Package ‘EllipCopulas’

January 20, 2025

Type Package

Title Inference of Elliptical Distributions and Copulas

Version 0.1.4.1

Description Provides functions for the simulation and
the nonparametric estimation of elliptical distributions,
meta-elliptical copulas and trans-elliptical distributions,
following the article Derumigny and Fermanian (2022) <doi:10.1016/j.jmva.2022.104962>.

License GPL-3

Encoding UTF-8

LazyLoad yes

RoxygenNote 7.3.2

Depends R (>= 3.6.0)

Imports Runuran, wdm, Matrix, kStatistics, pbapply

Suggests mvtnorm, Rmpfr, testthat (>= 3.0.0)

BugReports https://github.com/AlexisDerumigny/ElliptCopulas/issues

URL https://github.com/AlexisDerumigny/ElliptCopulas

Config/testthat/edition 3

NeedsCompilation no

Author Alexis Derumigny [aut, cre] (<https://orcid.org/0000-0002-6163-8097>),
Jean-David Fermanian [aut] (<https://orcid.org/0000-0001-5960-5555>),
Victor Ryan [aut],
Rutger van der Spek [ctb]

Maintainer Alexis Derumigny <a.f.f.derumigny@tudelft.nl>

Repository CRAN

Date/Publication 2024-09-09 15:10:06 UTC

Contents

`conv_funct` *Conversion Functions for Elliptical Distributions*

Description

An elliptical random vector X of density $|det(\Sigma)|^{-1/2} g_d(x'\Sigma^{-1}x)$ can always be written as $X = \mu + R * A * U$ for some positive random variable R and a random vector U on the d-dimensional sphere. Furthermore, there is a one-to-one mapping between g_d and its one-dimensional marginal g_1.

Usage

```
Convert_gd_To_g1(grid, g_d, d)

Convert_g1_To_Fg1(grid, g_1)

Convert_g1_To_Qg1(grid, g_1)

Convert_g1_To_f1(grid, g_1)

Convert_gd_To_fR2(grid, g_d, d)
```

Arguments

`grid`	the grid on which the values of the functions in parameter are given.
`g_d`	the d-dimensional density generator.
`d`	the dimension of the random vector.
`g_1`	the 1-dimensional density generator.

Value

One of the following

- g_1 the 1-dimensional density generator.
- Fg1 the 1-dimensional marginal cumulative distribution function.
- Qg1 the 1-dimensional marginal quantile function (approximately equal to the inverse function of Fg1).
- f1 the density of a 1-dimensional margin if $\mu = 0$ and A is the identity matrix.
- fR2 the density function of R^2.

The function Convert_gd_To_g1 returns a numerical vector of (approximated) values of g_1 on the same grid as gd. In all other cases, a function is returned (see the examples section).

See Also

DensityGenerator.normalize to compute the normalized version of a given d-dimensional generator.

Examples

```
grid = seq(0,100,by = 0.01)
g_d = DensityGenerator.normalize(grid = grid, grid_g = 1/(1+grid^3), d = 3)
g_1 = Convert_gd_To_g1(grid = grid, g_d = g_d, d = 3)
Fg_1 = Convert_g1_To_Fg1(grid = grid, g_1 = g_1)
Qg_1 = Convert_g1_To_Qg1(grid = grid, g_1 = g_1)
f1 = Convert_g1_To_f1(grid = grid, g_1 = g_1)
fR2 = Convert_gd_To_fR2(grid = grid, g_d = g_d, d = 3)
plot(grid, g_d, type = "l", xlim = c(0,10))
plot(grid, g_1, type = "l", xlim = c(0,10))
plot(Fg_1, xlim = c(-3,3))
plot(Qg_1, xlim = c(0.01,0.99))
plot(f1, xlim = c(-3,3))
plot(fR2, xlim = c(0,3))
```

DensityGenerator.normalize

Normalization of an elliptical copula generator

Description

The function DensityGenerator.normalize transforms an elliptical copula generator into an elliptical copula generator,generating the same distribution and which is normalized to follow the normalization constraint

$$\frac{\pi^{d/2}}{\Gamma(d/2)} \int_0^{+\infty} g_k(t) t^{(d-2)/2} dt = 1.$$

as well as the identification constraint

$$\frac{\pi^{(d-1)/2}}{\Gamma((d-1)/2)} \int_0^{+\infty} g_k(t) t^{(d-3)/2} dt = b.$$

The function `DensityGenerator.check` checks, for a given generator, whether these two constraints are satisfied.

Usage

```
DensityGenerator.normalize(grid, grid_g, d, verbose = 0, b = 1)

DensityGenerator.check(grid, grid_g, d, b = 1)
```

Arguments

`grid`	the regularly spaced grid on which the values of the generator are given.
`grid_g`	the values of the d-dimensional generator at points of the grid.
`d`	the dimension of the space.
`verbose`	if 1, prints the estimated (alpha, beta) such that `new_g(t) = alpha * old_g(beta*t)`.
`b`	the target value for the identification constraint.

Value

`DensityGenerator.normalize` returns the normalized generator, as a list of values on the same `grid`.

`DensityGenerator.check` returns (invisibly) a vector of two booleans where the first element is `TRUE` if the normalization constraint is satisfied and the second element is `TRUE` if the identification constraint is satisfied.

References

Derumigny, A., & Fermanian, J. D. (2022). Identifiability and estimation of meta-elliptical copula generators. Journal of Multivariate Analysis, article 104962. doi:10.1016/j.jmva.2022.104962.

See Also

`EllCopSim()` for the simulation of elliptical copula samples, `EllCopEst()` for the estimation of elliptical copula, conversion functions for the conversion between different representation of the generator of an elliptical copula.

