Package: LambertW (via r-universe)

October 27, 2024

Type Package

Title Probabilistic Models to Analyze and Gaussianize Heavy-Tailed, Skewed Data

Version 0.6.9-1

URL `https://github.com/gmgeorg/LambertW` `https://arxiv.org/abs/0912.4554` `https://arxiv.org/abs/1010.2265` `https://arxiv.org/abs/1602.02200`

BugReports `https://github.com/gmgeorg/LambertW/issues`

Description Lambert W x F distributions are a generalized framework to analyze skewed, heavy-tailed data. It is based on an input/output system, where the output random variable (RV) Y is a non-linearly transformed version of an input RV X ~ F with similar properties as X, but slightly skewed (heavy-tailed). The transformed RV Y has a Lambert W x F distribution. This package contains functions to model and analyze skewed, heavy-tailed data the Lambert Way: simulate random samples, estimate parameters, compute quantiles, and plot/ print results nicely. The most useful function is 'Gaussianize', which works similarly to 'scale', but actually makes the data Gaussian. A do-it-yourself toolkit allows users to define their own Lambert W x 'MyFavoriteDistribution' and use it in their analysis right away.

Depends MASS, ggplot2,

Imports lamW (>= 1.3.0), stats, graphics, grDevices, RColorBrewer, reshape2, Rcpp (>= 1.0.4), methods

Suggests boot, Rsolnp, nortest, numDeriv, testthat, data.table, moments, knitr, markdown, vars,

License GPL (>= 2)

LazyLoad yes

NeedsCompilation yes

LinkingTo Rcpp, lamW

RoxygenNote 7.2.3

Encoding UTF-8

VignetteBuilder knitr

Repository https://gmgeorg.r-universe.dev

RemoteUrl https://github.com/gmgeorg/lambertw

RemoteRef HEAD

RemoteSha 592c342ad1be39e3113744f1b8908a8bf872ffd7

Contents

`LambertW-package` *R package for Lambert W× F distributions*

Description

This package is based on notation, definitions, and results of Goerg (2011, 2015, 2016). I will not include these references in the description of each single function.

Lambert W× F distributions are a general framework to model and transform skewed, heavy-tailed data. Lambert W× F random variables (RV) are based on an input/ouput system with input RV $X \sim F_X(x \mid \boldsymbol{\beta})$ and output Y, which is a non-linearly transformed version of X – with similar properties to X, but slightly skewed and/or heavy-tailed. Then Y has a 'Lambert W $\times F_X$' distribution - see References.

`get_distnames` lists all implemented Lambert W × F distributions in this package. If you want to generate a skewed/heavy-tailed version of a distribution that is not implemented, you can use the do-it-yourself modular toolkit (`create_LambertW_input` and `create_LambertW_output`). It allows users to quickly implement their own Lambert W x 'MyFavoriteDistribution' and use it in their analysis right away.

This package contains several functions to analyze skewed and heavy-tailed data: simulate random samples (`rLambertW`), evaluate pdf and cdf (`dLambertW` and `pLambertW`), estimate parameters (`IGMM` and `MLE_LambertW`), compute quantiles (`qLambertW`), and plot/print results nicely (`plot.LambertW_fit`, `print.LambertW_fit`, `summary.LambertW_fit`).

Probably the most useful function is `Gaussianize`, which works similarly to `scale`, but makes your data Gaussian (not just centers and scales it, but also makes it symmetric and removes excess kurtosis).

If you use this package in your work please cite it (`citation("LambertW")`). You can also send me an implementation of your 'Lambert W × YourFavoriteDistribution' to add to the **LambertW** package (and I will reference your work introducing your 'Lambert W × YourFavoriteDistribution' here.)

Feel free to contact me for comments, suggestions, code improvements, implementation of new input distributions, bug reports, etc.

Author(s)

Author and maintainer: Georg M. Goerg (im (at) gmge.org)

References

Goerg, G.M. (2011). “Lambert W Random Variables - A New Family of Generalized Skewed Distributions with Applications to Risk Estimation”. Annals of Applied Statistics, 5 (3), 2197-2230. (https://arxiv.org/abs/0912.4554).

Goerg, G.M. (2015). “The Lambert Way to Gaussianize heavy-tailed data with the inverse of Tukey’s h transformation as a special case”. The Scientific World Journal: Probability and Statistics with Applications in Finance and Economics. Available at https://www.hindawi.com/journals/tswj/2015/909231/.

Goerg, G.M. (2016). “Rebuttal of the “Letter to the Editor of Annals of Applied Statistics” on Lambert W x F distributions and the IGMM algorithm”. Available on arxiv.

Examples

```
## Not run:
# Replicate parts of the analysis in Goerg (2011)
data(AA)
y <- AA[AA$sex=="f", "bmi"]
test_normality(y)

fit.gmm <- IGMM(y, type = "s")
summary(fit.gmm)  # gamma is significant and positive
plot(fit.gmm)

# Compare empirical to theoretical moments (given parameter estimates)
moments.theory <-
 mLambertW(theta = list(beta = fit.gmm$tau[c("mu_x", "sigma_x")],
                        gamma = fit.gmm$tau["gamma"]),
           distname = "normal")
TAB <- rbind(unlist(moments.theory),
             c(mean(y), sd(y), skewness(y), kurtosis(y)))
rownames(TAB) <- c("Theoretical (IGMM)", "Empirical")
TAB

x <- get_input(y, fit.gmm$tau)
test_normality(x) # input is normal -> fit a Lambert W x Gaussian by MLE

fit.ml <- MLE_LambertW(y, type = "s", distname = "normal", hessian = TRUE)
summary(fit.ml)
plot(fit.ml)

## End(Not run)
```

`analyze_convergence` *Analyze convergence of Lambert W estimators*

Description

Analyzes the feasibility of a Lambert W x F distribution for a given dataset based on bootstrapping. In particular it checks whether parameter estimates support the hypothesis that the data indeed follows a Lambert W x F distribution with finite mean and variance of the input distribution, which is an implicit assumption of Lambert W x F random variables in Goerg (2011).

