Package ‘mvpd’

September 3, 2023

Title Multivariate Product Distributions for Elliptically Contoured Distributions

Version 0.0.4

Description Estimates multivariate subgaussian stable densities and probabilities as well as generates random variates using product distribution theory. A function for estimating the parameters from data to fit a distribution to data is also provided, using the method from Nolan (2013) <doi:10.1007/s00180-013-0396-7>.

Imports matrixStats, stabdist, libstable4u, mvtarm, stats, cubature, Matrix

Depends R (>= 3.4.0)

License MIT + file LICENSE

RoxygenNote 7.2.1

Suggests rmarkdown, knitr, testthat (>= 3.0.0)

Config/testthat/edition 3

URL https://github.com/swihart/mvpd

BugReports https://github.com/swihart/mvpd/issues

NeedsCompilation no

Author Bruce Swihart [aut, cre] (<https://orcid.org/0000-0002-4216-9942>)

Maintainer Bruce Swihart <bruce.swihart@gmail.com>

Repository CRAN

Date/Publication 2023-09-03 14:20:02 UTC

R topics documented:

adaptIntegrate_inf_limPD .................................................. 2
dmvss ................................................................. 3
fit_mvss ................................................................. 5
mvpd ................................................................. 7
Adaptive multivariate integration over hypercubes (admitting infinite limits)

Description

The function performs adaptive multidimensional integration (cubature) of (possibly) vector-valued integrands over hypercubes. It is a wrapper for cubature:::adaptIntegrate, transforming (-)Inf appropriately as described in cubature’s help page (http://ab-initio.mit.edu/wiki/index.php/Cubature#Infinite_intervals).

Usage

adaptIntegrate_inf_limPD(
  f,
  lowerLimit,
  upperLimit,
  ..., 
  tol.ai = 1e-05, 
  fDim.ai = 1, 
  maxEval.ai = 0, 
  absError.ai = 0, 
  doChecking.ai = FALSE
)

Arguments

f The function (integrand) to be integrated
lowerLimit The lower limit of integration, a vector for hypercubes
upperLimit The upper limit of integration, a vector for hypercubes
... All other arguments passed to the function f
tol.ai The maximum tolerance, default 1e-5.
fDim.ai The dimension of the integrand, default 1, bears no relation to the dimension of the hypercube
maxEval.ai The maximum number of function evaluations needed, default 0 implying no limit
absError.ai The maximum absolute error tolerated
doChecking.ai A flag to be thorough checking inputs to C routines. A FALSE value results in approximately 9 percent speed gain in our experiments. Your mileage will of course vary. Default value is FALSE.
Examples

```r
## integrate Cauchy Density from -Inf to Inf
adaptIntegrate_inf_limPD(function(x) 1/pi * 1/(1+x^2), -Inf, Inf)
adaptIntegrate_inf_limitPD(function(x, scale) 1/(pi*scale) * 1/(1+(x/scale)^2), -Inf, Inf, scale=4)
## integrate Cauchy Density from -Inf to -3
adaptIntegrate_inf_limitPD(function(x) 1/pi * 1/(1+x^2), -Inf, -3)
stats::pcauchy(-3)
adaptIntegrate_inf_limitPD(function(x, scale) 1/(pi*scale) * 1/(1+(x/scale)^2), -Inf, -3, scale=4)
stats::pcauchy(-3, scale=4)
```

**dmvss**

*Multivariate Subgaussian Stable Density*

**Description**

Computes the density function of the multivariate subgaussian stable distribution for arbitrary alpha, shape matrices, and location vectors. See Nolan (2013).