EllCopEst *Estimate the density generator of a (meta-)elliptical copula*

Description

This function estimates the density generator of a (meta-)elliptical copula using the iterative procedure described in (Derumigny and Fermanian, 2022). This iterative procedure consists in alternating a step of estimating the data via Liebscher's procedure `EllDistrEst()` and estimating the quantile function of the underlying elliptical distribution to transform the data back to the unit cube.

Usage

```
EllCopEst(
  dataU,
  Sigma_m1,
  h,
  grid = seq(0, 10, by = 0.01),
  niter = 10,
  a = 1,
  Kernel = "epanechnikov",
  verbose = 1,
  startPoint = "identity",
  prenormalization = FALSE
)
```

Arguments

`dataU`	the data matrix on the $[0, 1]$ scale.
`Sigma_m1`	the inverse of the correlation matrix of the components of data
`h`	bandwidth of the kernel for Liebscher's procedure
`grid`	the grid at which the density generator is estimated.
`niter`	the number of iterations
`a`	tuning parameter to improve the performance at 0. See Liebscher (2005), Example p.210
`Kernel`	kernel used for the smoothing. Possible choices are `gaussian`, `epanechnikov` and `triangular`.
`verbose`	if 1, prints the progress of the iterations. If 2, prints the normalization constants used at each iteration, as computed by `DensityGenerator.normalize`.
`startPoint`	is the given starting point of the procedure • `startPoint = "gaussian"` for using the gaussian generator as starting point ; • `startPoint = "identity"` for a data-driven starting point ; • `startPoint = "A~Phi^{-1}"` for another data-driven starting point using the Gaussian quantile function.

prenormalization
: if TRUE, the procedure will normalize the variables at each iteration so that the variance is 1.

Value

a list of two elements:

- g_d_norm: the estimated elliptical copula generator at each point of the grid;
- list_path_gdh: the list of estimated elliptical copula generator at each iteration.

References

Derumigny, A., & Fermanian, J. D. (2022). Identifiability and estimation of meta-elliptical copula generators. Journal of Multivariate Analysis, article 104962. doi:10.1016/j.jmva.2022.104962.

Liebscher, E. (2005). A semiparametric density estimator based on elliptical distributions. Journal of Multivariate Analysis, 92(1), 205. doi:10.1016/j.jmva.2003.09.007

See Also

EllDistrEst for the estimation of elliptical distributions, EllCopSim for the simulation of elliptical copula samples, EllCopLikelihood for the computation of the likelihood of a given generator, DensityGenerator.normalize to compute the normalized version of a given generator.

Examples

```
# Simulation from a Gaussian copula
grid = seq(0,10,by = 0.01)
g_d = DensityGenerator.normalize(grid, grid_g = exp(-grid), d = 3)
n = 10
# To have a nice estimation, we suggest to use rather n=200
# (around 20s of computation time)
U = EllCopSim(n = n, d = 3, grid = grid, g_d = g_d)
result = EllCopEst(dataU = U, grid, Sigma_m1 = diag(3),
                   h = 0.1, a = 0.5)
plot(grid, g_d, type = "l", xlim = c(0,2))
lines(grid, result$g_d_norm, col = "red", xlim = c(0,2))

# Adding missing observations
n_NA = 2
U_NA = U
for (i in 1:n_NA){
  U_NA[sample.int(n,1), sample.int(3,1)] = NA
}
resultNA = EllCopEst(dataU = U_NA, grid, Sigma_m1 = diag(3),
                     h = 0.1, a = 0.5)
lines(grid, resultNA$g_d_norm, col = "blue", xlim = c(0,2))
```

`EllCopLikelihood` *Computation of the likelihood of an elliptical copula*

Description

Computes the likelihood

$$\frac{g(Q_g(U)\Sigma^{-1}Q_g(U))}{f_g(Q_g(U_1))\cdots f_g(Q_g(U_d))}$$

for a vector $(U_1,\dots,U_d)$ on the unit cube and for a d-dimensional generator g whose univariate density and quantile functions are respectively f_g and Q_g. This is to the likelihood of the copula associated with the elliptical distribution having density $|det(\Sigma)|^{-1/2}g(x\Sigma^{-1}x)$.

Usage

```
EllCopLikelihood(grid, g_d, pointsToCompute, Sigma_m1, log = TRUE)
```

Arguments

`grid`	the discretization grid on which the generator is given.
`g_d`	the values of the d-dimensional density generator on the grid.
`pointsToCompute`	the points U at which the likelihood should be computed. If `pointsToCompute` is a vector, then its length is used as the dimension d of the space. If it is a matrix, then the dimension of the space is the number of columns.
`Sigma_m1`	the inverse correlation matrix of the elliptical distribution.
`log`	if `TRUE`, this returns the log-likelihood instead of the likelihood.

Value

a vector (of length 1 if `pointsToCompute` is a vector) of likelihoods associated with each observation.

References

Derumigny, A., & Fermanian, J. D. (2022). Identifiability and estimation of meta-elliptical copula generators. Journal of Multivariate Analysis, article 104962. doi:10.1016/j.jmva.2022.104962.

See Also

`EllCopEst` for the estimation of elliptical copula, `EllCopEst` for the estimation of elliptical copula.

Examples

```
grid = seq(0,50,by = 0.01)
gdnorm = DensityGenerator.normalize(grid = grid, grid_g = exp(-grid/2), d = 3)
gdnorm2 = DensityGenerator.normalize(grid = grid, grid_g = 1/(1+grid^2), d = 3)
X = EllCopSim(n = 30, d = 3, grid = grid, g_d = gdnorm)
logLik = EllCopLikelihood(grid , g_d = gdnorm , X,
                          Sigma_m1 = diag(3), log = TRUE)
logLik2 = EllCopLikelihood(grid , g_d = gdnorm2 , X,
                           Sigma_m1 = diag(3), log = TRUE)
print(c(sum(logLik), sum(logLik2)))
```

EllCopSim — *Simulation from an elliptical copula model*

Description

Simulation from an elliptical copula model

Usage

```
EllCopSim(n, d, grid, g_d, A = diag(d), genR = list(method = "pinv"))
```

Arguments

`n`	number of observations.
`d`	dimension of X.
`grid`	grid on which values of density generator are known.
`g_d`	vector of values of the density generator on the `grid`.
`A`	square-root of the correlation matrix of X.
`genR`	additional arguments for the generation of the squared radius. It must be a list with a component method: • If `genR$method == "pinv"`, the radius is generated using the function `Runuran::pinv.new()`. • If `genR$method == "MH"`, the generation is done using the Metropolis-Hasting algorithm, with a N(0,1) move at each step.