See Goerg (2016) for an alternative definition that does not rely on fnite second order moments (set `use.mean.variance = FALSE` to use that type of Lambert W × F distributions).

Usage

```
analyze_convergence(
  LambertW_fit,
  sample.sizes = round(seq(0.2, 1, length = 5) * length(LambertW_fit$data)),
  ...
)

## S3 method for class 'convergence_LambertW_fit'
summary(object, type = c("basic", "norm", "perc", "bca"), ...)

## S3 method for class 'convergence_LambertW_fit'
plot(x, ...)
```

Arguments

`LambertW_fit`, `object`, `x`	an object of class `"LambertW_fit"` with an `IGMM` or `MLE_LambertW` estimate.
`sample.sizes`	sample sizes for several steps of the convergence analysis. By default, one of them equals the length of the original data, which leads to improved plots (see `plot.convergence_LambertW_fit`); it is not necessary, though.
`...`	additional arguments passed to `bootstrap` or `boot.ci` in **boot** package.
`type`	type of confidence interval from bootstrap estimates. Passes this argument along to `boot.ci`. However, contrary to the `type` argument in `boot.ci`, the `summary` function can only take one of `c("basic", "norm", "perc", "bca")`. See `boot.ci` for details.

Details

Stehlik and Hermann (2015) show that when researchers use the IGMM algorithm outlined in Goerg (2011) erroneously on data that does not have finite input variance (and hence mean), the algorithm estimates do not converge.

In practice, researchers should of course first check if a given model is appropriate for their data-generating process. Since original Lambert W x F distributions assume that mean and variance are finite, it is not a given that for a specific dataset the Lambert W x F setting makes sense.

The bootstrap analysis reverses Stehlik and Hermann's argument and checks whether the IGMM estimates $\{\hat{\tau}^{(n)}\}_n$ converge for increasing (bootstrapped) sample size n: if they do, then modeling the data with a Lambert W x F distribution is appropriate; if estimates do not converge, then this indicates that the input data is too heavy tailed for a classic skewed location-scale Lambert W x F framework. In this case, take a look at (double-)heavy tailed Lambert W x F distributions (`type = 'hh'`) or unrestricted location-scale Lambert W x F distributions (`use.mean.variance = FALSE`). For details see Goerg (2016).

References

Stehlik and Hermann (2015). “Letter to the Editor”. Ann. Appl. Stat. 9 2051. doi:10.1214/15-AOAS864 – https://projecteuclid.org/euclid.aoas/1453994190

Examples

```
## Not run:

sim.data <- list("Lambert W x Gaussian" =
                   rLambertW(n = 100, distname = "normal",
                             theta = list(gamma = 0.1, beta = c(1, 2))),
                "Cauchy" = rcauchy(n = 100))
# do not use lapply() as it does not work well with match.call() in
# bootstrap()
igmm.ests <- list()
conv.analyses <- list()
for (nn in names(sim.data)) {
  igmm.ests[[nn]] <- IGMM(sim.data[[nn]], type = "s")
  conv.analyses[[nn]] <- analyze_convergence(igmm.ests[[nn]])
}
plot.lists <- lapply(conv.analyses, plot)
for (nn in names(plot.lists)) {
  plot.lists[[nn]] <- lapply(plot.lists[[nn]], "+", ggtitle(nn))
}

require(gridExtra)
for (jj in seq_along(plot.lists[[1]])) {
  grid.arrange(plot.lists[[1]][[jj]], plot.lists[[2]][[jj]], ncol = 2)
}

## End(Not run)
```

`beta-utils` *Utilities for parameter vector beta of the input distribution*

Description

The parameter $\boldsymbol{\beta}$ specifies the input distribution $X \sim F_X(x \mid \boldsymbol{\beta})$.

`beta2tau` converts $\boldsymbol{\beta}$ to the transformation vector $\tau = (\mu_x, \sigma_x, \gamma = 0, \alpha = 1, \delta = 0)$, which defines the Lambert W$\times$ F random variable mapping from X to Y (see `tau-utils`). Parameters μ_x and σ_x of X in general depend on $\boldsymbol{\beta}$ (and may not even exist for `use.mean.variance = TRUE`; in this case `beta2tau` will throw an error).

`check_beta` checks if $\boldsymbol{\beta}$ defines a valid distribution, e.g., for normal distribution `'sigma'` must be positive.

`estimate_beta` estimates $\boldsymbol{\beta}$ for a given F_X using MLE or methods of moments. Closed form solutions are used if they exist; otherwise the MLE is obtained numerically using `fitdistr`.

`get_beta_names` returns (typical) names for each component of $\boldsymbol{\beta}$.

Depending on the distribution $\boldsymbol{\beta}$ has different length and names: e.g., for a `"normal"` distribution `beta` is of length 2 (`"mu"`, `"sigma"`); for an `"exp"`onential distribution `beta` is a scalar (rate `"lambda"`).

Usage

```
beta2tau(beta, distname, use.mean.variance = TRUE)

check_beta(beta, distname)

estimate_beta(x, distname)

get_beta_names(distname)
```

Arguments

`beta`	numeric; vector $\boldsymbol{\beta}$ of the input distribution; specifications as they are for the R implementation of this distribution. For example, if `distname = "exp"`, then `beta = 2` means that the rate of the exponential distribution equals 2; if `distname = "normal"` then `beta = c(1,2)` means that the mean and standard deviation are 1 and 2, respectively.
`distname`	character; name of input distribution; see `get_distnames`.
`use.mean.variance`	logical; if `TRUE` it uses mean and variance implied by $\boldsymbol{\beta}$ to do the transformation (Goerg 2011). If `FALSE`, it uses the alternative definition from Goerg (2016) with location and scale parameter.
`x`	a numeric vector of real values (the *input* data).

Details

`estimate_beta` does not do any data transformation as part of the Lambert W× F input/output framework. For an initial estimate of θ for Lambert W× F distributions see `get_initial_theta` and `get_initial_tau`.