**Usage**

```r
dmvss(
  x,
  alpha = 1,
  Q = NULL,
  delta = rep(0, d),
  outermost.int = c("stats::integrate", "cubature::adaptIntegrate")[1],
  spherical = FALSE,
  subdivisions.si = 100L,
  rel.tol.si = .Machine$double.eps^0.25,
  abs.tol.si = rel.tol.si,
  stop.on.error.si = TRUE,
  tol.ai = 1e-05,
  fDim.ai = 1,
  maxEval.ai = 0,
  absError.ai = 0,
  doChecking.ai = FALSE,
  which.stable = c("libstable4u", "stabledist")[1]
)
```

**Arguments**

- `x` vector of quantiles.
- `alpha` default to 1 (Cauchy). Must be 0<alpha<2
- `Q` Shape matrix. See Nolan (2013).
- `delta` location vector
outermost.int  select which integration function to use for outermost integral. Default is "stats::integrate" and one can specify the following options with the .si suffix. For diagonal Q, one can also specify "cubature::adaptIntegrate" and use the .ai suffix options below (currently there is a bug for non-diagonal Q).

spherical  default is FALSE. If true, use the spherical transformation. Results will be identical to spherical = FALSE but may be faster.

subdivisions.si  the maximum number of subintervals. The suffix .si indicates a stats::integrate() option for the outermost semi-infinite integral in the product distribution formulation.

rel.tol.si  relative accuracy requested. The suffix .si indicates a stats::integrate() option for the outermost semi-infinite integral in the product distribution formulation.

abs.tol.si  absolute accuracy requested. The suffix .si indicates a stats::integrate() option for the outermost semi-infinite integral in the product distribution formulation.

stop.on.error.si  logical. If true (the default) an error stops the function. If false some errors will give a result with a warning in the message component. The suffix .si indicates a stats::integrate() option for the outermost semi-infinite integral in the product distribution formulation.

tol.ai  The maximum tolerance, default 1e-5. The suffix .ai indicates a cubature::adaptIntegrate type option for the outermost semi-infinite integral in the product distribution formulation.

fDim.ai  The dimension of the integrand, default 1, bears no relation to the dimension of the hypercube The suffix .ai indicates a cubature::adaptIntegrate type option for the outermost semi-infinite integral in the product distribution formulation.

maxEval.ai  The maximum number of function evaluations needed, default 0 implying no limit The suffix .ai indicates a cubature::adaptIntegrate type option for the outermost semi-infinite integral in the product distribution formulation.

absError.ai  The maximum absolute error tolerated The suffix .ai indicates a cubature::adaptIntegrate type option for the outermost semi-infinite integral in the product distribution formulation.

doChecking.ai  A flag to be thorough checking inputs to C routines. A FALSE value results in approximately 9 percent speed gain in our experiments. Your mileage will of course vary. Default value is FALSE. The suffix .ai indicates a cubature::adaptIntegrate type option for the outermost semi-infinite integral in the product distribution formulation.

which.stable  defaults to "libstable4u", other option is "stabledist". Indicates which package should provide the univariate stable distribution in this production distribution form of a univariate stable and multivariate normal.

Value

The object returned depends on what is selected for outermost.int. In the case of the default, stats::integrate, the value is a list of class "integrate" with components:
• value the final estimate of the integral.
• abs.error estimate of the modulus of the absolute error.
• subdivisions the number of subintervals produced in the subdivision process.
• message "OK" or a character string giving the error message.
• call the matched call.

Note: The reported abs.error is likely an under-estimate as integrate assumes the integrand was without error, which is not the case in this application.

References


Examples

## print("mvsbgaussPD (d=2, alpha=1.71):")
Q <- matrix(c(10,7.5,7.5,10),2)
mvpd::dmvss(x=c(0,1), alpha=1.71, Q=Q)

## more accuracy = longer runtime
mvpd::dmvss(x=c(0,1),alpha=1.71, Q=Q, abs.tol=1e-8)

Q <- matrix(c(10,7.5,7.5,7.5,10,7.5,7.5,7.5,10),3)
## print("mvsbgaussPD (d=3, alpha=1.71):")
mvpd::dmvss(x=c(0,1,2), alpha=1.71, Q=Q)
mvpd::dmvss(x=c(0,1,2), alpha=1.71, Q=Q, spherical=TRUE)

## How 'delta' works: same as centering
X <- c(1,1,1)
Q <- matrix(c(10,7.5,7.5,7.5,10,7.5,7.5,7.5,10),3)
D <- c(0.75, 0.65, -0.35)
mvpd::dmvss(X-D, alpha=1.71, Q=Q)
mvpd::dmvss(X , alpha=1.71, Q=Q, delta=D)

---

**fit_mvss**

*Fit a Multivariate Subgaussian Distribution*

**Description**

Estimates the parameters (namely, alpha, shape matrix Q, and location vector) of the multivariate subgaussian distribution for an input matrix X.

