", "cyan", "#7FFF7F",
  "yellow", "#FF7F00", "red", "#7F0000"
))
jet <- function(x) rgb(jet.colors(exp(x / 3) / (1 + exp(x / 3))),
    maxColorValue = 255
  )
for (k in 1:3) {
  tmp <- outer(u, u, function(x, y)
    eta0 + calib.surf[[sel[k, 1]]](x) + calib.surf[[sel[k, 2]]](y))
  persp(u, u, tmp,
    border = NA, theta = 60, phi = 30, zlab = "",
    col = matrix(jet(tmp), nrow = 100),
    xlab = paste("X", sel[k, 1], sep = ""),
    ylab = paste("X", sel[k, 2], sep = ""),
    main = paste("eta0+f", sel[k, 1],
```

```
      "(X", sel[k, 1], ") +f", sel[k, 2],
      "(X", sel[k, 2], ")",
      sep = ""
    )
  )
}

## 3-dimensional matrix X of covariates
covariates.distr <- mvdc(normalCopula(rho, dim = 3),
  c("unif"), list(list(min = 0, max = 1)),
  marginsIdentical = TRUE
)
X <- rMvdc(n, covariates.distr)

## U in [0,1]x[0,1] with copula parameter depending on X
U <- condBiCopSim(fam, function(x1, x2, x3) {
  eta0 + sum(mapply(function(f, x)
    f(x), calib.surf, c(x1, x2, x3)))
}, X[, 1:3], par2 = 6, return.par = TRUE)

## Merge U and X
data <- data.frame(U$data, X)
names(data) <- c(paste("u", 1:2, sep = ""), paste("x", 1:3, sep = ""))

## Display the data
dev.off()
plot(data[, "u1"], data[, "u2"], xlab = "U1", ylab = "U2")
```

`dim.gamVine` *Dimension of an Object of the Class gamVine*

Description

Retrieve the dimension of an object of the class `gamVine`.

Usage

```
## S4 method for signature 'gamVine'
dim(x)
```

Arguments

`x`	An object of the class `gamVine`.

Value

Dimension of the `gamVine` object.

See Also

`gamVine`.

EDF *Equivalent Degrees of Freedom for an Object of the Class gamBiCop*

Description

Function calculating the Equivalent Degrees of Freedom (EDF) for a gamBiCop object. It basically sums the edf of the gamObject for each smooth component.

Usage

```
EDF(object)
```

Arguments

object	An object of the class gamBiCop.

Value

Estimated degrees of freedom for each smooth component.

formula.gamBiCop *Model Formula of the gamBiCop Object*

Description

Extracts the gam formula from an object of the class gamBiCop. This function is a wrapper to formula.gam from the mgcv package.

Usage

```
## S4 method for signature 'gamBiCop'
formula(x, ...)
```

Arguments

x	An object of the class gamBiCop.
...	un-used in this class

See Also

formula.gam function from the mgcv package.

gamBiCop *Construction of a gamBiCop Class Object*

Description

Constructs an object of the class gamBiCop.

Usage

```
gamBiCop(family, model, par2 = 0, tau = TRUE)
```

Arguments

family	A copula family: 1 Gaussian, 2 Student t, 5 Frank, 301 Double Clayton type I (standard and rotated 90 degrees), 302 Double Clayton type II (standard and rotated 270 degrees), 303 Double Clayton type III (survival and rotated 90 degrees), 304 Double Clayton type IV (survival and rotated 270 degrees), 401 Double Gumbel type I (standard and rotated 90 degrees), 402 Double Gumbel type II (standard and rotated 270 degrees), 403 Double Gumbel type III (survival and rotated 90 degrees), 404 Double Gumbel type IV (survival and rotated 270 degrees).
model	A gamObject as return by the gam function from the mgcv package.
par2	Second parameter for the Student t-copula.
tau	FALSE for a calibration function specified for the Copula parameter or TRUE (default) for a calibration function specified for Kendall's tau.

Value

An object of the class gamBiCop.

See Also

gamBiCop, gamBiCopFit, gamBiCopPredict and gamBiCopSimulate.

gamBiCop-class *The gamBiCop Class*

Description

gamBiCop is an S4 class to store a Generalized Additive Model for bivariate copula a parameter or Kendall's tau. Objects can be created by calls of the form new("gamBiCop", ...), or by function gamBiCop.

Slots

`family` A copula family: `1` Gaussian, `2` Student t, `5` Frank, `301` Double Clayton type I (standard and rotated 90 degrees), `302` Double Clayton type II (standard and rotated 270 degrees), `303` Double Clayton type III (survival and rotated 90 degrees), `304` Double Clayton type IV (survival and rotated 270 degrees), `401` Double Gumbel type I (standard and rotated 90 degrees), `402` Double Gumbel type II (standard and rotated 270 degrees), `403` Double Gumbel type III (survival and rotated 90 degrees), `404` Double Gumbel type IV (survival and rotated 270 degrees).

`model` A `gamObject` as return by the `gam` function from the `mgcv` package.

`par2` Second parameter for the Studen t-copula.

`tau` `FALSE` (default) for a calibration fonction specified for the Copula parameter or `TRUE` for a calibration function specified for Kendall's tau.

See Also

`gamBiCopFit`, `gamBiCopPredict` and `gamBiCopSimulate`.

`gamBiCopCDF` *Conditional distribution function of a Generalized Additive model for the copula parameter or Kendall's tau*

Description

This function returns the distribution function of a bivariate conditional copula, where either the copula parameter or the Kendall's tau is modeled as a function of the covariates.

Usage

```
gamBiCopCDF(object, newdata = NULL)
```

Arguments

`object`	`gamBiCop-class` object.
`newdata`	(Same as in `predict.gam` from the `mgcv` package) A matrix or data frame containing the values of the model covariates at which predictions are required. If this is not provided then the distribution corresponding to the original data are returned. If `newdata` is provided then it should contain all the variables needed for prediction: a warning is generated if not.

Value

The conditional density.

See Also

`gamBiCop` and `gamBiCopPredict`.

Examples

```
require(copula)
set.seed(0)

## Simulation parameters (sample size, correlation between covariates,
## Gaussian copula family)
n <- 2e2
rho <- 0.5
fam <- 1

## A calibration surface depending on three variables
eta0 <- 1
calib.surf <- list(
  calib.quad <- function(t, Ti = 0, Tf = 1, b = 8) {
    Tm <- (Tf - Ti) / 2
    a <- -(b / 3) * (Tf^2 - 3 * Tf * Tm + 3 * Tm^2)
    return(a + b * (t - Tm)^2)
  },
  calib.sin <- function(t, Ti = 0, Tf = 1, b = 1, f = 1) {
    a <- b * (1 - 2 * Tf * pi / (f * Tf * pi +
      cos(2 * f * pi * (Tf - Ti))
      - cos(2 * f * pi * Ti)))
    return((a + b) / 2 + (b - a) * sin(2 * f * pi * (t - Ti)) / 2)
  },
  calib.exp <- function(t, Ti = 0, Tf = 1, b = 2, s = Tf / 8) {
    Tm <- (Tf - Ti) / 2
    a <- (b * s * sqrt(2 * pi) / Tf) * (pnorm(0, Tm, s) - pnorm(Tf, Tm, s))
    return(a + b * exp(-(t - Tm)^2 / (2 * s^2)))
  }
)

## 3-dimensional matrix X of covariates
covariates.distr <- mvdc(normalCopula(rho, dim = 3),
  c("unif"), list(list(min = 0, max = 1)),
  marginsIdentical = TRUE
)
X <- rMvdc(n, covariates.distr)
colnames(X) <- paste("x", 1:3, sep = "")

## U in [0,1]x[0,1] with copula parameter depending on X
U <- condBiCopSim(fam, function(x1, x2, x3) {
  eta0 + sum(mapply(function(f, x)
    f(x), calib.surf, c(x1, x2, x3)))
}, X[, 1:3], par2 = 6, return.par = TRUE)

## Merge U and X
data <- data.frame(U$data, X)
names(data) <- c(paste("u", 1:2, sep = ""), paste("x", 1:3, sep = ""))

## Model fit with penalized cubic splines (via min GCV)
basis <- c(3, 10, 10)
formula <- ~ s(x1, k = basis[1], bs = "cr") +
```

```
  s(x2, k = basis[2], bs = "cr") +
  s(x3, k = basis[3], bs = "cr")
system.time(fit <- gamBiCopFit(data, formula, fam))

## Evaluate the conditional density
gamBiCopCDF(fit$res)
```

`gamBiCopFit` *Maximum penalized likelihood estimation of a Generalized Additive model for the copula parameter or Kendall's tau.*

Description

This function estimates the parameter(s) of a Generalized Additive model (gam) for the copula parameter or Kendall's tau. It solves the maximum penalized likelihood estimation for the copula families supported in this package by reformulating each Newton-Raphson iteration as a generalized ridge regression, which is solved using the `mgcv` package.

Usage