Value

a matrix of size `(n,d)` with `n` observations of the `d`-dimensional elliptical copula.

References

Derumigny, A., & Fermanian, J. D. (2022). Identifiability and estimation of meta-elliptical copula generators. Journal of Multivariate Analysis, article 104962. doi:10.1016/j.jmva.2022.104962.

See Also

EllDistrSim for the simulation of elliptical distributions samples, EllCopEst for the estimation of elliptical copula, EllCopLikelihood for the computation of the likelihood of a given generator, DensityGenerator.normalize to compute the normalized version of a given generator.

Examples

```
# Simulation from a Gaussian copula
grid = seq(0,5,by = 0.01)
X = EllCopSim(n = 20, d = 2, grid = grid, g_d = exp(-grid/2))
X = EllCopSim(n = 20, d = 2, grid = grid, g_d = exp(-grid/2),
              genR = list(method = "MH", niter = 500) )
plot(X)
```

EllDistrDerivEst *Estimate the derivatives of a generator*

Description

A continuous elliptical distribution has a density of the form

$$f_X(x) = |\Sigma|^{-1/2} g\left((x-\mu)^\top \Sigma^{-1} (x-\mu) \right),$$

where $x \in \mathbb{R}^d$, $\mu \in \mathbb{R}^d$ is the mean, Σ is a $d \times d$ positive-definite matrix and a function $g : \mathbb{R}_+ \to \mathbb{R}_+$, called the density generator of X. The goal is to estimate the derivatives of g at some point ξ, by kernel smoothing, following Section 3 of (Ryan and Derumigny, 2024).

Usage

```
EllDistrDerivEst(
  X,
  mu = 0,
  Sigma_m1 = diag(NCOL(X)),
  grid,
  h,
  Kernel = "gaussian",
  a = 1,
  k,
  mpfr = FALSE,
  precBits = 100,
  dopb = TRUE
)
```

Arguments

- `X` — a matrix of size $n \times d$, assumed to be n i.i.d. observations (rows) of a d-dimensional elliptical distribution.
- `mu` — mean of X. This can be the true value or an estimate. It must be a vector of dimension d.
- `Sigma_m1` — inverse of the covariance matrix of X. This can be the true value or an estimate. It must be a matrix of dimension $d \times d$.
- `grid` — grid of values on which to estimate the density generator.
- `h` — bandwidth of the kernel. Can be either a number or a vector of the size `length(grid)`.
- `Kernel` — name of the kernel. Possible choices are `"gaussian"`, `"epanechnikov"`, `"triangular"`.
- `a` — tuning parameter to improve the performance at 0.
- `k` — highest order of the derivative of the generator that is to be estimated. For example, `k = 1` corresponds to the estimation of the generator and of its derivative. `k = 2` corresponds to the estimation of the generator as well as its first and second derivatives.
- `mpfr` — if `mpfr = TRUE`, multiple precision floating point is used via the package Rmpfr. This allows for a higher (numerical) accuracy, at the expense of computing time. It is recommended to use this option for higher dimensions.
- `precBits` — number of precBits used for floating point precision (only used if `mpfr = TRUE`).
- `dopb` — a Boolean value. If `dopb = TRUE`, a progress bar is displayed.

Details

Note that this function may be rather slow for higher-order derivatives. Furthermore, it is likely that the number of observations needs to be quite high for the higher-order derivatives to be estimated well enough.

Value

a matrix of size `length(grid) * (kmax + 1)` with the estimated value of the generator and all its derivatives at all orders until and including `kmax`, at all points of the grid.

Author(s)

Alexis Derumigny, Victor Ryan

Victor Ryan, Alexis Derumigny

References

Ryan, V., & Derumigny, A. (2024). On the choice of the two tuning parameters for nonparametric estimation of an elliptical distribution generator arxiv:2408.17087.

See Also

`EllDistrEst` for the nonparametric estimation of the elliptical distribution density generator itself, `EllDistrSim` for the simulation of elliptical distribution samples.

This function uses the internal functions `compute_etahat` and `compute_matrix_alpha`.

Examples

```
# Comparison between the estimated and true generator of the Gaussian distribution
n = 50000
d = 3
X = matrix(rnorm(n * d), ncol = d)
grid = seq(0, 5, by = 0.1)
a = 1.5

gprimeEst = EllDistrDerivEst(X = X, grid = grid, a = a, h = 0.09, k = 1)[,2]
plot(grid, gprimeEst, type = "l")

# Computation of true values
g = exp(-grid/2)/(2*pi)^{3/2}
gprime = (-1/2) * exp(-grid/2)/(2*pi)^{3/2}

lines(grid, gprime, col = "red")
```

`EllDistrEst` *Nonparametric estimation of the density generator of an elliptical distribution*