**Usage**

`fit_mvss(x)`
Arguments

x a matrix for which the parameters for a d-dimensional multivariate subgaussian distribution will be estimated. The number of columns will be d.

Details

Using the protocols outlined in Nolan (2013), this function uses libstable4u's univariate fit functions for each component.

Value

A list with parameters from the column-wise univariate fits and the multivariate alpha and shape matrix estimates (the univ_deltas are the mult_deltas):

- univ_alphas - the alphas from the column-wise univariate fits
- univ_betas - the betas from the column-wise univariate fits
- univ_gammas - the gammas from the column-wise univariate fits
- univ_deltas - the deltas from the column-wise univariate fits
- mult_alpha - the mean(univ_alphas); equivalently the multivariate alpha estimate
- mult_Q_raw - the multivariate shape matrix estimate (before applying nearPD())
- mult_Q_posdef - the nearest positive definite multivariate shape matrix estimate, nearPD(mult_Q_raw)

References


See Also

Rfast::mvnorm.mle, alphastable::mfitstab.elliptical

Examples

```r
## create a 4x4 shape matrix symMat
S <- matrix(rnorm(4*4, mean=2, sd=4),4);
symMat <- as.matrix(Matrix::nearPD(0.5 * (S + t(S)))$mat)
symMat
## generate 10,000 r.v.'s from 4-dimensional mvss
X <- mvpd::rmvss(1e4, alpha=1.5, Q=symMat, delta=c(1,2,3,4))
## use fit_mvss to recover the parameters, compare to symMat
fmv <- mvpd::fit_mvss(X)
fmv

## then use the fitted parameters to calculate a probability:
mvpd::pmvss(lower=rep(0,4),
        upper=rep(5,4),
        alpha=fmv$mult_alpha,
        Q=fmv$mult_Q_posdef,
```
Description

The purpose of this package is to offer density, probability, and random variate generating (abbreviated as [d/p/r], respectively) functions for multivariate distributions that can be represented as a product distribution. Specifically, the package will primarily focus on the product of a multivariate normal distribution and a univariate random variable. These product distributions are called Scale Mixtures of Multivariate Normal Distributions, and for particular choices of the univariate random variable distribution the resultant product distribution may be a family of interest. For instance, the square-root of a positive stable random variable multiplied by a multivariate normal distribution is the multivariate subgaussian stable distribution. Product distribution theory is applied for implementing their computation.

Multivariate subgaussian stable distributions

- **dmvss** – multivariate subgaussian stable distribution density
- **pmvss** – multivariate subgaussian stable distribution probabilities
- **rmvss** – multivariate subgaussian stable distribution random variates
- **pmvss.mc** – Monte Carlo version of probabilities, using rmvss
- **fit_mvss** – Fit a multivariate subgaussian stable distribution (e.g. estimate parameters given data)

Author(s)

**Maintainer**: Bruce Swihart <bruce.swihart@gmail.com> (ORCID)

See Also

Useful links:

- [https://github.com/swihart/mvpd](https://github.com/swihart/mvpd)
**pmvss**  
*Multivariate Subgaussian Stable Distribution*

**Description**
Computes the probabilities for the multivariate subgaussian stable distribution for arbitrary limits, alpha, shape matrices, and location vectors. See Nolan (2013).