```
gamBiCopFit(
  data,
  formula = ~1,
  family = 1,
  tau = TRUE,
  method = "FS",
  tol.rel = 0.001,
  n.iters = 10,
  verbose = FALSE,
  ...
)
```

Arguments

`data`	A list, data frame or matrix containing the model responses, (u1,u2) in [0,1]x[0,1], and covariates required by the formula.
`formula`	A gam formula (see `gam`, `formula.gam` and `gam.models` from `mgcv`).
`family`	A copula family: `1` Gaussian, `2` Student t, `5` Frank, `301` Double Clayton type I (standard and rotated 90 degrees), `302` Double Clayton type II (standard and rotated 270 degrees), `303` Double Clayton type III (survival and rotated 90 degrees), `304` Double Clayton type IV (survival and rotated 270 degrees), `401` Double Gumbel type I (standard and rotated 90 degrees), `402` Double Gumbel type II (standard and rotated 270 degrees), `403` Double Gumbel type III (survival and rotated 90 degrees), `404` Double Gumbel type IV (survival and rotated 270 degrees).
`tau`	`FALSE` (default) for a calibration function specified for the Copula parameter or `TRUE` for a calibration function specified for Kendall's tau.

method	'NR' for Newton-Raphson and 'FS' for Fisher-scoring (default).
tol.rel	Relative tolerance for 'FS'/'NR' algorithm.
n.iters	Maximal number of iterations for 'FS'/'NR' algorithm.
verbose	TRUE if informations should be printed during the estimation and FALSE (default) for a silent version.
...	Additional parameters to be passed to gam from mgcv.

Value

gamBiCopFit returns a list consisting of

res	S4 gamBiCop-class object.
method	'FS' for Fisher-scoring (default) and 'NR' for Newton-Raphson.
tol.rel	relative tolerance for 'FS'/'NR' algorithm.
n.iters	maximal number of iterations for 'FS'/'NR' algorithm.
trace	the estimation procedure's trace.
conv	0 if the algorithm converged and 1 otherwise.

See Also

gamBiCop and gamBiCopSimulate.

Examples

```
require(copula)
require(mgcv)
set.seed(0)

## Simulation parameters (sample size, correlation between covariates,
## Gaussian copula family)
n <- 5e2
rho <- 0.5
fam <- 1


## A calibration surface depending on three variables
eta0 <- 1
calib.surf <- list(
  calib.quad <- function(t, Ti = 0, Tf = 1, b = 8) {
    Tm <- (Tf - Ti) / 2
    a <- -(b / 3) * (Tf^2 - 3 * Tf * Tm + 3 * Tm^2)
    return(a + b * (t - Tm)^2)
  },
  calib.sin <- function(t, Ti = 0, Tf = 1, b = 1, f = 1) {
    a <- b * (1 - 2 * Tf * pi / (f * Tf * pi +
      cos(2 * f * pi * (Tf - Ti))
      - cos(2 * f * pi * Ti)))
    return((a + b) / 2 + (b - a) * sin(2 * f * pi * (t - Ti)) / 2)
  },
```

```
  calib.exp <- function(t, Ti = 0, Tf = 1, b = 2, s = Tf / 8) {
    Tm <- (Tf - Ti) / 2
    a <- (b * s * sqrt(2 * pi) / Tf) * (pnorm(0, Tm, s) - pnorm(Tf, Tm, s))
    return(a + b * exp(-(t - Tm)^2 / (2 * s^2)))
  }
)

## Display the calibration surface
par(mfrow = c(1, 3), pty = "s", mar = c(1, 1, 4, 1))
u <- seq(0, 1, length.out = 100)
sel <- matrix(c(1, 1, 2, 2, 3, 3), ncol = 2)
jet.colors <- colorRamp(c(
  "#00007F", "blue", "#007FFF", "cyan", "#7FFF7F",
  "yellow", "#FF7F00", "red", "#7F0000"
))
jet <- function(x) rgb(jet.colors(exp(x / 3) / (1 + exp(x / 3))),
    maxColorValue = 255
  )
for (k in 1:3) {
  tmp <- outer(u, u, function(x, y)
    eta0 + calib.surf[[sel[k, 1]]](x) + calib.surf[[sel[k, 2]]](y))
  persp(u, u, tmp,
    border = NA, theta = 60, phi = 30, zlab = "",
    col = matrix(jet(tmp), nrow = 100),
    xlab = paste("X", sel[k, 1], sep = ""),
    ylab = paste("X", sel[k, 2], sep = ""),
    main = paste("eta0+f", sel[k, 1],
      "(X", sel[k, 1], ") +f", sel[k, 2],
      "(X", sel[k, 2], ")",
      sep = ""
    )
  )
}

## 3-dimensional matrix X of covariates
covariates.distr <- mvdc(normalCopula(rho, dim = 3),
  c("unif"), list(list(min = 0, max = 1)),
  marginsIdentical = TRUE
)
X <- rMvdc(n, covariates.distr)

## U in [0,1]x[0,1] with copula parameter depending on X
U <- condBiCopSim(fam, function(x1, x2, x3) {
  eta0 + sum(mapply(function(f, x)
    f(x), calib.surf, c(x1, x2, x3)))
}, X[, 1:3], par2 = 6, return.par = TRUE)

## Merge U and X
data <- data.frame(U$data, X)
names(data) <- c(paste("u", 1:2, sep = ""), paste("x", 1:3, sep = ""))

## Display the data
dev.off()
```

```
plot(data[, "u1"], data[, "u2"], xlab = "U1", ylab = "U2")

## Model fit with a basis size (arguably) too small
## and unpenalized cubic spines
pen <- FALSE
basis0 <- c(3, 4, 4)
formula <- ~ s(x1, k = basis0[1], bs = "cr", fx = !pen) +
  s(x2, k = basis0[2], bs = "cr", fx = !pen) +
  s(x3, k = basis0[3], bs = "cr", fx = !pen)
system.time(fit0 <- gamBiCopFit(data, formula, fam))

## Model fit with a better basis size and penalized cubic splines (via min GCV)
pen <- TRUE
basis1 <- c(3, 10, 10)
formula <- ~ s(x1, k = basis1[1], bs = "cr", fx = !pen) +
  s(x2, k = basis1[2], bs = "cr", fx = !pen) +
  s(x3, k = basis1[3], bs = "cr", fx = !pen)
system.time(fit1 <- gamBiCopFit(data, formula, fam))

## Extract the gamBiCop objects and show various methods
(res <- sapply(list(fit0, fit1), function(fit) {
  fit$res
}))
metds <- list("logLik" = logLik, "AIC" = AIC, "BIC" = BIC, "EDF" = EDF)
lapply(res, function(x) sapply(metds, function(f) f(x)))


## Comparison between fitted, true smooth and spline approximation for each
## true smooth function for the two basis sizes
fitted <- lapply(res, function(x) gamBiCopPredict(x, data.frame(x1 = u, x2 = u, x3 = u),
    type = "terms"
  )$calib)
true <- vector("list", 3)
for (i in 1:3) {
  y <- eta0 + calib.surf[[i]](u)
  true[[i]]$true <- y - eta0
  temp <- gam(y ~ s(u, k = basis0[i], bs = "cr", fx = TRUE))
  true[[i]]$approx <- predict.gam(temp, type = "terms")
  temp <- gam(y ~ s(u, k = basis1[i], bs = "cr", fx = FALSE))
  true[[i]]$approx2 <- predict.gam(temp, type = "terms")
}

## Display results
par(mfrow = c(1, 3), pty = "s")
yy <- range(true, fitted)
yy[1] <- yy[1] * 1.5
for (k in 1:3) {
  plot(u, true[[k]]$true,
    type = "l", ylim = yy,
    xlab = paste("Covariate", k), ylab = paste("Smooth", k)
  )
  lines(u, true[[k]]$approx, col = "red", lty = 2)
  lines(u, fitted[[1]][, k], col = "red")
```

```
  lines(u, fitted[[2]][, k], col = "green")
  lines(u, true[[k]]$approx2, col = "green", lty = 2)
  legend("bottomleft",
    cex = 0.6, lty = c(1, 1, 2, 1, 2),
    c("True", "Fitted", "Appox 1", "Fitted 2", "Approx 2"),
    col = c("black", "red", "red", "green", "green")
  )
}
```

---

gamBiCopPDF *Conditional density function of a Generalized Additive model for the copula parameter or Kendall's tau*

Description

This function returns the density of a bivariate conditional copula, where either the copula parameter or the Kendall's tau is modeled as a function of the covariates.

Usage