A quick initial estimate of θ is obtained by first finding the (approximate) input $\widehat{\boldsymbol{x}}_{\widehat{\theta}}$ by `IGMM`, and then getting the MLE of $\boldsymbol{\beta}$ for this input data $\widehat{\boldsymbol{x}}_{\widehat{\theta}} \sim F_X(x \mid \boldsymbol{\beta})$ (usually using `fitdistr`).

Value

`beta2tau` returns a numeric vector, which is $\tau = \tau(\boldsymbol{\beta})$ implied by `beta` and `distname`.

`check_beta` throws an error if $\boldsymbol{\beta}$ is not appropriate for the given distribution; e.g., if it has too many values or if they are not within proper bounds (e.g., `beta['sigma']` of a `"normal"` distribution must be positive).

`estimate_beta` returns a named vector with estimates for β given `x`.

`get_beta_names` returns a vector of characters.

See Also

`tau-utils`, `theta-utils`

Examples

```
# By default: delta = gamma = 0 and alpha = 1
beta2tau(c(1, 1), distname = "normal")
## Not run:
  beta2tau(c(1, 4, 1), distname = "t")

## End(Not run)
beta2tau(c(1, 4, 1), distname = "t", use.mean.variance = FALSE)
beta2tau(c(1, 4, 3), distname = "t") # no problem


## Not run:
check_beta(beta = c(1, 1, -1), distname = "normal")

## End(Not run)


set.seed(124)
xx <- rnorm(100)^2
estimate_beta(xx, "exp")
estimate_beta(xx, "chisq")
```

`bootstrap` *Bootstrap Lambert W x F estimates*

Description

Analyzes the Lambert W x F for a given dataset based on bootstrapping. Depends on the **boot** package and returns a `"boot"` object.

Usage

```
bootstrap(object, ...)

## S3 method for class 'LambertW_fit'
bootstrap(object, sample.size = length(object$data), R = 100, ...)
```

Arguments

`object`	an object of class `"LambertW_fit"`; usually output of `IGMM` or `MLE_LambertW`.
`...`	additional arguments passed to `boot`.
`sample.size`	sample size of the bootstrap. By default, equal to the original data length.
`R`	number of replicates for the bootstrap. See `boot` for details.

Value

An object of class `"boot"` representing the bootstrap analysis of $\hat{\theta}$ (or $\hat{\tau}$) of an Lambert W x F estimator (`LambertW_fit`).

Examples

```
## Not run:
yy <- rLambertW(n = 1000, theta = list(delta = c(0.1), beta = c(2, 1)),
                distname = "normal")
mod.igmm <- IGMM(yy, type = "h")
boot.est <- bootstrap(mod.igmm, R = 100)
# use summary and plot from 'boot' pkg
plot(boot.est, 3)
summary(boot.est)

## End(Not run)
```

`common-arguments` *Common arguments for several functions*

Description

Reference list of most common function arguments in this package.

Arguments

`y`	a numeric vector of real values (the observed data).
`distname`	character; name of input distribution; see `get_distnames`.
`type`	type of Lambert W × F distribution: skewed `"s"`; heavy-tail `"h"`; or skewed heavy-tail `"hh"`.
`theta`	list; a (possibly incomplete) list of parameters `alpha`, `beta`, `gamma`, `delta`. `complete_theta` fills in default values for missing entries.
`beta`	numeric vector (deprecated); parameter β of the input distribution. See `check_beta` on how to specify `beta` for each distribution.
`gamma`	scalar (deprecated); skewness parameter; default: `0`.
`delta`	scalar or vector (length 2) (deprecated); heavy-tail parameter(s); default: `0`.
`alpha`	scalar or vector (length 2) (deprecated); heavy tail exponent(s); default: `1`.

`tau`	named vector τ which defines the variable transformation. Must have at least `'mu_x'` and `'sigma_x'` element; see `complete_tau` for details.
`return.u`	logical; if `TRUE`, it returns the standardized input that corresponds to U, which is the zero-mean and/or unit-variance version of input $X \sim F_X$.
`use.mean.variance`	logical; if `TRUE` it uses mean and variance implied by $\boldsymbol{\beta}$ to do the transformation (Goerg 2011). If `FALSE`, it uses the alternative definition from Goerg (2016) with location and scale parameter.

`datasets` *Datasets*

Description

Collection of datasets in this package.

The Australian athletes dataset (`AA`) were collected in a study of how data on various characteristics of the blood varied with sport body size and sex of the athlete.

The `SolarFlares` data are $12,773$ observations of peak gamma-ray intensity of solar flares recorded from Feb, 1980 - Dec, 1989. It was analyzed for power-law properties in Clauset et al. (2009) and comes originally from Dennis et al. (1991). Thanks to the authors for giving permission to include the dataset in this package.

Format

`AA` is a `data.frame` with 13 columns and 202 rows. See `ais` dataset in the **DAAG** package for details.

Source

`AA` was the basis for the analyses that are reported in Telford and Cunningham (1991).

Resources on the `SolarFlares` dataset can be found at:

`https://sites.santafe.edu/~aaronc/powerlaws/data.htm`

`https://ui.adsabs.harvard.edu/abs/1991chxb.book.....D/abstract`

See also References.

References

Telford, R.D. and Cunningham, R.B. 1991. Sex, sport and body-size dependency of hematology in highly trained athletes. Medicine and Science in Sports and Exercise 23: 788-794.

Dennis, B. R.; Orwig, L. E.; Kennard, G. S.; Labow, G. J.; Schwartz, R. A.; Shaver, A. R.; Tolbert, A. K. (1991). “The Complete Hard X Ray Burst Spectrometer Event List, 1980-1989.” NASA Technical Memorandum 4332.

Clauset, A., C. R. Shalizi, and M. E. J. Newman (2009). “Power-law distributions in empirical data”. SIAM Review 51, 661-703 (2009). See also `https://sites.santafe.edu/~aaronc/powerlaws/data.htm`.

`delta_01` *Input parameters to get zero mean, unit variance output given delta*

Description

Computes the input mean $\mu_x(\delta)$ and standard deviation $\sigma_x(\delta)$ for input $X \sim F(x \mid \boldsymbol{\beta})$ such that the resulting heavy-tail Lambert W x F RV Y with δ has zero-mean and unit-variance. So far works only for Gaussian input and scalar δ.