Description

This function uses Liebscher's algorithm to estimate the density generator of an elliptical distribution by kernel smoothing. A continuous elliptical distribution has a density of the form

$$f_X(x) = |\Sigma|^{-1/2} g\left((x-\mu)^\top \Sigma^{-1} (x-\mu) \right),$$

where $x \in \mathbb{R}^d$, $\mu \in \mathbb{R}^d$ is the mean, Σ is a $d \times d$ positive-definite matrix and a function $g : \mathbb{R}_+ \to \mathbb{R}_+$, called the density generator of X. The goal is to estimate g at some point ξ, by

$$\widehat{g}_{n,h,a}(\xi) := \frac{\xi^{\frac{-d+2}{2}} \psi_a'(\xi)}{nhs_d} \sum_{i=1}^{n} K\left(\frac{\psi_a(\xi) - \psi_a(\xi_i)}{h} \right) + K\left(\frac{\psi_a(\xi) + \psi_a(\xi_i)}{h} \right),$$

where $s_d := \pi^{d/2}/\Gamma(d/2)$, Γ is the Gamma function, h and a are tuning parameters (respectively the bandwidth and a parameter controlling the bias at $\xi = 0$), $\psi_a(\xi) := -a + (a^{d/2} + \xi^{d/2})^{2/d}$, $\xi \in \mathbb{R}$, K is a kernel function and $\xi_i := (X_i - \mu)^\top \Sigma^{-1} (X_i - \mu)$, for a sample $X_1, \dots, X_n$.

Usage

```
EllDistrEst(
  X,
  mu = 0,
  Sigma_m1 = diag(d),
  grid,
  h,
  Kernel = "epanechnikov",
```

```
  a = 1,
  mpfr = FALSE,
  precBits = 100,
  dopb = TRUE
)
```

Arguments

X	a matrix of size $n \times d$, assumed to be n i.i.d. observations (rows) of a d-dimensional elliptical distribution.
mu	mean of X. This can be the true value or an estimate. It must be a vector of dimension d.
Sigma_m1	inverse of the covariance matrix of X. This can be the true value or an estimate. It must be a matrix of dimension $d \times d$.
grid	grid of values of ξ at which we want to estimate the density generator.
h	bandwidth of the kernel. Can be either a number or a vector of the size `length(grid)`.
Kernel	name of the kernel. Possible choices are `"gaussian"`, `"epanechnikov"`, `"triangular"`.
a	tuning parameter to improve the performance at 0. Can be either a number or a vector of the size `length(grid)`. If this is a vector, the code will need to allocate a matrix of size `nrow(X) * length(grid)` which can be prohibitive in some cases.
mpfr	if `mpfr = TRUE`, multiple precision floating point is used via the package Rmpfr. This allows for a higher (numerical) accuracy, at the expense of computing time. It is recommended to use this option for higher dimensions.
precBits	number of precBits used for floating point precision (only used if `mpfr = TRUE`).
dopb	a Boolean value. If `dopb = TRUE`, a progress bar is displayed.

Value

the values of the density generator of the elliptical copula, estimated at each point of the `grid`.

Author(s)

Alexis Derumigny, Rutger van der Spek

References

Liebscher, E. (2005). A semiparametric density estimator based on elliptical distributions. Journal of Multivariate Analysis, 92(1), 205. doi:10.1016/j.jmva.2003.09.007

The function ψ_a is introduced in Liebscher (2005), Example p.210.

See Also

- `EllDistrSim` for the simulation of elliptical distribution samples.
- `estim_tilde_AMSE` for the estimation of a component of the asymptotic mean-square error (AMSE) of this estimator $\widehat{g}_{n,h,a}(\xi)$, assuming h has been optimally chosen.

- EllDistrEst.adapt for the adaptive nonparametric estimation of the generator of an elliptical distribution.
- EllDistrDerivEst for the nonparametric estimation of the derivatives of the generator.
- EllCopEst for the estimation of elliptical copulas density generators.

Examples

```
# Comparison between the estimated and true generator of the Gaussian distribution
X = matrix(rnorm(500*3), ncol = 3)
grid = seq(0,5,by=0.1)
g_3 = EllDistrEst(X = X, grid = grid, a = 0.7, h=0.05)
g_3mpfr = EllDistrEst(X = X, grid = grid, a = 0.7, h=0.05,
                      mpfr = TRUE, precBits = 20)
plot(grid, g_3, type = "l")
lines(grid, exp(-grid/2)/(2*pi)^{3/2}, col = "red")

# In higher dimensions

d = 250
X = matrix(rnorm(500*d), ncol = d)
grid = seq(0, 400, by = 25)
true_g = exp(-grid/2) / (2*pi)^{d/2}

g_d = EllDistrEst(X = X, grid = grid, a = 100, h=40)

g_dmpfr = EllDistrEst(X = X, grid = grid, a = 100, h=40,
                      mpfr = TRUE, precBits = 10000)
ylim = c(min(c(true_g, as.numeric(g_dmpfr[which(g_dmpfr>0)]))),
         max(c(true_g, as.numeric(g_dmpfr)), na.rm=TRUE) )
plot(grid, g_dmpfr, type = "l", col = "red", ylim = ylim, log = "y")
lines(grid, g_d, type = "l")
lines(grid, true_g, col = "blue")
```

EllDistrEst.adapt — *Estimation of the generator of the elliptical distribution by kernel smoothing with adaptive choice of the bandwidth*

Description

A continuous elliptical distribution has a density of the form

$$f_X(x) = |\Sigma|^{-1/2} g\left((x-\mu)^\top \Sigma^{-1} (x-\mu) \right),$$

where $x \in \mathbb{R}^d$, $\mu \in \mathbb{R}^d$ is the mean, Σ is a $d \times d$ positive-definite matrix and a function $g : \mathbb{R}_+ \to \mathbb{R}_+$, called the density generator of X. The goal is to estimate g at some point ξ, by

$$\widehat{g}_{n,h,a}(\xi) := \frac{\xi^{\frac{-d+2}{2}} \psi_a'(\xi)}{nhs_d} \sum_{i=1}^n K\left(\frac{\psi_a(\xi) - \psi_a(\xi_i)}{h} \right) + K\left(\frac{\psi_a(\xi) + \psi_a(\xi_i)}{h} \right),$$

where $s_d := \pi^{d/2}/\Gamma(d/2)$, Γ is the Gamma function, h and a are tuning parameters (respectively the bandwidth and a parameter controlling the bias at $\xi = 0$), $\psi_a(\xi) := -a + (a^{d/2} + \xi^{d/2})^{2/d}$, $\xi \in \mathbb{R}$, K is a kernel function and $\xi_i := (X_i - \mu)^\top \Sigma^{-1} (X_i - \mu)$, for a sample $X_1, \dots, X_n$. This function computes "optimal asymptotic" values for the bandwidth h and the tuning parameter a from a first step bandwidth that the user needs to provide.