```
gamBiCopPDF(object, newdata = NULL)
```

Arguments

object	gamBiCop-class object.
newdata	(Same as in predict.gam from the mgcv package) A matrix or data frame containing the values of the model covariates at which predictions are required, along with two columns named ‘"u1"‘ and ‘"u2"‘. If this is not provided then the density corresponding to the original data are returned. If newdata is provided then it should contain all the variables needed for prediction: a warning is generated if not.

Value

The conditional density.

See Also

gamBiCop and gamBiCopPredict.

Examples

```
require(copula)
set.seed(0)

## Simulation parameters (sample size, correlation between covariates,
## Gaussian copula family)
n <- 2e2
rho <- 0.5
```

```
fam <- 1

## A calibration surface depending on three variables
eta0 <- 1
calib.surf <- list(
  calib.quad <- function(t, Ti = 0, Tf = 1, b = 8) {
    Tm <- (Tf - Ti) / 2
    a <- -(b / 3) * (Tf^2 - 3 * Tf * Tm + 3 * Tm^2)
    return(a + b * (t - Tm)^2)
  },
  calib.sin <- function(t, Ti = 0, Tf = 1, b = 1, f = 1) {
    a <- b * (1 - 2 * Tf * pi / (f * Tf * pi +
      cos(2 * f * pi * (Tf - Ti))
      - cos(2 * f * pi * Ti)))
    return((a + b) / 2 + (b - a) * sin(2 * f * pi * (t - Ti)) / 2)
  },
  calib.exp <- function(t, Ti = 0, Tf = 1, b = 2, s = Tf / 8) {
    Tm <- (Tf - Ti) / 2
    a <- (b * s * sqrt(2 * pi) / Tf) * (pnorm(0, Tm, s) - pnorm(Tf, Tm, s))
    return(a + b * exp(-(t - Tm)^2 / (2 * s^2)))
  }
)

## 3-dimensional matrix X of covariates
covariates.distr <- mvdc(normalCopula(rho, dim = 3),
  c("unif"), list(list(min = 0, max = 1)),
  marginsIdentical = TRUE
)
X <- rMvdc(n, covariates.distr)
colnames(X) <- paste("x", 1:3, sep = "")

## U in [0,1]x[0,1] with copula parameter depending on X
U <- condBiCopSim(fam, function(x1, x2, x3) {
  eta0 + sum(mapply(function(f, x)
    f(x), calib.surf, c(x1, x2, x3)))
}, X[, 1:3], par2 = 6, return.par = TRUE)

## Merge U and X
data <- data.frame(U$data, X)
names(data) <- c(paste("u", 1:2, sep = ""), paste("x", 1:3, sep = ""))

## Model fit with penalized cubic splines (via min GCV)
basis <- c(3, 10, 10)
formula <- ~ s(x1, k = basis[1], bs = "cr") +
  s(x2, k = basis[2], bs = "cr") +
  s(x3, k = basis[3], bs = "cr")
system.time(fit <- gamBiCopFit(data, formula, fam))

## Evaluate the conditional density
gamBiCopPDF(fit$res)
```

`gamBiCopPredict` *Predict method of a Generalized Additive model for the copula parameter or Kendall's tau*

Description

Predict method of a Generalized Additive model for the copula parameter or Kendall's tau

Usage

```
gamBiCopPredict(
  object,
  newdata = NULL,
  target = "calib",
  alpha = 0,
  type = "link"
)
```

Arguments

`object`	`gamBiCop-class` object.
`newdata`	(Same as in `predict.gam` from the `mgcv` package) A matrix or data frame containing the values of the model covariates at which predictions are required. If this is not provided then predictions corresponding to the original data are returned. If `newdata` is provided then it should contain all the variables needed for prediction: a warning is generated if not.
`target`	Either `'calib'`, `'par'` or `'tau'` or a combination of those. `'calib'` (default) corresponds to the calibration function, `'par'` to the copula parameter and `'tau'` to Kendall's tau.
`alpha`	In (0,1) to return the corresponding confidence interval.
`type`	(Similar as in `predict.gam` from the `mgcv` package, only active for `type = 'calib'`). When this has the value `'link'` (default), the calibration function is returned. When `type = 'terms'` each component of the linear predictor is returned separately (possibly with standard errors): this includes parametric model components, followed by each smooth component, but excludes any offset and any intercept. When `type = 'lpmatrix'` then a matrix is returned which yields the values of the linear predictor (minus any offset) when post-multiplied by the parameter vector (in this case alpha is ignored).

Value

If `target = 'calib'`, then a list with 1 item `calib`. If `target = 'par'`, `target = 'tau'` or `target = c('par', 'tau')`, then a list with 2, 2 or 3 items, namely `calib` and `par`, `tau` and `par`, or `calib`, `tau` and `par`.

If `alpha` is in (0,1), then a additional items of the list are `calib.CI` as well as e.g. `par.CI` and/or `tau.CI` depending on the value of `target`.

Otherwise, if type = 'lpmatrix' (only active for type = 'calib'), then a matrix is returned which will give a vector of linear predictor values (minus any offset) at the supplied covariate values, when applied to the model coefficient vector (similar as predict.gam from the mgcv).

See Also

gamBiCop and gamBiCopFit.

Examples

```
require(copula)
set.seed(0)

## Simulation parameters (sample size, correlation between covariates,
## Clayton copula family)
n <- 5e2
rho <- 0.5
fam <- 1

## A calibration surface depending on three variables
eta0 <- 1
calib.surf <- list(
  calib.quad <- function(t, Ti = 0, Tf = 1, b = 8) {
    Tm <- (Tf - Ti) / 2
    a <- -(b / 3) * (Tf^2 - 3 * Tf * Tm + 3 * Tm^2)
    return(a + b * (t - Tm)^2)
  },
  calib.sin <- function(t, Ti = 0, Tf = 1, b = 1, f = 1) {
    a <- b * (1 - 2 * Tf * pi / (f * Tf * pi +
      cos(2 * f * pi * (Tf - Ti))
      - cos(2 * f * pi * Ti)))
    return((a + b) / 2 + (b - a) * sin(2 * f * pi * (t - Ti)) / 2)
  },
  calib.exp <- function(t, Ti = 0, Tf = 1, b = 2, s = Tf / 8) {
    Tm <- (Tf - Ti) / 2
    a <- (b * s * sqrt(2 * pi) / Tf) * (pnorm(0, Tm, s) - pnorm(Tf, Tm, s))
    return(a + b * exp(-(t - Tm)^2 / (2 * s^2)))
  }
)

## 3-dimensional matrix X of covariates
covariates.distr <- mvdc(normalCopula(rho, dim = 3),
  c("unif"), list(list(min = 0, max = 1)),
  marginsIdentical = TRUE
)
X <- rMvdc(n, covariates.distr)
colnames(X) <- paste("x", 1:3, sep = "")

## U in [0,1]x[0,1] with copula parameter depending on X
U <- condBiCopSim(fam, function(x1, x2, x3) {
  eta0 + sum(mapply(function(f, x)
    f(x), calib.surf, c(x1, x2, x3)))
```

```
}, X[, 1:3], par2 = 6, return.par = TRUE)

## Merge U and X
data <- data.frame(U$data, X)
names(data) <- c(paste("u", 1:2, sep = ""), paste("x", 1:3, sep = ""))

## Model fit with penalized cubic splines (via min GCV)
basis <- c(3, 10, 10)
formula <- ~ s(x1, k = basis[1], bs = "cr") +
  s(x2, k = basis[2], bs = "cr") +
  s(x3, k = basis[3], bs = "cr")
system.time(fit <- gamBiCopFit(data, formula, fam))

## Extract the gamBiCop objects and show various methods
(res <- fit$res)
EDF(res)
pred <- gamBiCopPredict(fit$res, X, target = c("calib", "par", "tau"))
```

---

gamBiCopSelect *Selection and Maximum penalized likelihood estimation of a Generalized Additive model (gam) for the copula parameter or Kendall's tau.*

Description

This function selects an appropriate bivariate copula family for given bivariate copula data using one of a range of methods. The corresponding parameter estimates are obtained by maximum penalized likelihood estimation, where each Newton-Raphson iteration is reformulated as a generalized ridge regression solved using the mgcv package.

Usage