The function works for any output mean and standard deviation, but default values are $\mu_y = 0$ and $\sigma_y = 1$ since they are the most useful, e.g., to generate a standardized Lambert W white noise sequence.

Usage

```
delta_01(delta, mu.y = 0, sigma.y = 1, distname = "normal")
```

Arguments

`delta`	scalar; heavy-tail parameter.
`mu.y`	output mean; default: `0`.
`sigma.y`	output standard deviation; default: `1`.
`distname`	string; distribution name. Currently this function only supports `"normal"`.

Value

5-dimensional vector $(\mu_x(\delta), \sigma_x(\delta), 0, \delta, 1)$, where $\gamma = 0$ and $\alpha = 1$ are set for the sake of compatiblity with other functions.

Examples

```
delta_01(0) # for delta = 0, input == output, therefore (0,1,0,0,1)
# delta > 0 (heavy-tails):
#   Y is symmetric for all delta:
#   mean = 0; however, sd must be smaller
delta_01(0.1)
delta_01(1/3)  # only moments up to order 2 exist
delta_01(1)  # neither mean nor variance exist, thus NA
```

`delta_GMM` *Estimate delta*

Description

This function minimizes the Euclidean distance between the sample kurtosis of the back-transformed data $W_\delta(z)$ and a user-specified target kurtosis as a function of δ (see References). Only an iterative application of this function will give a good estimate of δ (see `IGMM`).

Usage

```
delta_GMM(
  z,
  type = c("h", "hh"),
  kurtosis.x = 3,
  skewness.x = 0,
  delta.init = delta_Taylor(z),
  tol = .Machine$double.eps^0.25,
  not.negative = FALSE,
  optim.fct = c("nlm", "optimize"),
  lower = -1,
  upper = 3
)
```

Arguments

`z`	a numeric vector of data values.
`type`	type of Lambert W $\times$ F distribution: skewed `"s"`; heavy-tail `"h"`; or skewed heavy-tail `"hh"`.
`kurtosis.x`	theoretical kurtosis of the input X; default: `3` (e.g., for $X \sim$ Gaussian).
`skewness.x`	theoretical skewness of the input X. Only used if `type = "hh"`; default: `0` (e.g., for $X \sim$ symmetric).
`delta.init`	starting value for optimization; default: `delta_Taylor`.
`tol`	a positive scalar; tolerance level for terminating the iterative algorithm; default: `.Machine$double.eps^0.25`.
`not.negative`	logical; if `TRUE` the estimate for δ is restricted to the non-negative reals. Default: `FALSE`.
`optim.fct`	which R optimization function should be used. Either `'optimize'` (only for `type = 'h'` and if `not.negative = FALSE`) or `'nlm'`. Performance-wise there is no big difference.
`lower, upper`	lower and upper bound for optimization. Default: `-1` and `3` (this covers most real-world heavy-tail scenarios).

Value

A list with two elements:

`delta`	optimal δ for data z,
`iterations`	number of iterations (`NA` for `'optimize'`).

See Also

`gamma_GMM` for the skewed version of this function; `IGMM` to estimate all parameters jointly.

Examples

```
# very heavy-tailed (like a Cauchy)
y <- rLambertW(n = 1000, theta = list(beta = c(1, 2), delta = 1),
               distname = "normal")
delta_GMM(y) # after the first iteration
```

`delta_Taylor` *Estimate of delta by Taylor approximation*

Description

Computes an initial estimate of δ based on the Taylor approximation of the kurtosis of Lambert W $\times$ Gaussian RVs. See Details for the formula.

This is the initial estimate for `IGMM` and `delta_GMM`.

Usage

```
delta_Taylor(y, kurtosis.y = kurtosis(y), distname = "normal")
```

Arguments

`y`	a numeric vector of data values.
`kurtosis.y`	kurtosis of y; default: empirical kurtosis of data `y`.
`distname`	string; name of the distribution. Currently only supports `"normal"`.

Details

The second order Taylor approximation of the theoretical kurtosis of a heavy tail Lambert W x Gaussian RV around $\delta = 0$ equals

$$\gamma_2(\delta) = 3 + 12\delta + 66\delta^2 + \mathcal{O}(\delta^3).$$

Ignoring higher order terms, using the empirical estimate on the left hand side, and solving for δ yields (positive root)

$$\widehat{\delta}_{Taylor} = \frac{1}{66} \cdot \left(\sqrt{66\widehat{\gamma}_2(\mathbf{y}) - 162} - 6 \right),$$

where $\widehat{\gamma}_2(\mathbf{y})$ is the empirical kurtosis of $\mathbf{y}$.

Since the kurtosis is finite only for $\delta < 1/4$, `delta_Taylor` upper-bounds the returned estimate by 0.25.

Value

scalar; estimated δ.

See Also

`IGMM` to estimate all parameters jointly.

Examples

```
set.seed(2)
# a little heavy-tailed (kurtosis does exist)
y <- rLambertW(n = 1000, theta = list(beta = c(0, 1), delta = 0.2),
               distname = "normal")
# good initial estimate since true delta=0.2 close to 0, and
# empirical kurtosis well-defined.
delta_Taylor(y)
delta_GMM(y) # iterative estimate

y <- rLambertW(n = 1000, theta = list(beta = c(0, 1), delta = 1),
               distname = "normal") # very heavy-tailed (like a Cauchy)
delta_Taylor(y) # bounded by 1/4 (as otherwise kurtosis does not exist)
delta_GMM(y) # iterative estimate
```

`deprecated-functions` *List of deprecated functions*

Description

These functions have been deprecated in v0.5 of **LambertW** mostly for sake of following R style guides with respect to naming of functions. This means that all deprecated functions here have an analogous function with a similar – more style consistent – name. See also the `NEWS` file.

As of v0.6.8-1 deprecated functions will throw errors (‘stop()‘) and print out the suggested new function (name).