**Usage**

```r
pmvss(
  lower = rep(-Inf, d),
  upper = rep(Inf, d),
  alpha = 1,
  Q = NULL,
  delta = rep(0, d),
  maxpts.pmvnorm = 25000,
  abseps.pmvnorm = 0.001,
  outermost.int = c("stats::integrate", "cubature::adaptIntegrate")[1],
  subdivisions.si = 100L,
  rel.tol.si = .Machine$double.eps*0.25,
  abs.tol.si = rel.tol.si,
  stop.on.error.si = TRUE,
  tol.ai = 1e-05,
  fDim.ai = 1,
  maxEval.ai = 0,
  absError.ai = 0,
  doChecking.ai = FALSE,
  which.stable = c("libstable4u", "stabledist")[1]
)
```

**Arguments**
- `lower` the vector of lower limits of length n.
- `upper` the vector of upper limits of length n.
- `alpha` default to 1 (Cauchy). Must be 0<alpha<2
- `Q` Shape matrix. See Nolan (2013).
- `delta` location vector.
- `maxpts.pmvnorm` Defaults to 25000. Passed to the F_G = pmvnorm() in the integrand of the outermost integral.
- `abseps.pmvnorm` Defaults to 1e-3. Passed to the F_G = pmvnorm() in the integrand of the outermost integral.
- `outermost.int` select which integration function to use for outermost integral. Default is "stats::integrate" and one can specify the following options with the .si suffix. For diagonal Q, one can also specify "cubature::adaptIntegrate" and use the .ai suffix options below (currently there is a bug for non-diagonal Q).
subdivisions.si
the maximum number of subintervals. The suffix .si indicates a stats::integrate() option for the outermost semi-infinite integral in the product distribution formulation.

rel.tol.si
relative accuracy requested. The suffix .si indicates a stats::integrate() option for the outermost semi-infinite integral in the product distribution formulation.

abs.tol.si
absolute accuracy requested. The suffix .si indicates a stats::integrate() option for the outermost semi-infinite integral in the product distribution formulation.

stop.on.error.si
logical. If true (the default) an error stops the function. If false some errors will give a result with a warning in the message component. The suffix .si indicates a stats::integrate() option for the outermost semi-infinite integral in the product distribution formulation.

tol.ai
The maximum tolerance, default 1e-5. The suffix .ai indicates a cubature::adaptIntegrate type option for the outermost semi-infinite integral in the product distribution formulation.

fDim.ai
The dimension of the integrand, default 1, bears no relation to the dimension of the hypercube. The suffix .ai indicates a cubature::adaptIntegrate type option for the outermost semi-infinite integral in the product distribution formulation.

maxEval.ai
The maximum number of function evaluations needed, default 0 implying no limit. The suffix .ai indicates a cubature::adaptIntegrate type option for the outermost semi-infinite integral in the product distribution formulation.

absError.ai
The maximum absolute error tolerated. The suffix .ai indicates a cubature::adaptIntegrate type option for the outermost semi-infinite integral in the product distribution formulation.

doChecking.ai
A flag to be thorough checking inputs to C routines. A FALSE value results in approximately 9 percent speed gain in our experiments. Your mileage will of course vary. Default value is FALSE. The suffix .ai indicates a cubature::adaptIntegrate type option for the outermost semi-infinite integral in the product distribution formulation.

which.stable
defaults to "libstable4u", other option is "stabledist". Indicates which package should provide the univariate stable distribution in this production distribution form of a univariate stable and multivariate normal.

Value
The object returned depends on what is selected for outermost.int. In the case of the default, stats::integrate, the value is a list of class "integrate" with components:

• value the final estimate of the integral.
• abs.error estimate of the modulus of the absolute error.
• subdivisions the number of subintervals produced in the subdivision process.
• message "OK" or a character string giving the error message.
• call the matched call.

Note: The reported abs.error is likely an under-estimate as integrate assumes the integrand was without error, which is not the case in this application.