```
gamBiCopSelect(
  udata,
  lin.covs = NULL,
  smooth.covs = NULL,
  familyset = NA,
  rotations = TRUE,
  familycrit = "AIC",
  level = 0.05,
  edf = 1.5,
  tau = TRUE,
  method = "FS",
  tol.rel = 0.001,
  n.iters = 10,
  parallel = FALSE,
  verbose = FALSE,
  select.once = TRUE,
  ...
)
```

Arguments

udata	A matrix or data frame containing the model responses, (u1,u2) in [0,1]x[0,1]
lin.covs	A matrix or data frame containing the parametric (i.e., linear) covariates.
smooth.covs	A matrix or data frame containing the non-parametric (i.e., smooth) covariates.
familyset	(Similar to BiCopSelect from the VineCopula package) Vector of bivariate copula families to select from. If familyset = NA (default), selection among all possible families is performed. Coding of bivariate copula families: 1 Gaussian, 2 Student t, 5 Frank, 301 Double Clayton type I (standard and rotated 90 degrees), 302 Double Clayton type II (standard and rotated 270 degrees), 303 Double Clayton type III (survival and rotated 90 degrees), 304 Double Clayton type IV (survival and rotated 270 degrees), 401 Double Gumbel type I (standard and rotated 90 degrees), 402 Double Gumbel type II (standard and rotated 270 degrees), 403 Double Gumbel type III (survival and rotated 90 degrees), 404 Double Gumbel type IV (survival and rotated 270 degrees).
rotations	If TRUE, all rotations of the families in familyset are included.
familycrit	Character indicating the criterion for bivariate copula selection. Possible choices: familycrit = 'AIC' (default) or 'BIC', as in BiCopSelect from the VineCopula package.
level	Numerical; significance level of the test for removing individual predictors (default: level = 0.05).
edf	Numerical; if the estimated EDF for individual predictors is smaller than edf but the predictor is still significant, then it is set as linear (default: edf = 1.5).
tau	FALSE for a calibration function specified for the Copula parameter or TRUE (default) for a calibration function specified for Kendall's tau.
method	'FS' for Fisher-scoring (default) and 'NR' for Newton-Raphson.
tol.rel	Relative tolerance for 'FS'/'NR' algorithm.
n.iters	Maximal number of iterations for 'FS'/'NR' algorithm.
parallel	TRUE for a parallel estimation across copula families.
verbose	TRUE prints informations during the estimation.
select.once	if TRUE the GAM structure is only selected once, for the family that appears first in familyset.
...	Additional parameters to be passed to gam

Value

gamBiCopFit returns a list consisting of

res	S4 gamBiCop-class object.
method	'FS' for Fisher-scoring and 'NR' for Newton-Raphson.
tol.rel	relative tolerance for 'FS'/'NR' algorithm.
n.iters	maximal number of iterations for 'FS'/'NR' algorithm.
trace	the estimation procedure's trace.
conv	0 if the algorithm converged and 1 otherwise.

See Also

gamBiCop and gamBiCopFit.

Examples

```
require(copula)
set.seed(0)

## Simulation parameters (sample size, correlation between covariates,
## Student copula with 4 degrees of freedom)
n <- 5e2
rho <- 0.9
fam <- 2
par2 <- 4

## A calibration surface depending on four variables
eta0 <- 1
calib.surf <- list(
  calib.lin <- function(t, Ti = 0, Tf = 1, b = 2) {
    return(-2 + 4 * t)
  },
  calib.quad <- function(t, Ti = 0, Tf = 1, b = 8) {
    Tm <- (Tf - Ti) / 2
    a <- -(b / 3) * (Tf^2 - 3 * Tf * Tm + 3 * Tm^2)
    return(a + b * (t - Tm)^2)
  },
  calib.sin <- function(t, Ti = 0, Tf = 1, b = 1, f = 1) {
    a <- b * (1 - 2 * Tf * pi / (f * Tf * pi +
      cos(2 * f * pi * (Tf - Ti))
      - cos(2 * f * pi * Ti)))
    return((a + b) / 2 + (b - a) * sin(2 * f * pi * (t - Ti)) / 2)
  },
  calib.exp <- function(t, Ti = 0, Tf = 1, b = 2, s = Tf / 8) {
    Tm <- (Tf - Ti) / 2
    a <- (b * s * sqrt(2 * pi) / Tf) * (pnorm(0, Tm, s) - pnorm(Tf, Tm, s))
    return(a + b * exp(-(t - Tm)^2 / (2 * s^2)))
  }
)

## 6-dimensional matrix X of covariates
covariates.distr <- mvdc(normalCopula(rho, dim = 6),
  c("unif"), list(list(min = 0, max = 1)),
  marginsIdentical = TRUE
)
X <- rMvdc(n, covariates.distr)
colnames(X) <- paste("x", 1:6, sep = "")

## U in [0,1]x[0,1] depending on the four first columns of X
U <- condBiCopSim(fam, function(x1, x2, x3, x4) {
  eta0 + sum(mapply(function(f, x)
    f(x), calib.surf, c(x1, x2, x3, x4)))
}, X[, 1:4], par2 = 4, return.par = TRUE)
```

```
## Not run:
## Selection using AIC (about 30sec on single core)
## Use parallel = TRUE to speed-up....
system.time(best <- gamBiCopSelect(U$data, smooth.covs = X))
print(best$res)
EDF(best$res) ## The first function is linear
## Plot only the smooth component
par(mfrow = c(2, 2))
plot(best$res)

## End(Not run)
```

---

gamBiCopSimulate *Simulate from* gamBiCop-class *object*

Description

Simulate from gamBiCop-class object

Usage

```
gamBiCopSimulate(
  object,
  newdata = NULL,
  N = NULL,
  return.calib = FALSE,
  return.par = FALSE,
  return.tau = FALSE
)
```

Arguments

object	gamBiCop-class object.
newdata	(same as in predict.gam from the mgcv package) A matrix or data frame containing the values of the model covariates at which simulations are required. If this is not provided then simulations corresponding to the original data are returned.
N	sample size.
return.calib	should the calibration function (TRUE) be returned or not (FALSE)?
return.par	should the copula parameter (TRUE) be returned or not (FALSE)?
return.tau	should the Kendall's tau (TRUE) be returned or not (FALSE)?

Value

A list with 1 item `data`. When `N` is smaller or larger than the `newdata`'s number of rows (or the number of rows in the original data if `newdata` is not provided), then `N` observations are sampled uniformly (with replacement) among the row of `newdata` (or the rows of the original data if `newdata` is not provided).

If `return.calib` = `TRUE`, `return.par` = `TRUE` and/or `return.tau` = `TRUE`, then the list also contains respectively items `calib`, `par` and/or `tau`.

Examples

```
require(copula)
set.seed(1)

## Simulation parameters (sample size, correlation between covariates,
## Gaussian copula family)
n <- 5e2
rho <- 0.5
fam <- 1

## A calibration surface depending on three variables
eta0 <- 1
calib.surf <- list(
  calib.quad <- function(t, Ti = 0, Tf = 1, b = 8) {
    Tm <- (Tf - Ti) / 2
    a <- -(b / 3) * (Tf^2 - 3 * Tf * Tm + 3 * Tm^2)
    return(a + b * (t - Tm)^2)
  },
  calib.sin <- function(t, Ti = 0, Tf = 1, b = 1, f = 1) {
    a <- b * (1 - 2 * Tf * pi / (f * Tf * pi +
      cos(2 * f * pi * (Tf - Ti))
      - cos(2 * f * pi * Ti)))
    return((a + b) / 2 + (b - a) * sin(2 * f * pi * (t - Ti)) / 2)
  },
  calib.exp <- function(t, Ti = 0, Tf = 1, b = 2, s = Tf / 8) {
    Tm <- (Tf - Ti) / 2
    a <- (b * s * sqrt(2 * pi) / Tf) * (pnorm(0, Tm, s) - pnorm(Tf, Tm, s))
    return(a + b * exp(-(t - Tm)^2 / (2 * s^2)))
  }
)

## 3-dimensional matrix X of covariates
covariates.distr <- mvdc(normalCopula(rho, dim = 3),
  c("unif"), list(list(min = 0, max = 1)),
  marginsIdentical = TRUE
)
X <- rMvdc(n, covariates.distr)
colnames(X) <- paste("x", 1:3, sep = "")

## U in [0,1]x[0,1] with copula parameter depending on X
U <- condBiCopSim(fam, function(x1, x2, x3) {
  eta0 + sum(mapply(function(f, x)
```

```
    f(x), calib.surf, c(x1, x2, x3)))
}, X[, 1:3], par2 = 6, return.par = TRUE)

## Merge U and X
data <- data.frame(U$data, X)
names(data) <- c(paste("u", 1:2, sep = ""), paste("x", 1:3, sep = ""))

## Model fit with penalized cubic splines (via min GCV)
basis <- c(3, 10, 10)
formula <- ~ s(x1, k = basis[1], bs = "cr") +
  s(x2, k = basis[2], bs = "cr") +
  s(x3, k = basis[3], bs = "cr")
system.time(fit <- gamBiCopFit(data, formula, fam))

## Extract the gamBiCop objects and show various methods
(res <- fit$res)
EDF(res)
sim <- gamBiCopSimulate(fit$res, X)
```

---

gamVine *Construction of a gamVine Class Object*

Description

Constructs an object of the class gamVine.

Usage