Usage

```
beta_names(...)

bounds_theta(...)

d1W_1(z, W.z = W(z, branch = -1))
```

```
p_1(...)

params2theta(...)

skewness_test(...)

starting_theta(...)

support(...)

normfit(...)

theta2params(...)

vec.norm(...)

W_1(z)

W_gamma_1(z, gamma)

H(...)
```

Arguments

`...`	arguments passed to deprecated functions.
`z`, `W.z`	see `deriv_W`
`gamma`	see `W_gamma`.

`distname-utils` *Utilities for distributions supported in this package*

Description

The Lambert W× F framework can take any (continuous) random variable with distribution F and make it skewed (`type = "s"`), heavy tailed (`type = "h"`), or both (`type = "hh"`).

In principle, this works for any F. Of course, this package implements only a finite number of distributions, which can be specified with the `distname` argument. Most functions in this package, however, also allow you to pass your own distribution and parameters and create a Lambert W× F version of it.

`check_distname` checks if the distribution specified by the `distname` argument is implemented in this package.

`get_distname_family` determines whether a distribution is a location, scale, or location-scale family. It also returns whether the distribution is supported on non-negative values only.

`get_distnames` lists all currently implemented distributions F_X.

Usage

```
check_distname(distname)

get_distname_family(distname)

get_distnames()
```

Arguments

`distname`	character; name of input distribution; see `get_distnames`.

Value

`check_distname` returns (invisible) that the distribution is implemented, or throws an error otherwise.

`get_distname_family` returns a list with

`location`	logical; if `TRUE`, the distribution is a location family,
`scale`	logical; if `TRUE`, the distribution is a scale family.
`is.non.negative`	logical; if `TRUE`, the distribution has support only for the non-negative reals (this is usually the case when `location` = `FALSE` and `scale` = `TRUE`)

`get_distnames` returns a vector of strings in alphabetical order. It lists all supported distributions. Each string can be passed as the `distname` argument to several functions in this package.

See Also

`create_LambertW_input`, `create_LambertW_output`.

Examples

```
check_distname("normal")
## Not run:
check_distname("my_great_distribution")

## End(Not run)

get_distname_family("normal")
```

`gamma_01` *Input parameters to get a zero mean, unit variance output for a given gamma*

Description

Computes the input mean $\mu_x(\gamma)$ and standard deviation $\sigma_x(\gamma)$ for input $X \sim F(x \mid \boldsymbol{\beta})$ such that the resulting skewed Lambert W x F RV Y with γ has zero-mean and unit-variance. So far works only for Gaussian input and scalar γ.

The function works for any output mean and standard deviation, but $\mu_y = 0$ and $\sigma_y = 1$ are set as default as they are the most useful, e.g., to generate a standardized Lambert W white noise sequence.

Usage

```
gamma_01(gamma, mu.y = 0, sigma.y = 1, distname = "normal")
```

Arguments

`gamma`	skewness parameter
`mu.y`	output mean; default: `0`.
`sigma.y`	output standard deviation; default: `1`.
`distname`	string; name of distribution. Currently only supports `"normal"`.

Value

A 5-dimensional vector ($\mu_x(\gamma)$, $\sigma_x(\gamma)$, γ, 0, 1), where $\delta = 0$ and $\alpha = 1$ are set for the sake of compatiblity with other functions.

Examples

```
gamma_01(0) # for gamma = 0, input == output, therefore (0,1,0,0,1)
# input mean must be slightly negative to get a zero-mean output
gamma_01(0.1) # gamma = 0.1 means it is positively skewed
gamma_01(1)
```

`gamma_GMM` *Estimate gamma*

Description

This function minimizes the Euclidean distance between the theoretical skewness of a skewed Lambert W x Gaussian random variable and the sample skewness of the back-transformed data $W_\gamma(\boldsymbol{z})$ as a function of γ (see References). Only an interative application of this function will give a good estimate of γ (see `IGMM`).

Usage

```
gamma_GMM(
  z,
  skewness.x = 0,
  gamma.init = gamma_Taylor(z),
  robust = FALSE,
  tol = .Machine$double.eps^0.25,
  not.negative = FALSE,
  optim.fct = c("optimize", "nlminb")
)
```

Arguments

`z`	a numeric vector of data values.
`skewness.x`	theoretical skewness of the input X; default: `0`.
`gamma.init`	starting value for γ; default: `gamma_Taylor`.
`robust`	logical; if `TRUE`, robust measure of asymmetry (`medcouple_estimator`) will be used; default: `FALSE`.
`tol`	a positive scalar; tolerance level for terminating the iterative algorithm; default: `.Machine$double.eps^0.25`.
`not.negative`	logical; if `TRUE`, the estimate for γ is restricted to non-negative reals, which is useful for scale-family Lambert W$\times$ F random variables. Default: `FALSE`.
`optim.fct`	string; which R optimization function should be used. By default it uses `optimize` which is about 8-10x faster than `nlminb`.

Value

A list with two elements:

`gamma`	scalar; optimal γ,
`iterations`	number of iterations (`NA` for `"optimize"`).

See Also

`delta_GMM` for the heavy-tail version of this function; `medcouple_estimator` for a robust measure of asymmetry; `IGMM` for an iterative method to estimate all parameters jointly.

Examples

```
# highly skewed
y <- rLambertW(n = 1000, theta = list(beta = c(1, 2), gamma = 0.5),
               distname = "normal")
gamma_GMM(y, optim.fct = "nlminb")
gamma_GMM(y)
```

`gamma_Taylor` *Estimate gamma by Taylor approximation*

Description

Computes an initial estimate of γ based on the Taylor approximation of the skewness of Lambert W $\times$ Gaussian RVs around $\gamma = 0$. See Details for the formula.

This is the initial estimate for `IGMM` and `gamma_GMM`.

Usage

```
gamma_Taylor(y, skewness.y = skewness(y), skewness.x = 0, degree = 3)
```

Arguments

`y`	a numeric vector of data values.
`skewness.y`	skewness of y; default: empirical skewness of data `y`.
`skewness.x`	skewness for input X; default: 0 (symmetric input).
`degree`	degree of the Taylor approximation; in Goerg (2011) it just uses the first order approximation ($6 \cdot \gamma$); a much better approximation is the third order ($6 \cdot \gamma + 8 \cdot \gamma^3$). By default it uses the better `degree = 3` approximation.