References


Examples

```r
## bivariate
U <- c(1,1)
L <- -U
Q <- matrix(c(10,7.5,7.5,10),2)
mvpd::pmvss(L, U, alpha=1.71, Q=Q)

## trivariate
U <- c(1,1,1)
L <- -U
Q <- matrix(c(10,7.5,7.5,7.5,10,7.5,7.5,7.5,10),3)
mvpd::pmvss(L, U, alpha=1.71, Q=Q)

## How 'delta' works: same as centering
U <- c(1,1,1)
L <- -U
Q <- matrix(c(10,7.5,7.5,7.5,10,7.5,7.5,7.5,10),3)
D <- c(0.75, 0.65, -0.35)
mvpd::pmvss(L-D, U-D, alpha=1.71, Q=Q)
mvpd::pmvss(L , U , alpha=1.71, Q=Q, delta=D)
```

---

**pmvss_mc**

*Monte Carlo Multivariate Subgaussian Stable Distribution*

**Description**

Computes probabilities of the multivariate subgaussian stable distribution for arbitrary limits, alpha, shape matrices, and location vectors via Monte Carlo (thus the suffix _mc_).

**Usage**

```r
pmvss_mc(
    lower = rep(-Inf, d),
    upper = rep(Inf, d),
    alpha = 1,
    ```
pmvss_mc

    Q = NULL,
    delta = rep(0, d),
    which.stable = c("libstable4u", "stabledist")[1],
    n = NULL
  )

Arguments

lower             the vector of lower limits of length n.
upper             the vector of upper limits of length n.
alpha             default to 1 (Cauchy). Must be 0<alpha<2
Q                  Shape matrix. See Nolan (2013).
delta             location vector.
which.stable       defaults to "libstable4u", other option is "stabledist". Indicates which package
                   should provide the univariate stable distribution in this production distribution
                   form of a univariate stable and multivariate normal.
n             number of random vectors to be drawn for Monte Carlo calculation.

Value

    a number between 0 and 1, the estimated probability via Monte Carlo

References


Examples

    ## print("mvpd (d=2, alpha=1.71):")
    U <- c(1,1)
    L <- -U
    Q <- matrix(c(10,7.5,7.5,10),2)
    mvpd::pmvss_mc(L, U, alpha=1.71, Q=Q, n=1e3)
    mvpd::pmvss (L, U, alpha=1.71, Q=Q)

    ## more accuracy = longer runtime
    mvpd::pmvss_mc(L, U, alpha=1.71, Q=Q, n=1e4)

    U <- c(1,1,1)
    L <- -U
    Q <- matrix(c(10,7.5,7.5,7.5,10,7.5,7.5,7.5,10),3)
    ## print("mvpd: (d=3, alpha=1.71)")
    mvpd::pmvss_mc(L, U, alpha=1.71, Q=Q, n=1e3)
Multivariate Subgaussian Stable Random Variates

Description


Usage

```
rmvss(
  n,
  alpha = 1,
  Q = NULL,
  delta = rep(0, d),
  which.stable = c("libstable4u", "stabledist")[1]
)
```

Arguments

- `n`: number of observations
- `alpha`: default to 1 (Cauchy). Must be 0<alpha<2
- `delta`: location vector.
- `which.stable`: defaults to "libstable4u", other option is "stabledist". Indicates which package should provide the univariate stable distribution in this production distribution form of a univariate stable and multivariate normal.

Value

Returns the n by d matrix containing multivariate subgaussian stable random variates where d=nrow(Q).

References


Examples

```
## generate 10 random variates of a bivariate mvss
rmvss(n=10, alpha=1.71, Q=matrix(c(10,7.5,7.5,10),2))

## generate 10 random variates of a trivariate mvss
Q <- matrix(c(10,7.5,7.5,7.5,10,7.5,7.5,7.5,10),3)
rmvss(n=10, alpha=1.71, Q=Q)
```
Index

* distribution
  dmvss, 3
  fit_mvss, 5
  pmvss, 8
  rmvss, 12

* integration
  adaptIntegrate_inf_limPD, 2

* multivariate
  adaptIntegrate_inf_limPD, 2

* numerical
  adaptIntegrate_inf_limPD, 2

adaptIntegrate_inf_limPD, 2

dmvss, 3, 7

fit_mvss, 5, 7

mvpd, 7
mvpd-package (mvpd), 7

pmvss, 7, 8
pmvss_mc, 7, 10

rmvss, 7, 12