```
gamVine(Matrix, model, names = NA, covariates = NA)
```

Arguments

Matrix	lower triangular d x d matrix that defines the tree structure.
model	list containing d x (d-1)/2 lists with three numeric items (family, par and par2) and/or objects of the class gamBiCop.
names	vector of d names.
covariates	vector of names for the covariates.

Value

An object of the class gamVine.

See Also

gamVine, RVineMatrix, gamBiCop gamVineSeqFit, gamVineCopSelect, gamVineStructureSelect and gamVineSimulate.

gamVine-class *The gamVine Class*

Description

gamVine is an S4 class to store a conditional and potentially non-simplified pair-copula construction. Objects can be created by calls of the form `new("gamVine", ...)`, or by function `gamVine`.

Slots

`Matrix` Lower triangular d x d matrix that defines the tree structure.

`model` list containing d x (d-1)/2 lists with three numeric items (family, par and par2) and/or `gamBiCop` objects.

`names` vector of d names.

`covariates` vector of names for the exogenous covariates.

See Also

`gamVine`, `RVineMatrix`, `gamBiCop` `gamVineSeqFit`, `gamVineCopSelect`, `gamVineStructureSelect` and `gamVineSimulate`.

gamVineCopSelect *Sequential pair-copula selection and maximum penalized likelihood estimation of a GAM-Vine model.*

Description

This function select the copula family and estimates the parameter(s) of a Generalized Additive model (GAM) Vine model, where GAMs for individual edges are specified either for the copula parameter or Kendall's tau. It solves the maximum penalized likelihood estimation for the copula families supported in this package by reformulating each Newton-Raphson iteration as a generalized ridge regression, which is solved using the `mgcv` package.

Usage

```
gamVineCopSelect(
  data,
  Matrix,
  lin.covs = NULL,
  smooth.covs = NULL,
  simplified = FALSE,
  familyset = NA,
  rotations = TRUE,
  familycrit = "AIC",
```

```
  level = 0.05,
  trunclevel = NA,
  tau = TRUE,
  method = "FS",
  tol.rel = 0.001,
  n.iters = 10,
  parallel = FALSE,
  verbose = FALSE,
  select.once = TRUE,
  ...
)
```

Arguments

data	A matrix or data frame containing the data in [0,1]^d.
Matrix	Lower triangular d x d matrix that defines the R-vine tree structure.
lin.covs	A matrix or data frame containing the parametric (i.e., linear) covariates (default: lin.covs = NULL).
smooth.covs	A matrix or data frame containing the non-parametric (i.e., smooth) covariates (default: smooth.covs = NULL).
simplified	If TRUE, then a simplified vine is fitted (which is possible only if there are exogenous covariates). If FALSE (default), then a non-simplified vine is fitted.
familyset	An integer vector of pair-copula families to select from (the independence copula MUST NOT be specified in this vector unless one wants to fit an independence vine!). The vector has to include at least one pair-copula family that allows for positive and one that allows for negative dependence. Not listed copula families might be included to better handle limit cases. If familyset = NA (default), selection among all possible families is performed. Coding of pair-copula families: 1 Gaussian, 2 Student t, 3 Clayton, 4 Gumbel, 13 Survival Clayton, 14 Survival Gumbel, 23 Rotated (90 degrees) Clayton, 24 Rotated (90 degrees) Gumbel, 33 Rotated (270 degrees) Clayton and 34 Rotated (270 degrees) Gumbel.
rotations	If TRUE, all rotations of the families in familyset are included.
familycrit	Character indicating the criterion for bivariate copula selection. Possible choices: familycrit = 'AIC' (default) or 'BIC', as in BiCopSelect from the VineCopula package.
level	Numerical; Passed to gamBiCopSelect, it is the significance level of the test for removing individual predictors (default: level = 0.05) for each conditional pair-copula.
trunclevel	Integer; level of truncation.
tau	TRUE (default) for a calibration function specified for Kendall's tau or FALSE for a calibration function specified for the Copula parameter.
method	'NR' for Newton-Raphson and 'FS' for Fisher-scoring (default).
tol.rel	Relative tolerance for 'FS'/'NR' algorithm.
n.iters	Maximal number of iterations for 'FS'/'NR' algorithm.

parallel	TRUE (default) for parallel selection of copula family at each edge or FALSE for the sequential version. for the Copula parameter.
verbose	TRUE if informations should be printed during the estimation and FALSE (default) for a silent version. from mgcv.
select.once	if TRUE the GAM structure is only selected once, for the family that appears first in familyset.
...	Additional parameters to be passed to gam from mgcv.

Value

gamVineCopSelect returns a gamVine-class object.

See Also

gamVineSeqFit,gamVineStructureSelect, gamVine-class, gamVineSimulate and gamBiCopFit.

Examples

```
require(mgcv)
set.seed(0)

##  Simulation parameters
# Sample size
n <- 1e3
# Copula families
familyset <- c(1:2, 301:304, 401:404)
# Define a 4-dimensional R-vine tree structure matrix
d <- 4
Matrix <- c(2, 3, 4, 1, 0, 3, 4, 1, 0, 0, 4, 1, 0, 0, 0, 1)
Matrix <- matrix(Matrix, d, d)
nnames <- paste("X", 1:d, sep = "")

## A function factory
eta0 <- 1
calib.surf <- list(
  calib.quad <- function(t, Ti = 0, Tf = 1, b = 8) {
    Tm <- (Tf - Ti) / 2
    a <- -(b / 3) * (Tf^2 - 3 * Tf * Tm + 3 * Tm^2)
    return(a + b * (t - Tm)^2)
  },
  calib.sin <- function(t, Ti = 0, Tf = 1, b = 1, f = 1) {
    a <- b * (1 - 2 * Tf * pi / (f * Tf * pi +
      cos(2 * f * pi * (Tf - Ti))
      - cos(2 * f * pi * Ti)))
    return((a + b) / 2 + (b - a) * sin(2 * f * pi * (t - Ti)) / 2)
  },
  calib.exp <- function(t, Ti = 0, Tf = 1, b = 2, s = Tf / 8) {
    Tm <- (Tf - Ti) / 2
    a <- (b * s * sqrt(2 * pi) / Tf) * (pnorm(0, Tm, s) - pnorm(Tf, Tm, s))
    return(a + b * exp(-(t - Tm)^2 / (2 * s^2)))
  }
```

```
)

##  Create the model
# Define gam-vine model list
count <- 1
model <- vector(mode = "list", length = d * (d - 1) / 2)
sel <- seq(d, d^2 - d, by = d)

# First tree
for (i in 1:(d - 1)) {
  # Select a copula family
  family <- sample(familyset, 1)
  model[[count]]$family <- family

  # Use the canonical link and a randomly generated parameter
  if (is.element(family, c(1, 2))) {
    model[[count]]$par <- tanh(rnorm(1) / 2)
    if (family == 2) {
      model[[count]]$par2 <- 2 + exp(rnorm(1))
    }
  } else {
    if (is.element(family, c(401:404))) {
      rr <- rnorm(1)
      model[[count]]$par <- sign(rr) * (1 + abs(rr))
    } else {
      model[[count]]$par <- rnorm(1)
    }
    model[[count]]$par2 <- 0
  }
  count <- count + 1
}

# A dummy dataset
data <- data.frame(u1 = runif(1e2), u2 = runif(1e2), matrix(runif(1e2 * d), 1e2, d))

# Trees 2 to (d-1)
for (j in 2:(d - 1)) {
  for (i in 1:(d - j)) {
    # Select a copula family
    family <- sample(familyset, 1)

    # Select the conditiong set and create a model formula
    cond <- nnames[sort(Matrix[(d - j + 2):d, i])]
    tmpform <- paste("~", paste(paste("s(", cond, ", k=10, bs='cr')",
      sep = ""
    ), collapse = " + "))
    l <- length(cond)
    temp <- sample(3, l, replace = TRUE)

    # Spline approximation of the true function
    m <- 1e2
    x <- matrix(seq(0, 1, length.out = m), nrow = m, ncol = 1)
    if (l != 1) {
```

```
      tmp.fct <- paste("function(x){eta0+",
        paste(sapply(1:l, function(x)
          paste("calib.surf[[", temp[x], "]](x[", x, "])",
            sep = ""
          )), collapse = "+"), "}",
        sep = ""
      )
      tmp.fct <- eval(parse(text = tmp.fct))
      x <- eval(parse(text = paste0("expand.grid(",
        paste0(rep("x", l), collapse = ","), ")",
        collapse = ""
      )))
      y <- apply(x, 1, tmp.fct)
    } else {
      tmp.fct <- function(x) eta0 + calib.surf[[temp]](x)
      colnames(x) <- cond
      y <- tmp.fct(x)
    }

    # Estimate the gam model
    form <- as.formula(paste0("y", tmpform))
    dd <- data.frame(y, x)
    names(dd) <- c("y", cond)
    b <- gam(form, data = dd)
    # plot(x[,1],(y-fitted(b))/y)