Details

The first order Taylor approximation of the theoretical skewness γ_1 (not to be confused with the skewness parameter γ) of a Lambert W x Gaussian random variable around $\gamma = 0$ equals

$$\gamma_1(\gamma) = 6\gamma + \mathcal{O}(\gamma^3).$$

Ignoring higher order terms, using the empirical estimate on the left hand side, and solving γ yields a first order Taylor approximation estimate of γ as

$$\widehat{\gamma}_{Taylor}^{(1)} = \frac{1}{6}\widehat{\gamma}_1(\mathbf{y}),$$

where $\widehat{\gamma}_1(\mathbf{y})$ is the empirical skewness of the data $\mathbf{y}$.

As the Taylor approximation is only good in a neighborhood of $\gamma = 0$, the output of `gamma_Taylor` is restricted to the interval $(-0.5, 0.5)$.

The solution of the third order Taylor approximation

$$\gamma_1(\gamma) = 6\gamma + 8\gamma^3 + \mathcal{O}(\gamma^5),$$

is also supported. See code for the solution to this third order polynomial.

Value

Scalar; estimate of γ.

See Also

`IGMM` to estimate all parameters jointly.

Examples

```
set.seed(2)
# a little skewness
yy <- rLambertW(n = 1000, theta = list(beta = c(0, 1), gamma = 0.1),
                distname = "normal")
# Taylor estimate is good because true gamma = 0.1 close to 0
gamma_Taylor(yy)

# very highly negatively skewed
yy <- rLambertW(n = 1000, theta = list(beta = c(0, 1), gamma = -0.75),
                distname = "normal")
# Taylor estimate is bad since gamma = -0.75 is far from 0;
# and gamma = -0.5 is the lower bound by default.
gamma_Taylor(yy)
```

`Gaussianize` *Gaussianize matrix-like objects*

Description

`Gaussianize` is probably the most useful function in this package. It works the same way as `scale`, but instead of just centering and scaling the data, it actually *Gaussianizes* the data (works well for unimodal data). See Goerg (2011, 2016) and Examples.

Important: For multivariate input `X` it performs a column-wise Gaussianization (by simply calling `apply(X, 2, Gaussianize)`), which is only a marginal Gaussianization. This does *not* mean (and is in general definitely not the case) that the transformed data is then jointly Gaussian.

By default `Gaussianize` returns the $X \sim N(\mu_x, \sigma_x^2)$ input, not the zero-mean, unit-variance $U \sim N(0, 1)$ input. Use `return.u = TRUE` to obtain U.

Usage

```
Gaussianize(
  data = NULL,
  type = c("h", "hh", "s"),
  method = c("IGMM", "MLE"),
  return.tau.mat = FALSE,
  inverse = FALSE,
  tau.mat = NULL,
  verbose = FALSE,
  return.u = FALSE,
  input.u = NULL
)
```

Arguments

`data`	a numeric matrix-like object; either the data that should be Gaussianized; or the data that should ”DeGaussianized” (`inverse = TRUE`), i.e., converted back to the original space.
`type`	what type of non-normality: symmetric heavy-tails `"h"` (default), skewed heavy-tails `"hh"`, or just skewed `"s"`.
`method`	what estimator should be used: `"MLE"` or `"IGMM"`. `"IGMM"` gives exactly Gaussian characteristics (kurtosis $\equiv 3$ for `"h"` or skewness $\equiv 0$ for `"s"`), `"MLE"` comes close to this. Default: `"IGMM"` since it is much faster than `"MLE"`.
`return.tau.mat`	logical; if `TRUE` it also returns the estimated τ parameters as a matrix (same number of columns as `data`). This matrix can then be used to `Gaussianize` new data with pre-estimated τ. It can also be used to “DeGaussianize” data by passing it as an argument (`tau.mat`) to `Gaussianize()` and set `inverse = TRUE`.
`inverse`	logical; if `TRUE` it performs the inverse transformation using `tau.mat` to "De-Gaussianize" the data back to the original space again.
`tau.mat`	instead of estimating τ from the data you can pass it as a matrix (usually obtained via `Gaussianize(..., return.tau.mat = TRUE)`). If `inverse = TRUE` it uses this `tau` matrix to “DeGaussianize” the data again. This is useful to back-transform new data in the Gaussianized space, e.g., predictions or fits, back to the original space.
`verbose`	logical; if `TRUE`, it prints out progress information in the console. Default: `FALSE`.
`return.u`	logical; if `TRUE` it returns the zero-mean, unit variance Gaussian input. If `FALSE` (default) it returns the input X.
`input.u`	optional; if you used `return.u = TRUE` in a previous step, and now you want to convert the data back to original space, then you have to pass it as `input.u`. If you pass numeric data as `data`, `Gaussianize` assumes that `data` is the input corresponding to X, not U.

Value

numeric matrix-like object with same dimension/size as input `data`. If `inverse = FALSE` it is the Gaussianize matrix / vector; if `TRUE` it is the “DeGaussianized” matrix / vector.

The numeric parameters of mean, scale, and skewness/heavy-tail parameters that were used in the Gaussianizing transformation are returned as attributes of the output matrix: `'Gaussianized:mu'`, `'Gaussianized:sigma'`, and for

`type = "h"`:	`'Gaussianized:delta'` & `'Gaussianized:alpha'`,
`type = "hh"`:	`'Gaussianized:delta_l'` and `'Gaussianized:delta_r'` & `'Gaussianized:alpha_l'` and `'Gaussianized:alpha_r'`,
`type = "s"`:	`'Gaussianized:gamma'`.

They can also be returned as a separate matrix using `return.tau.mat = TRUE`. In this case `Gaussianize` returns a list with elements:

`input`	Gaussianized input data $\boldsymbol{x}$ (or $\boldsymbol{u}$ if `return.u` = `TRUE`),
`tau.mat`	matrix with τ estimates that we used to get `x`; has same number of columns as `x`, and 3, 5, or 6 rows (depending on `type='s'`, `'h'`, or `'hh'`).