Usage

```
EllDistrEst.adapt(
  X,
  mu = 0,
  Sigma_m1 = diag(NCOL(X)),
  grid,
  h_firstStep,
  grid_a = NULL,
  Kernel = "gaussian",
  mpfr = FALSE,
  precBits = 100,
  dopb = TRUE
)
```

Arguments

X	a matrix of size $n \times d$, assumed to be n i.i.d. observations (rows) of a d-dimensional elliptical distribution.
mu	mean of X. This can be the true value or an estimate. It must be a vector of dimension d.
Sigma_m1	inverse of the covariance matrix of X. This can be the true value or an estimate. It must be a matrix of dimension $d \times d$.
grid	vector containing the values at which we want the generator to be estimated.
h_firstStep	a vector of size 2 containing first-step bandwidths to be used. The first one is used for the estimation of the asymptotic mean-squared error. The second one is used for the first step estimation of g. From these two estimators, a final value of the bandwidth h is determined, which is used for the final estimator of g. If `h_firstStep` is of length 1, its value is reused for both purposes (estimation of the AMSE and first-step estimation of g).
grid_a	the grid of possible values of a to be used. If missing, a default sequence is used.
Kernel	name of the kernel. Possible choices are "gaussian", "epanechnikov", "triangular".
mpfr	if `mpfr = TRUE`, multiple precision floating point is used via the package Rmpfr. This allows for a higher (numerical) accuracy, at the expense of computing time. It is recommended to use this option for higher dimensions.
precBits	number of precBits used for floating point precision (only used if `mpfr = TRUE`).
dopb	a Boolean value. If `dopb = TRUE`, a progress bar is displayed.

Value

a list with the following elements:

- g a vector of size n1 = length(grid). Each component of this vector is an estimator of $g(x[i])$ where x[i] is the i-th element of the grid.
- best_a a vector of the same size as grid indicating for each value of the grid what is the optimal choice of a found by our algorithm (which is used to estimate g).
- best_h a vector of the same size as grid indicating for each value of the grid what is the optimal choice of h found by our algorithm (which is used to estimate g).
- first_step_g first step estimator of g, computed using the tuning parameters best_a and h_firstStep[2].
- AMSE_estimated an estimator of the part of the asymptotic MSE that only depends on a.

Author(s)

Alexis Derumigny, Victor Ryan

References

Ryan, V., & Derumigny, A. (2024). On the choice of the two tuning parameters for nonparametric estimation of an elliptical distribution generator arxiv:2408.17087.

See Also

EllDistrEst for the nonparametric estimation of the elliptical distribution density generator, EllDistrSim for the simulation of elliptical distribution samples.

estim_tilde_AMSE which is used in this function. It estimates a component of the asymptotic mean-square error (AMSE) of the nonparametric estimator of the elliptical density generator assuming h has been optimally chosen.

Examples

```
n = 500
d = 3
X = matrix(rnorm(n * d), ncol = d)
grid = seq(0, 5, by = 0.1)

result = EllDistrEst.adapt(X = X, grid = grid, h = 0.05)
plot(grid, result$g, type = "l")
lines(grid, result$first_step_g, col = "blue")

# Computation of true values
g = exp(-grid/2)/(2*pi)^{3/2}
lines(grid, g, type = "l", col = "red")

plot(grid, result$best_a, type = "l", col = "red")
plot(grid, result$best_h, type = "l", col = "red")

sum((g - result$g)^2, na.rm = TRUE) < sum((g - result$first_step_g)^2, na.rm = TRUE)
```

EllDistrSim *Simulation of elliptically symmetric random vectors*

Description

This function uses the decomposition $X = \mu + R * A * U$ where μ is the mean of X, R is the random radius, A is the square-root of the covariance matrix of X, and U is a uniform random variable of the d-dimensional unit sphere. Note that R is generated using the Metropolis-Hasting algorithm.

Usage

```
EllDistrSim(
  n,
  d,
  A = diag(d),
  mu = 0,
  density_R2,
  genR = list(method = "pinv")
)
```

Arguments

`n`	number of observations.
`d`	dimension of X.
`A`	square-root of the covariance matrix of X.
`mu`	mean of X. It should be a vector of size `d`.
`density_R2`	density of the random variable R^2, i.e. the density of the $\|\|X\|\|_2^2$ if $\mu = 0$ and A is the identity matrix. Note that this function **must** return 0 for negative inputs, otherwise negative values of R^2 may be generated. The simplest way to do this is to add `* (x > 0)` at the end of the return value of the provided `density_R2` function (see example below).
`genR`	additional arguments for the generation of the squared radius. It must be a list with a component method: • If `genR$method == "pinv"`, the radius is generated using the function `Runuran::pinv.new()`. • If `genR$method == "MH"`, the generation is done using the Metropolis-Hasting algorithm, with a $N(0, 1)$ move at each step.

Value

a matrix of dimensions `(n,d)` of simulated observations.

See Also

`EllCopSim` for the simulation of elliptical copula samples, `EllCopEst` for the estimation of elliptical distributions, `EllDistrSimCond` for the conditional simulation of elliptically distributed random vectors given some observe components.