    # Create a dummy gamBiCop object
    tmp <- gamBiCopFit(data = data, formula = form, family = 1, n.iters = 1)$res

    # Update the copula family and the model coefficients
    attr(tmp, "model")$coefficients <- coefficients(b)
    attr(tmp, "model")$smooth <- b$smooth
    attr(tmp, "family") <- family
    if (family == 2) {
      attr(tmp, "par2") <- 2 + exp(rnorm(1))
    }
    model[[count]] <- tmp
    count <- count + 1
  }
}

# Create the gamVineCopula object
GVC <- gamVine(Matrix = Matrix, model = model, names = nnames)
print(GVC)
## Not run:
## Simulate and fit the model
sim <- gamVineSimulate(n, GVC)
fitGVC <- gamVineSeqFit(sim, GVC, verbose = TRUE)
fitGVC2 <- gamVineCopSelect(sim, Matrix, verbose = TRUE)

## Plot the results
par(mfrow = c(3, 4))
plot(GVC, ylim = c(-2.5, 2.5))
```

```
plot(fitGVC, ylim = c(-2.5, 2.5))

plot(fitGVC2, ylim = c(-2.5, 2.5))

## End(Not run)
```

---

gamVineFamily *Family Matrix of an Object of the Class gamVine*

Description

Return the matrix of copula family (see gamBiCop) corresponding to the model list in the gamVine object.

Usage

```
gamVineFamily(GVC)
```

Arguments

GVC	An object of the class gamVine.

Value

Matrix of copula families corresponding to the model list in the gamVine object.

See Also

gamVine.

---

gamVineNormalize *Normalize an Object of the Class gamVine*

Description

Change the R-vine matrix in the natural order, i.e. with d:1 on the diagonal

Usage

```
gamVineNormalize(GVC)
```

Arguments

GVC	An object of the class gamVine.

Value

The normalized gamVine object.

See Also

gamVine.

---

gamVinePDF *Conditional density function of a gamVine*

Description

This function returns the density of a conditional pair-copula constructions, where either the copula parameters or the Kendall's taus are modeled as a function of the covariates.

Usage

```
gamVinePDF(object, data)
```

Arguments

object	gamVine-class object.
data	(Same as in predict.gam from the mgcv package) A matrix or data frame containing the values of the model covariates at which predictions are required, along with a number of additional columns corresponding to the variables in the pair copula decomposition.

Value

The conditional density.

See Also

gamVine, gamVineCopSelect, gamVineStructureSelect, gamVine-class, gamVineSimulate and gamBiCopFit.

Examples

```
require(mgcv)
set.seed(0)

##  Simulation parameters
# Sample size
n <- 1e3
# Copula families
familyset <- c(1:2, 301:304, 401:404)
# Define a 4-dimensional R-vine tree structure matrix
d <- 4
```

```
Matrix <- c(2, 3, 4, 1, 0, 3, 4, 1, 0, 0, 4, 1, 0, 0, 0, 1)
Matrix <- matrix(Matrix, d, d)
nnames <- paste("X", 1:d, sep = "")

## A function factory
eta0 <- 1
calib.surf <- list(
  calib.quad <- function(t, Ti = 0, Tf = 1, b = 8) {
    Tm <- (Tf - Ti) / 2
    a <- -(b / 3) * (Tf^2 - 3 * Tf * Tm + 3 * Tm^2)
    return(a + b * (t - Tm)^2)
  },
  calib.sin <- function(t, Ti = 0, Tf = 1, b = 1, f = 1) {
    a <- b * (1 - 2 * Tf * pi / (f * Tf * pi +
      cos(2 * f * pi * (Tf - Ti))
      - cos(2 * f * pi * Ti)))
    return((a + b) / 2 + (b - a) * sin(2 * f * pi * (t - Ti)) / 2)
  },
  calib.exp <- function(t, Ti = 0, Tf = 1, b = 2, s = Tf / 8) {
    Tm <- (Tf - Ti) / 2
    a <- (b * s * sqrt(2 * pi) / Tf) * (pnorm(0, Tm, s) - pnorm(Tf, Tm, s))
    return(a + b * exp(-(t - Tm)^2 / (2 * s^2)))
  }
)

##  Create the model
# Define gam-vine model list
count <- 1
model <- vector(mode = "list", length = d * (d - 1) / 2)
sel <- seq(d, d^2 - d, by = d)

# First tree
for (i in 1:(d - 1)) {
  # Select a copula family
  family <- sample(familyset, 1)
  model[[count]]$family <- family

  # Use the canonical link and a randomly generated parameter
  if (is.element(family, c(1, 2))) {
    model[[count]]$par <- tanh(rnorm(1) / 2)
    if (family == 2) {
      model[[count]]$par2 <- 2 + exp(rnorm(1))
    }
  } else {
    if (is.element(family, c(401:404))) {
      rr <- rnorm(1)
      model[[count]]$par <- sign(rr) * (1 + abs(rr))
    } else {
      model[[count]]$par <- rnorm(1)
    }
    model[[count]]$par2 <- 0
  }
  count <- count + 1
```

```
}

# A dummy dataset
data <- data.frame(u1 = runif(1e2), u2 = runif(1e2), matrix(runif(1e2 * d), 1e2, d))

# Trees 2 to (d-1)
for (j in 2:(d - 1)) {
  for (i in 1:(d - j)) {
    # Select a copula family
    family <- sample(familyset, 1)

    # Select the conditiong set and create a model formula
    cond <- nnames[sort(Matrix[(d - j + 2):d, i])]
    tmpform <- paste("~", paste(paste("s(", cond, ", k=10, bs='cr')",
      sep = ""
    ), collapse = " + "))
    l <- length(cond)
    temp <- sample(3, l, replace = TRUE)

    # Spline approximation of the true function
    m <- 1e2
    x <- matrix(seq(0, 1, length.out = m), nrow = m, ncol = 1)
    if (l != 1) {
      tmp.fct <- paste("function(x){eta0+",
        paste(sapply(1:l, function(x)
          paste("calib.surf[[", temp[x], "]](x[", x, "])",
            sep = ""
          )), collapse = "+"), "}",
        sep = ""
      )
      tmp.fct <- eval(parse(text = tmp.fct))
      x <- eval(parse(text = paste0("expand.grid(",
        paste0(rep("x", l), collapse = ","), ")",
        collapse = ""
      )))
      y <- apply(x, 1, tmp.fct)
    } else {
      tmp.fct <- function(x) eta0 + calib.surf[[temp]](x)
      colnames(x) <- cond
      y <- tmp.fct(x)
    }

    # Estimate the gam model
    form <- as.formula(paste0("y", tmpform))
    dd <- data.frame(y, x)
    names(dd) <- c("y", cond)
    b <- gam(form, data = dd)
    # plot(x[,1],(y-fitted(b))/y)

    # Create a dummy gamBiCop object
    tmp <- gamBiCopFit(data = data, formula = form, family = 1, n.iters = 1)$res

    # Update the copula family and the model coefficients
```

```
    attr(tmp, "model")$coefficients <- coefficients(b)
    attr(tmp, "model")$smooth <- b$smooth
    attr(tmp, "family") <- family
    if (family == 2) {
      attr(tmp, "par2") <- 2 + exp(rnorm(1))
    }
    model[[count]] <- tmp
    count <- count + 1
  }
}

# Create the gamVineCopula object
GVC <- gamVine(Matrix = Matrix, model = model, names = nnames)
print(GVC)
## Not run:
## Simulate and fit the model
sim <- gamVineSimulate(n, GVC)
fitGVC <- gamVineSeqFit(sim, GVC, verbose = TRUE)
fitGVC2 <- gamVineCopSelect(sim, Matrix, verbose = TRUE)
(gamVinePDF(GVC, sim[1:10, ]))

## Plot the results
dev.off()
par(mfrow = c(3, 4))
plot(GVC, ylim = c(-2.5, 2.5))

plot(fitGVC, ylim = c(-2.5, 2.5))

plot(fitGVC2, ylim = c(-2.5, 2.5))

## End(Not run)
```

`gamVineSeqFit` *Sequential maximum penalized likelihood estimation of a GAM-Vine model.*

Description

This function estimates the parameter(s) of a Generalized Additive model (GAM) Vine model, where GAMs for individual edges are specified either for the copula parameter or Kendall's tau. It solves the maximum penalized likelihood estimation for the copula families supported in this package by reformulating each Newton-Raphson iteration as a generalized ridge regression, which is solved using the mgcv package.

Usage