Examples

```
# Univariate example
set.seed(20)
y1 <- rcauchy(n = 100)
out <- Gaussianize(y1, return.tau.mat = TRUE)
x1 <- get_input(y1, c(out$tau.mat[, 1]))  # same as out$input
test_normality(out$input) # Gaussianized a Cauchy!

kStartFrom <- 20
y.cum.avg <- (cumsum(y1)/seq_along(y1))[-seq_len(kStartFrom)]
x.cum.avg <- (cumsum(x1)/seq_along(x1))[-seq_len(kStartFrom)]

plot(c((kStartFrom + 1): length(y1)), y.cum.avg, type=”l” , lwd = 2,
     main=”CLT in practice”, xlab = ”n”,
     ylab=”Cumulative sample average”,
     ylim = range(y.cum.avg, x.cum.avg))
lines(c((kStartFrom+1): length(y1)), x.cum.avg, col=2, lwd=2)
abline(h = 0)
grid()
legend(”bottomright”, c(”Cauchy”, ”Gaussianize”), col = c(1, 2),
       box.lty = 0, lwd = 2, lty = 1)

plot(x1, y1, xlab=”Gaussian-like input”, ylab = ”Cauchy - output”)
grid()
## Not run:
# multivariate example
y2 <- 0.5 * y1 + rnorm(length(y1))
YY <- cbind(y1, y2)
plot(YY)

XX <- Gaussianize(YY, type = ”hh”)
plot(XX)

out <- Gaussianize(YY, type = ”h”, return.tau.mat = TRUE,
                   verbose = TRUE, method = ”IGMM”)

plot(out$input)
out$tau.mat

YY.hat <- Gaussianize(data = out$input, tau.mat = out$tau.mat,
                      inverse = TRUE)
plot(YY.hat[, 1], YY[, 1])

## End(Not run)
```

`get_gamma_bounds` *Get bounds for gamma*

Description

`get_gamma_bounds` returns lower and upper bounds for γ, so that the observed data range falls within the theoretical bounds of the support of the distribution. This is only important for location family input.

Usage

```
get_gamma_bounds(y, tau)
```

Arguments

`y`	a numeric vector of real values (the observed data).
`tau`	named vector τ which defines the variable transformation. Must have at least `'mu_x'` and `'sigma_x'` element; see `complete_tau` for details.

Details

Skewed Lambert W× F distributions have parameter-dependent support for location family input. Thus the parameter γ must be bounded such that the observed data is within the theoretical support of the distribution. This theoretical bounds are determined by the Lambert W function (`W`), which has only real-valued solutions for $z \geq -1/\exp(1)$. Thus, `W_gamma` has real-valued solutions only for $z \geq -1/\exp(1)\gamma$ These lower and upper bounds are determined by minimum and maxiumum of the normalized data $\mathbf{z} = (\mathbf{y} - \mu_x)/\sigma_x$.

Value

`get_gamma_bounds` returns a vector of length 2 with `"lower"` and `"upper"` bounds of γ given the range of `y`.

`get_input` *Back-transform Y to X*

Description

`get_input` back-transforms the observed data $\boldsymbol{y}$ to the (approximate) input data $\boldsymbol{x}_\tau$ using the transformation vector $\tau = (\mu_x(\boldsymbol{\beta}), \sigma_x(\boldsymbol{\beta}), \gamma, \alpha, \delta)$.

Note that `get.input` should be deprecated; however, since it was explicitly referenced in Goerg (2011) I keep it here for future reference. New code should use `get_input` exclusively.

Usage

```
get_input(y, tau, return.u = FALSE)

get.input(...)
```

Arguments

`y`	a numeric vector of data values or an object of class `LambertW_fit`.
`tau`	named vector τ which defines the variable transformation. Must have at least `'mu_x'` and `'sigma_x'` element; see `complete_tau` for details.
`return.u`	should the normalized input be returned; default: `FALSE`.
`...`	arguments passed to `get_input`.

Value

The (approximated) input data vector $\widehat{\boldsymbol{x}}_{\tau}$.

For `gamma != 0` it uses the principal branch solution `W_gamma(z, branch = 0)` to get a unique input.

For `gamma = 0` the back-transformation is bijective (for any $\delta \geq 0, \alpha \geq 0$).

If `return.u = TRUE`, then it returns a list with 2 vectors

`u`	centered and normalized input $\widehat{\boldsymbol{u}}_{\theta}$,
`x`	input data $\widehat{\boldsymbol{x}}_{\theta}$.

See Also

`get_output`

Examples

```
set.seed(12)
# unskew very skewed data
y <- rLambertW(n = 1000, theta = list(beta = c(0, 1), gamma = 0.3),
               distname = "normal")
test_normality(y)
fit.gmm <- IGMM(y, type="s")

x <- get_input(y, fit.gmm$tau)
# the same as
x <- get_input(fit.gmm)
test_normality(x) # symmetric Gaussian
```

`get_output` *Transform input X to output Y*

Description

`get_output` transforms the input $\boldsymbol{x}$ to the observed data $\boldsymbol{y}$ given the transformation vector $\tau = (\mu_x(\boldsymbol{\beta}), \sigma_x(\boldsymbol{\beta}), \gamma, \alpha, \delta)$.

This is the inverse of `get_input`.

Usage

```
get_output(x, tau, return.z = FALSE)
```

Arguments

`x`	a numeric vector of data values.
`tau`	named vector τ which defines the variable transformation. Must have at least `'mu_x'` and `'sigma_x'` element; see `complete_tau` for details.
`return.z`	should the shifted and scaled output also be returned? Default: `FALSE`.

Value

A numeric object of same size/dimension as input `x`.

If `return.z = TRUE`, then it returns a list with 2 vectors

`z`	shifted and scaled input $\boldsymbol{z}$,
`y`	transformed output data $\boldsymbol{y}$, which has a Lambert W × F distribution.

See Also

`get_input`; `Gaussianize` with argument `inverse = TRUE`.

`get_support` *Computes support for skewed Lambert W x F distributions*

Description

If the input $X \sim F$ has support on the entire real line $(-\infty, \infty)$, then the skewed Lambert W × F distribution has truncated support $[a, b]$, $a, b \in R \cup \pm\infty$ depending on $\boldsymbol{\beta}$ and (the sign of) γ.