Examples

```
# Sample from a 3-dimensional normal distribution
X = EllDistrSim(n = 200, d = 3, density_R2 = function(x){stats::dchisq(x=x,df=3)})
plot(X[,1], X[,2])
X = EllDistrSim(n = 200, d = 3, density_R2 = function(x){stats::dchisq(x=x,df=3)},
                genR = list(method = "MH", niter = 500))
plot(X[,1], X[,2])

# Sample from an Elliptical distribution for which the squared radius
# follows an exponential distribution
cov1 = rbind(c(1,0.5), c(0.5,1))
X = EllDistrSim(n = 1000, d = 2,
                A = chol(cov1), mu = c(2,6),
                density_R2 = function(x){return(exp(-x) * (x > 0))} )
```

EllDistrSimCond — *Simulation of elliptically symmetric random vectors conditionally to some observed part.*

Description

Simulation of elliptically symmetric random vectors conditionally to some observed part.

Usage

```
EllDistrSimCond(
  n,
  xobs,
  d,
  Sigma = diag(d),
  mu = 0,
  density_R2_,
  genR = list(method = "pinv")
)
```

Arguments

`n`	number of observations to be simulated from the conditional distribution.
`xobs`	observed value of X that we condition on. `NA` represent unknown components of the vectors to be simulated.
`d`	dimension of the random vector
`Sigma`	(unconditional) covariance matrix
`mu`	(unconditional) mean
`density_R2_`	(unconditional) density of the squared radius.

genR additional arguments for the generation of the squared radius. It must be a list with a component method:

- If `genR$method == "pinv"`, the radius is generated using the function `Runuran::pinv.new()`.
- If `genR$method == "MH"`, the generation is done using the Metropolis-Hasting algorithm, with a N(0,1) move at each step.

Value

a matrix of size (n,d) of simulated observations.

References

Cambanis, S., Huang, S., & Simons, G. (1981). On the Theory of Elliptically Contoured Distributions, Journal of Multivariate Analysis. (Corollary 5, p.376)

See Also

`EllDistrSim` for the (unconditional) simulation of elliptically distributed random vectors.

Examples

```
d = 3
Sigma = rbind(c(1, 0.8, 0.9),
              c(0.8, 1, 0.7),
              c(0.9, 0.7, 1))
mu = c(0, 0, 0)
result = EllDistrSimCond(n = 100, xobs = c(NA, 2, NA), d = d,
  Sigma = Sigma, mu = mu, density_R2_ = function(x){stats::dchisq(x=x,df=3)})
plot(result)

result2 = EllDistrSimCond(n = 1000, xobs = c(1.3, 2, NA), d = d,
  Sigma = Sigma, mu = mu, density_R2_ = function(x){stats::dchisq(x=x,df=3)})
hist(result2)
```

`estim_tilde_AMSE` *Estimate the part of the AMSE of the elliptical density generator that only depends on the parameter "a" assuming h has been optimally chosen*

Description

A continuous elliptical distribution has a density of the form

$$f_X(x) = |\Sigma|^{-1/2} g\left((x-\mu)^\top \Sigma^{-1} (x-\mu) \right),$$

where $x \in \mathbb{R}^d$, $\mu \in \mathbb{R}^d$ is the mean, Σ is a $d \times d$ positive-definite matrix and a function $g : \mathbb{R}_+ \to \mathbb{R}_+$, called the density generator of X. The goal is to estimate g at some point ξ, by

$$\widehat{g}_{n,h,a}(\xi) := \frac{\xi^{\frac{-d+2}{2}} \psi_a'(\xi)}{nhs_d} \sum_{i=1}^{n} K\left(\frac{\psi_a(\xi) - \psi_a(\xi_i)}{h}\right) + K\left(\frac{\psi_a(\xi) + \psi_a(\xi_i)}{h}\right),$$

where $s_d := \pi^{d/2}/\Gamma(d/2)$, Γ is the Gamma function, h and a are tuning parameters (respectively the bandwidth and a parameter controlling the bias at $\xi = 0$), $\psi_a(\xi) := -a + (a^{d/2} + \xi^{d/2})^{2/d}$, $\xi \in \mathbb{R}$, K is a kernel function and $\xi_i := (X_i - \mu)^\top \Sigma^{-1} (X_i - \mu)$, for a sample $X_1, \dots, X_n$. Thanks to Proposition 2.2 in (Ryan and Derumigny, 2024), the asymptotic mean square error of $\widehat{g}_{n,h,a}(\xi)$ can be decomposed into a product of a constant (that depends on the true g) and a term that depends on g and a. This function computes this term. It can be useful to find out the best value of the parameter a to be used.

Usage

```
estim_tilde_AMSE(
  X,
  mu = 0,
  Sigma_m1 = diag(NCOL(X)),
  grid,
  h,
  Kernel = "gaussian",
  a = 1,
  mpfr = FALSE,
  precBits = 100,
  dopb = TRUE
)
```

Arguments

`X`	a matrix of size $n \times d$, assumed to be n i.i.d. observations (rows) of a d-dimensional elliptical distribution.
`mu`	mean of X. This can be the true value or an estimate. It must be a vector of dimension d.
`Sigma_m1`	inverse of the covariance matrix of X. This can be the true value or an estimate. It must be a matrix of dimension $d \times d$.
`grid`	grid of values of ξ at which we want to estimate the density generator.
`h`	bandwidth of the kernel. Can be either a number or a vector of the size `length(grid)`.
`Kernel`	name of the kernel. Possible choices are `"gaussian"`, `"epanechnikov"`, `"triangular"`.
`a`	tuning parameter to improve the performance at 0. Can be either a number or a vector of the size `length(grid)`. If this is a vector, the code will need to allocate a matrix of size `nrow(X) * length(grid)` which can be prohibitive in some cases.
`mpfr`	if `mpfr = TRUE`, multiple precision floating point is used via the package Rmpfr. This allows for a higher (numerical) accuracy, at the expense of computing time. It is recommended to use this option for higher dimensions.

precBits	number of precBits used for floating point precision (only used if mpfr = TRUE).
dopb	a Boolean value. If dopb = TRUE, a progress bar is displayed.