```
gamVineSeqFit(
  data,
  GVC,
```

```
  covariates = NA,
  method = "FS",
  tol.rel = 0.001,
  n.iters = 10,
  verbose = FALSE,
  ...
)
```

Arguments

data	A matrix or data frame containing the data in [0,1]^d.
GVC	A gamVine object.
covariates	Vector of names for the covariates.
method	'NR' for Newton-Raphson and 'FS' for Fisher-scoring (default).
tol.rel	Relative tolerance for 'FS'/'NR' algorithm.
n.iters	Maximal number of iterations for 'FS'/'NR' algorithm.
verbose	TRUE if informations should be printed during the estimation and FALSE (default) for a silent version.
...	Additional parameters to be passed to gam from mgcv.

Value

gamVineSeqFit returns a gamVine object.

See Also

gamVineCopSelect and gamVineStructureSelect

gamVineCopSelect,gamVineStructureSelect, gamVine-class, gamVineSimulate and gamBiCopFit.

Examples

```
require(mgcv)
set.seed(0)

##  Simulation parameters
# Sample size
n <- 1e3
# Copula families
familyset <- c(1:2, 301:304, 401:404)
# Define a 4-dimensional R-vine tree structure matrix
d <- 4
Matrix <- c(2, 3, 4, 1, 0, 3, 4, 1, 0, 0, 4, 1, 0, 0, 0, 1)
Matrix <- matrix(Matrix, d, d)
nnames <- paste("X", 1:d, sep = "")

## A function factory
eta0 <- 1
calib.surf <- list(
```

```
  calib.quad <- function(t, Ti = 0, Tf = 1, b = 8) {
    Tm <- (Tf - Ti) / 2
    a <- -(b / 3) * (Tf^2 - 3 * Tf * Tm + 3 * Tm^2)
    return(a + b * (t - Tm)^2)
  },
  calib.sin <- function(t, Ti = 0, Tf = 1, b = 1, f = 1) {
    a <- b * (1 - 2 * Tf * pi / (f * Tf * pi +
      cos(2 * f * pi * (Tf - Ti))
      - cos(2 * f * pi * Ti)))
    return((a + b) / 2 + (b - a) * sin(2 * f * pi * (t - Ti)) / 2)
  },
  calib.exp <- function(t, Ti = 0, Tf = 1, b = 2, s = Tf / 8) {
    Tm <- (Tf - Ti) / 2
    a <- (b * s * sqrt(2 * pi) / Tf) * (pnorm(0, Tm, s) - pnorm(Tf, Tm, s))
    return(a + b * exp(-(t - Tm)^2 / (2 * s^2)))
  }
)

##  Create the model
# Define gam-vine model list
count <- 1
model <- vector(mode = "list", length = d * (d - 1) / 2)
sel <- seq(d, d^2 - d, by = d)

# First tree
for (i in 1:(d - 1)) {
  # Select a copula family
  family <- sample(familyset, 1)
  model[[count]]$family <- family

  # Use the canonical link and a randomly generated parameter
  if (is.element(family, c(1, 2))) {
    model[[count]]$par <- tanh(rnorm(1) / 2)
    if (family == 2) {
      model[[count]]$par2 <- 2 + exp(rnorm(1))
    }
  } else {
    if (is.element(family, c(401:404))) {
      rr <- rnorm(1)
      model[[count]]$par <- sign(rr) * (1 + abs(rr))
    } else {
      model[[count]]$par <- rnorm(1)
    }
    model[[count]]$par2 <- 0
  }
  count <- count + 1
}

# A dummy dataset
data <- data.frame(u1 = runif(1e2), u2 = runif(1e2), matrix(runif(1e2 * d), 1e2, d))

# Trees 2 to (d-1)
for (j in 2:(d - 1)) {
```

```
for (i in 1:(d - j)) {
  # Select a copula family
  family <- sample(familyset, 1)

  # Select the conditiong set and create a model formula
  cond <- nnames[sort(Matrix[(d - j + 2):d, i])]
  tmpform <- paste("~", paste(paste("s(", cond, ", k=10, bs='cr')",
    sep = ""
  ), collapse = " + "))
  l <- length(cond)
  temp <- sample(3, l, replace = TRUE)

  # Spline approximation of the true function
  m <- 1e2
  x <- matrix(seq(0, 1, length.out = m), nrow = m, ncol = 1)
  if (l != 1) {
    tmp.fct <- paste("function(x){eta0+",
      paste(sapply(1:l, function(x)
        paste("calib.surf[[", temp[x], "]](x[", x, "])",
          sep = ""
        )), collapse = "+"), "}",
      sep = ""
    )
    tmp.fct <- eval(parse(text = tmp.fct))
    x <- eval(parse(text = paste0("expand.grid(",
      paste0(rep("x", l), collapse = ","), ")",
      collapse = ""
    )))
    y <- apply(x, 1, tmp.fct)
  } else {
    tmp.fct <- function(x) eta0 + calib.surf[[temp]](x)
    colnames(x) <- cond
    y <- tmp.fct(x)
  }

  # Estimate the gam model
  form <- as.formula(paste0("y", tmpform))
  dd <- data.frame(y, x)
  names(dd) <- c("y", cond)
  b <- gam(form, data = dd)
  # plot(x[,1],(y-fitted(b))/y)

  # Create a dummy gamBiCop object
  tmp <- gamBiCopFit(data = data, formula = form, family = 1, n.iters = 1)$res

  # Update the copula family and the model coefficients
  attr(tmp, "model")$coefficients <- coefficients(b)
  attr(tmp, "model")$smooth <- b$smooth
  attr(tmp, "family") <- family
  if (family == 2) {
    attr(tmp, "par2") <- 2 + exp(rnorm(1))
  }
  model[[count]] <- tmp
```

```
    count <- count + 1
  }
}

# Create the gamVineCopula object
GVC <- gamVine(Matrix = Matrix, model = model, names = nnames)
print(GVC)
## Not run:
## Simulate and fit the model
sim <- gamVineSimulate(n, GVC)
fitGVC <- gamVineSeqFit(sim, GVC, verbose = TRUE)
fitGVC2 <- gamVineCopSelect(sim, Matrix, verbose = TRUE)
(gamVinePDF(GVC, sim[1:10, ]))

## Plot the results
dev.off()
par(mfrow = c(3, 4))
plot(GVC, ylim = c(-2.5, 2.5))

plot(fitGVC, ylim = c(-2.5, 2.5))

plot(fitGVC2, ylim = c(-2.5, 2.5))

## End(Not run)
```

---

gamVineSimulate *Simulation from a* gamVine-class *object*

Description

Simulation from a gamVine-class object

Usage

```
gamVineSimulate(n, GVC, U = NULL, newdata = NULL)
```

Arguments

n	number of d-dimensional observations to simulate.
GVC	A gamVine object.
U	If not NULL, U is an (N,d)-matrix of U[0,1] random variates to be transformed to the copula sample.
newdata	If not NULL, which is mandatory when the attribute covariates from GVC is not NA, newdata is a data frame containing the values of the model covariates at which simulations are required.

Value

A matrix of data simulated from the given gamVine object.

Examples

```
require(VineCopula)

## Example adapted from RVineSim

## Define 5-dimensional R-vine tree structure matrix
Matrix <- c(
  5, 2, 3, 1, 4,
  0, 2, 3, 4, 1,
  0, 0, 3, 4, 1,
  0, 0, 0, 4, 1,
  0, 0, 0, 0, 1
)
Matrix <- matrix(Matrix, 5, 5)

## Define R-vine pair-copula family matrix
family <- c(
  0, 1, 3, 4, 4,
  0, 0, 3, 4, 1,
  0, 0, 0, 4, 1,
  0, 0, 0, 0, 3,
  0, 0, 0, 0, 0
)
family <- matrix(family, 5, 5)

## Define R-vine pair-copula parameter matrix
par <- c(
  0, 0.2, 0.9, 1.5, 3.9,
  0, 0, 1.1, 1.6, 0.9,
  0, 0, 0, 1.9, 0.5,
  0, 0, 0, 0, 4.8,
  0, 0, 0, 0, 0
)
par <- matrix(par, 5, 5)

## Define second R-vine pair-copula parameter matrix
par2 <- matrix(0, 5, 5)

## Define RVineMatrix object
RVM <- RVineMatrix(
  Matrix = Matrix, family = family,
  par = par, par2 = par2,
  names = c("V1", "V2", "V3", "V4", "V5")
)

## Convert to gamVine object
GVC <- RVM2GVC(RVM)
```

```
## U[0,1] random variates to be transformed to the copula sample
n <- 1e2
d <- 5
U <- matrix(runif(n * d), nrow = n)

## The output of gamVineSimulate correspond to that of RVineSim
sampleRVM <- RVineSim(n, RVM, U)
sampleGVC <- gamVineSimulate(n, GVC, U)
all.equal(sampleRVM, sampleGVC)