Value

a vector of the same size as the grid, with the corresponding value for the $\widetilde{AMSE}$.

Author(s)

Alexis Derumigny, Victor Ryan

References

Ryan, V., & Derumigny, A. (2024). On the choice of the two tuning parameters for nonparametric estimation of an elliptical distribution generator arxiv:2408.17087.

Examples

```
# Comparison between the estimated and true generator of the Gaussian distribution
n = 50000
d = 3
X = matrix(rnorm(n * d), ncol = d)
grid = seq(0, 5, by = 0.1)
a = 1.5

AMSE_est = estim_tilde_AMSE(X = X, grid = grid, a = a, h = 0.09)
plot(grid, abs(AMSE_est), type = "l")

# Computation of true values
g = exp(-grid/2)/(2*pi)^{3/2}
gprime = (-1/2) *exp(-grid/2)/(2*pi)^{3/2}
A = a^(d/2)
psia = -a + (A + grid^(d/2))^(2/d)
psiaprime = grid^(d/2 - 1) * (A + grid^(d/2))^(2/d - 1)
psiasecond = psiaprime * ( (d-2)/2 ) * grid^{-1} * A *
  ( grid^(d/2) + A )^(-1)

rhoprimexi = ((d-2) * grid^((d-4)/2) * psiaprime
  - 2 * grid^((d-2)/2) * psiasecond) / (2 * psiaprime^3) * g +
  grid^((d-2)/2) / (psiaprime^2) * gprime

AMSE = rhoprimexi / psiaprime

lines(grid, abs(AMSE), col = "red")


# Comparison as a function of $a$
n = 50000
d = 3
X = matrix(rnorm(n * d), ncol = d)
grid = 0.1
vec_a = c(0.001, 0.002, 0.005,
```

```
0.01, 0.02, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.8, 1, 1.5, 2)

AMSE_est = rep(NA, length = length(vec_a))
for (i in 1:length(vec_a)){
  AMSE_est[i] = estim_tilde_AMSE(X = X, grid = grid, a = vec_a[i], h = 0.09,
                           dopb = FALSE)
}

plot(vec_a, abs(AMSE_est), type = "l", log = "x")

# Computation of true values
a = vec_a

g = exp(-grid/2)/(2*pi)^{3/2}
gprime = (-1/2) *exp(-grid/2)/(2*pi)^{3/2}
A = a^(d/2)
psia = -a + (A + grid^(d/2))^(2/d)
psiaprime = grid^(d/2 - 1) * (A + grid^(d/2))^(2/d - 1)
psiasecond = psiaprime * ( (d-2)/2 ) * grid^{-1} * A *
  ( grid^(d/2) + A )^(-1)

rhoprimexi = ((d-2) * grid^((d-4)/2) * psiaprime
  - 2 * grid^((d-2)/2) * psiasecond) / (2 * psiaprime^3) * g +
  grid^((d-2)/2) / (psiaprime^2) * gprime

AMSE = rhoprimexi / psiaprime

yliminf = min(c(abs(AMSE_est), abs(AMSE)))
ylimsup = max(c(abs(AMSE_est), abs(AMSE)))

plot(vec_a, abs(AMSE_est), type = "l", log = "xy",
     ylim = c(yliminf, ylimsup))
lines(vec_a, abs(AMSE), col = "red")
```

`KTMatrixEst` *Fast estimation of Kendall's tau matrix*

Description

Estimate Kendall's tau matrix using averaging estimators. Under the structural assumption that Kendall's tau matrix is block-structured with constant values in each off-diagonal block, this function estimates Kendall's tau matrix "fast", in the sense that each interblock coefficient is estimated in time $N \cdot n \cdot log(n)$, where `N` is the amount of pairs that are averaged.

Usage

```
KTMatrixEst(dataMatrix, blockStructure = NULL, averaging = "no", N = NULL)
```

Arguments

- `dataMatrix` — matrix of size (n,d) containing n observations of a d-dimensional random vector.
- `blockStructure` — list of vectors. Each vector corresponds to one group of variables and contains the indexes of the variables that belongs to this group. `blockStructure` must be a partition of `1:d`, where `d` is the number of columns in `dataMatrix`.
- `averaging` — type of averaging used for fast estimation. Possible choices are
  - `no`: no averaging;
  - `all`: averaging all Kendall's taus in each block. `N` is then the number of entries in the block, i.e. the products of both dimensions.
  - `diag`: averaging along diagonal blocks elements. `N` is then the minimum of the block's dimensions.
  - `row`: averaging Kendall's tau along the smallest block side. `N` is then the minimum of the block's dimensions.
  - `random`: averaging Kendall's taus along a random sample of `N` entries of the given block. These entries are chosen uniformly without replacement.
- `N` — number of entries to average (n the random case. By default, `N` is then the minimum of the block's dimensions.

Value

matrix with dimensions depending on `averaging`.

- If `averaging = no`, the function returns a matrix of dimension `(n,n)` which estimates the Kendall's tau matrix.
- Else, the function returns a matrix of dimension `(length(blockStructure) , length(blockStructure))` giving the estimates of the Kendall's tau for each block with ones on the diagonal.

Author(s)

Rutger van der Spek, Alexis Derumigny

References

van der Spek, R., & Derumigny, A. (2022). Fast estimation of Kendall's Tau and conditional Kendall's Tau matrices under structural assumptions. arxiv:2204.03285.