## Fit the two models and compare the estimated parameter
fitRVM <- RVM2GVC(RVineSeqEst(sampleRVM, RVM))
fitGVC <- gamVineSeqFit(sampleGVC, GVC)
all.equal(
  simplify2array(attr(fitRVM, "model")),
  simplify2array(attr(fitGVC, "model"))
)
```

`gamVineStructureSelect` *Structure selection and estimation of a GAM-Vine model.*

Description

This function select the structure and estimates the parameter(s) of a Generalized Additive model (GAM) Vine model, where GAMs for individual edges are specified either for the copula parameter or Kendall's tau. It solves the maximum penalized likelihood estimation for the copula families supported in this package by reformulating each Newton-Raphson iteration as a generalized ridge regression, which is solved using the `mgcv` package.

Usage

```
gamVineStructureSelect(
  udata,
  lin.covs = NULL,
  smooth.covs = NULL,
  simplified = TRUE,
  type = 0,
  familyset = NA,
  rotations = TRUE,
  familycrit = "AIC",
  treecrit = "tau",
  level = 0.05,
  trunclevel = NA,
  tau = TRUE,
  method = "FS",
  tol.rel = 0.001,
  n.iters = 10,
```

```
  parallel = FALSE,
  verbose = FALSE,
  select.once = TRUE
)
```

Arguments

udata	A matrix or data frame containing the data in [0,1]^d.
lin.covs	A matrix or data frame containing the parametric (i.e., linear) covariates (default: lin.covs = NULL).
smooth.covs	A matrix or data frame containing the non-parametric (i.e., smooth) covariates (default: smooth.covs = NULL).
simplified	If TRUE (default), then a simplified vine is fitted (which is possible only if there are exogenous covariates). If FALSE, then a non-simplified vine is fitted.
type	type = 0 (default) for a R-Vine and type = 1 for a C-Vine.
familyset	An integer vector of pair-copula families to select from (the independence copula MUST NOT be specified in this vector unless one wants to fit an independence vine!). Not listed copula families might be included to better handle limit cases. If familyset = NA (default), selection among all possible families is performed. Coding of pair-copula families: 1 Gaussian, 2 Student t, 5 Frank, 301 Double Clayton type I (standard and rotated 90 degrees), 302 Double Clayton type II (standard and rotated 270 degrees), 303 Double Clayton type III (survival and rotated 90 degrees), 304 Double Clayton type IV (survival and rotated 270 degrees), 401 Double Gumbel type I (standard and rotated 90 degrees), 402 Double Gumbel type II (standard and rotated 270 degrees), 403 Double Gumbel type III (survival and rotated 90 degrees), 404 Double Gumbel type IV (survival and rotated 270 degrees).
rotations	If TRUE, all rotations of the families in familyset are included.
familycrit	Character indicating the criterion for bivariate copula selection. Possible choices: familycrit = 'AIC' (default) or 'BIC', as in BiCopSelect from the VineCopula package.
treecrit	Character indicating how pairs are selected in each tree. treecrit = "tau" uses the maximum spanning tree of the Kendall's tau (i.e., the tree of maximal overall dependence), treecrit = "rho" uses the Spearman's rho.
level	Numerical; Passed to gamBiCopSelect, it is the significance level of the test for removing individual predictors (default: level = 0.05) for each conditional pair-copula.
trunclevel	Integer; level of truncation.
tau	TRUE (default) for a calibration function specified for Kendall's tau or FALSE for a calibration function specified for the Copula parameter.
method	'NR' for Newton-Raphson and 'FS' for Fisher-scoring (default).
tol.rel	Relative tolerance for 'FS'/'NR' algorithm.
n.iters	Maximal number of iterations for 'FS'/'NR' algorithm.
parallel	TRUE (default) for parallel selection of copula family at each edge or FALSE for the sequential version. for the Copula parameter.

verbose	TRUE if informations should be printed during the estimation and FALSE (default) for a silent version. from mgcv.
select.once	if TRUE the GAM structure is only selected once, for the family that appears first in familyset.

Value

gamVineSeqFit returns a gamVine-class object.

See Also

gamVineSeqFit,gamVineCopSelect, gamVine-class, gamVineSimulate and gamBiCopSelect.

Examples

```
require(VineCopula)
set.seed(0)


## An example with a 3-dimensional GAM-Vine

# Sample size
n <- 1e3

# Define a R-vine tree structure matrix
d <- 3
Matrix <- c(2, 3, 1, 0, 3, 1, 0, 0, 1)
Matrix <- matrix(Matrix, d, d)
nnames <- paste("x", 1:d, sep = "")

# Copula families for each edge
fam <- c(301, 401, 1)

# Parameters for the first tree (two unconditional copulas)
par <- c(1, 2)

# Pre-allocate the GAM-Vine model list
count <- 1
model <- vector(mode = "list", length = d * (d - 1) / 2)

# The first tree contains only the two unconditional copulas
for (i in 1:(d - 1)) {
  model[[count]] <- list(family = fam[count], par = par[count], par2 = 0)
  count <- count + 1
}

# The second tree contains a unique conditional copula
# In this first example, we take a linear calibration function (10*x-5)

# Set-up a dummy dataset
tmp <- data.frame(u1 = runif(1e2), u2 = runif(1e2), x1 = runif(1e2))
```

```
# Set-up an arbitrary linear model for the calibration function
model[[count]] <- gamBiCopFit(tmp, ~x1, fam[count])$res

# Update the coefficients of the model
attr(model[[count]], "model")$coefficients <- c(-5, 10)

# Define gamVine object
GVC <- gamVine(Matrix = Matrix, model = model, names = nnames)
GVC
## Not run:
# Simulate new data
simData <- data.frame(gamVineSimulate(n, GVC))
colnames(simData) <- nnames

# Fit data using sequential estimation assuming true model known
summary(fitGVC <- gamVineSeqFit(simData, GVC))

# Fit data using structure selection and sequential estimation
summary(fitGVC2 <- gamVineStructureSelect(simData, simplified = FALSE))

## End(Not run)
```

`logLik.gamBiCop` *Extract the Log-likelihood from a gamBiCop Object*

Description

Function to extract the log-likelihood from an object of the class `gamBiCop` (note that the models are usually fitted by penalized likelihood maximization). This function is used by `AIC` and `BIC`.

Usage

```
## S4 method for signature 'gamBiCop'
logLik(object, ...)
```

Arguments

`object`	An object of the class `gamBiCop`.
`...`	un-used in this class

Value

Standard `logLik` object: see `logLik`.

See Also

`AIC` and `BIC`.

nobs.gamBiCop *Extract the Number of Observations from gamBiCop Object*

Description

Extract the number of 'observations' from a model fit. This is principally intended to be used in computing the BIC (see AIC).

Usage

```
## S4 method for signature 'gamBiCop'
nobs(object, ...)
```

Arguments

object	An object of the class gamBiCop.
...	un-used in this class

Value

A single number, normally an integer.

See Also

AIC and BIC.

plot.gamBiCop *Plot a gamBiCop Object*

Description

Plot from an object of the class gamBiCop. The function is based on (see plot.gam from mgcv).

Usage

```
## S4 method for signature 'gamBiCop,ANY'
plot(x, y, ...)
```

Arguments

x	An object of the class gamBiCop.
y	Not used with this class.
...	additional arguments to be passed to plot.gam.

Value

This function simply generates plots.

See Also

plot.gam from mgcv).

---

plot.gamVine *Plot an Object of the Class gamVine*

Description

Plot an object of the class gamVine. The function is based on (see plot.gam from mgcv).

Usage

```
## S4 method for signature 'gamVine,ANY'
plot(x, y, ...)
```

Arguments

x	An object of the class gamVine.
y	Not used with this class.
...	additional arguments to be passed to plot.gam.

Value

This function simply generates plots.

See Also

plot.gam from mgcv).

RVM2GVC *Transform an Object of the Class R-Vine into an Object of the Class gamVine*

Description

Transform an object of the class RVineMatrix into an object of the class gamVine.

Usage

```
RVM2GVC(RVM)
```

Arguments

RVM	An object of the class RVineMatrix.

Value

An object of the class gamVine.

See Also

RVineMatrix and gamVine.

---

summary.gamBiCop *Summary for a gamBiCop Object*

Description

Takes a gamBiCop object and produces various useful summaries from it.

Usage

```
## S4 method for signature 'gamBiCop'
summary(object, ...)
```

Arguments

object	An object of the class gamBiCop.
...	unused in this class

Value

A useful summary (see summary.gam from mgcv for more details).

See Also

summary.gam from mgcv

summary.gamVine *Summary for an Object of the Class gamVine*

Description

Takes an object of the class gamVine and produces various useful summaries from it.

Usage

```
## S4 method for signature 'gamVine'
summary(object, ...)
```

Arguments

object	An object of the class gamVine.
...	unused in this class

Value

A useful summary (see summary.gam from mgcv for more details).

See Also

summary.gam from mgcv

Index