Examples

```
# Estimating off-diagonal block Kendall's taus
matrixCor = matrix(c(1  , 0.5, 0.3 ,0.3, 0.3,
                     0.5,   1, 0.3, 0.3, 0.3,
                     0.3, 0.3,   1, 0.5, 0.5,
                     0.3, 0.3, 0.5,   1, 0.5,
                     0.3, 0.3, 0.5, 0.5,   1), ncol = 5 , nrow = 5)
dataMatrix = mvtnorm::rmvnorm(n = 100, mean = rep(0, times = 5), sigma = matrixCor)
blockStructure = list(1:2, 3:5)
estKTMatrix = list()
```

```
estKTMatrix$all = KTMatrixEst(dataMatrix = dataMatrix,
                              blockStructure = blockStructure,
                              averaging = "all")
estKTMatrix$row = KTMatrixEst(dataMatrix = dataMatrix,
                              blockStructure = blockStructure,
                              averaging = "row")
estKTMatrix$diag = KTMatrixEst(dataMatrix = dataMatrix,
                               blockStructure = blockStructure,
                               averaging = "diag")
estKTMatrix$random = KTMatrixEst(dataMatrix = dataMatrix,
                                 blockStructure = blockStructure,
                                 averaging = "random", N = 2)
InterBlockCor = lapply(estKTMatrix, FUN = function(x) {sin(x[1,2] * pi / 2)})

# Estimation of the correlation between variables of the first group
# and of the second group
print(unlist(InterBlockCor))
# to be compared with the true value: 0.3.
```

`TEllDistrEst` *Estimation of trans-elliptical distributions*

Description

This function estimates the parameters of a trans-elliptical distribution which is a distribution whose copula is (meta-)elliptical, with arbitrary margins, using the procedure proposed in (Derumigny & Fermanian, 2022).

Usage

```
TEllDistrEst(
  X, estimatorCDF = function(x){
    force(x)
    return( function(y){(stats::ecdf(x)(y) - 1/(2*length(x))) }) },
  h, verbose = 1, grid, ...)
```

Arguments

`X`	the matrix of observations of the variables
`estimatorCDF`	the way of estimating the marginal cumulative distribution functions. It should be either a function that takes in parameter a vector of observations and returns an estimated cdf (i.e. a function) or a list of such functions to be applied on the data. In this case, it is required that the length of the list should be the same as the number of columns of `X`. It is required that the functions returned by `estimatorCDF` should have values in the *open* interval $(0, 1)$.
`h`	bandwidth for the non-parametric estimation of the density generator.

verbose	if 1, prints the progress of the iterations. If 2, prints the normalizations constants used at each iteration, as computed by DensityGenerator.normalize.
grid	grid of values on which to estimate the density generator
...	other parameters to be passed to EllCopEst.

Value

This function returns a list with three components:

- listEstCDF: a list of estimated marginal CDF given by estimatorCDF;
- corMatrix: the estimated correlation matrix:
- estEllCopGen: the estimated generator of the meta-elliptical copula.

References

Derumigny, A., & Fermanian, J. D. (2022). Identifiability and estimation of meta-elliptical copula generators. Journal of Multivariate Analysis, article 104962. doi:10.1016/j.jmva.2022.104962.

Examples

```
cor = matrix(c(1, 0.5, 0.2,
               0.5, 1, 0.8,
               0.2, 0.8, 1), byrow = TRUE, nrow = 3)

grid = seq(0,10,by = 0.01)
g_d = DensityGenerator.normalize(grid, grid_g = exp(-grid), d = 3)
n = 10
# To have a nice estimation, we suggest to use rather n=200
# (around 20s of computation time)
U = EllCopSim(n = n, d = 3, grid = grid, g_d = g_d, A = chol(cor))
X = matrix(nrow = n, ncol = 3)
X[,1] = stats::qnorm(U[,1], mean = 2)
X[,2] = stats::qt(U[,2], df = 5)
X[,3] = stats::qt(U[,3], df = 8)

result = TEllDistrEst(X, h = 0.1, grid = grid)
plot(grid, g_d, type = "l", xlim = c(0,2))
lines(grid, result$estiEllCop$g_d_norm, col = "red")
print(result$corMatrix)

# Adding missing observations
n_NA = 2
X_NA = X
for (i in 1:n_NA){
  X_NA[sample.int(n,1), sample.int(3,1)] = NA
}
resultNA = TEllDistrEst(X_NA, h = 0.1, grid = grid, verbose = 1)
lines(grid, resultNA$estiEllCopGen, col = "blue")
```

---

vectorized_Faa_di_Bruno
Vectorized version of Faa di Bruno formula

Description

This code implements a vectorized version of the Faa di Bruno formula, relying internally on the Bell polynomials from the package kStatistics, via the function kStatistics::eBellPol.

Usage

```
vectorized_Faa_di_Bruno(f, g, x, k, args_f, args_g)
```

Arguments

f, g	two functions that take in argument • a vector x of numeric values • an integer k which is as to be understood as the order of the derivative of f • potentially other parameters (not vectorized)
x	vector of (one-dimensional) values at which the k-th order derivatives is to be evaluated.
k	the order of the derivative
args_f, args_g	the list of additional parameters to be passed on to f and g. This must be the same for all values of x.

Value

a vector of size `length(x)` for which the `i`-th component is $(f \circ g)^{(k)}(x[i])$

Author(s)

Alexis Derumigny, Victor Ryan

See Also

`compute_matrix_alpha` which also uses the Bell polynomials in a similar way.

Examples

```
g <- function(x, k, a){
  if (k == 0){ return ( exp(x) + a)
  } else {
    return (exp(x))
  }
}
args_g = list(a = 2)
```

```
f <- function(x, k, a){
  if (k == 0){ return ( x^2 + a)
  } else if (k == 1) {
    return ( 2 * x)
  } else if (k == 2) {
    return ( 2 )
  } else {
    return ( 0 )
  }
}
args_f = list(a = 5)

x = 1:5
vectorized_Faa_di_Bruno(f = f, g = g, x = x, k = 1,
  args_f = args_f, args_g = args_g)
# Derivative of ( exp(x) + 2 )^2 + 5
# which explicit expression is:
2 * exp(x) * ( exp(x) + 2 )
```

Index