For scale-families no truncation occurs.

Usage

```
get_support(tau, is.non.negative = FALSE, input.bounds = c(-Inf, Inf))
```

Arguments

`tau`	named vector τ which defines the variable transformation. Must have at least `'mu_x'` and `'sigma_x'` element; see `complete_tau` for details.
`is.non.negative`	logical; by default it is set to `TRUE` if the distribution is not a location but a scale family.
`input.bounds`	interval; the bounds of the input distribution. If `is.non.negative = FALSE`, then it will adjust it to `c(0,Inf)`; also useful for bounded input distributions, such as `"unif"`.

Details

Half-open interval on the real line (if $\gamma \neq 0$) for input with support on the entire real line. For $\gamma = 0$ the support of Y is the same as for X. Heavy-tail Lambert W RVs are not affected by truncated support (for $\delta \geq 0$); thus support is `c(lower = -Inf, upper = Inf)`.

Value

A vector of length 2 with names `'lower'` and `'upper'`.

Examples

```
get_support(c(mu_x = 0, sigma_x = 1, gamma = 0)) # as gamma = 0
# truncated on the left since gamma > 0
get_support(c(mu_x = 0, sigma_x = 1, gamma = 0.1))

# no truncation for heavy tail(s)
get_support(c(mu_x = 0, sigma_x = 1, delta = 0.1))
```

`G_delta_alpha` *Heavy tail transformation for Lambert W random variables*

Description

Heavy-tail Lambert W RV transformation: $G_{\delta,\alpha}(u) = u \exp(\frac{\delta}{2}(u^2)^{\alpha})$. Reduces to Tukey's h distribution for $\alpha = 1$ (`G_delta`) and Gaussian input.

Usage

```
G_delta_alpha(u, delta = 0, alpha = 1)

G_delta(u, delta = 0)

G_2delta_2alpha(u, delta = c(0, 0), alpha = c(1, 1))
```

Arguments

`u`	a numeric vector of real values.
`delta`	heavy tail parameter; default `delta = 0`, which implies `G_delta_alpha(u) = u`.
`alpha`	exponent in $(u^2)^\alpha$; default `alpha = 1` (Tukey's h).

Value

numeric; same dimension/size as `u`.

`H_gamma` *H transformation with gamma*

Description

Skewed Lambert W× F RV transformation: $H_{\gamma}(u) = u \exp(\gamma u)$.

Usage

```
H_gamma(u, gamma = 0)
```

Arguments

`u`	a numeric vector of real values.
`gamma`	skewness parameter; default `gamma = 0`, which implies `H_gamma(u) = u`.

Value

numeric; same dimension/size as `u`

See Also

`xexp`

`IGMM` *Iterative Generalized Method of Moments – IGMM*

Description

An iterative method of moments estimator to find this $\tau = (\mu_x, \sigma_x, \gamma)$ for `type = 's'` ($\tau = (\mu_x, \sigma_x, \delta)$ for `type = 'h'` or $\tau = (\mu_x, \sigma_x, \delta_l, \delta_r)$ for `type = "hh"`) which minimizes the distance between the sample and theoretical skewness (or kurtosis) of $\boldsymbol{x}$ and X.

This algorithm is only well-defined for data with finite mean and variance input X. See `analyze_convergence` and references therein for details.

Usage

```
IGMM(
  y,
  type = c("h", "hh", "s"),
  skewness.x = 0,
  kurtosis.x = 3,
  tau.init = get_initial_tau(y, type),
  robust = FALSE,
  tol = .Machine$double.eps^0.25,
  location.family = TRUE,
  not.negative = NULL,
  max.iter = 100,
  delta.lower = -1,
  delta.upper = 3
)
```

Arguments

`y`	a numeric vector of real values.
`type`	type of Lambert W × F distribution: skewed `"s"`; heavy-tail `"h"`; or skewed heavy-tail `"hh"`.
`skewness.x`	theoretical skewness of input X; default `0` (symmetric distribution).
`kurtosis.x`	theoretical kurtosis of input X; default `3` (Normal distribution reference).
`tau.init`	starting values for IGMM algorithm; default: `get_initial_tau`. See also `gamma_Taylor` and `delta_Taylor`.
`robust`	logical; only used for `type = "s"`. If `TRUE` a robust estimate of asymmetry is used (see `medcouple_estimator`); default: `FALSE`.
`tol`	a positive scalar specifiying the tolerance level for terminating the iterative algorithm. Default: `.Machine$double.eps^0.25`
`location.family`	logical; tell the algorithm whether the underlying input should have a location family distribution (for example, Gaussian input); default: `TRUE`. If `FALSE` (e.g., for `"exp"`onential input), then `tau['mu_x'] = 0` throughout the optimization.

`not.negative`	logical; if `TRUE`, the estimate for γ or δ is restricted to non-negative reals. If it is set to `NULL` (default) then it will be set internally to `TRUE` for heavy-tail(s) Lambert W× F distributions (`type = "h"` or `"hh"`). For skewed Lambert W× F (`type = "s"`) it will be set to `FALSE`, unless it is not a location-scale family (see `get_distname_family`).
`max.iter`	maximum number of iterations; default: `100`.
`delta.lower`, `delta.upper`	lower and upper bound for `delta_GMM` optimization. By default: `-1` and `3` which covers most real-world heavy-tail scenarios.

Details

For algorithm details see the References.

Value

A list of class `LambertW_fit`:

`tol`	see Arguments
`data`	data `y`
`n`	number of observations
`type`	see Arguments
`tau.init`	starting values for τ
`tau`	IGMM estimate for τ
`tau.trace`	entire iteration trace of $\tau^{(k)}$, $k = 0, ..., K$, where `K` <= `max.iter`.
`sub.iterations`	number of iterations only performed in GMM algorithm to find optimal γ (or δ)
`iterations`	number of iterations to update μ_x and σ_x. See References for detals.
`hessian`	Hessian matrix (obtained from simulations; see References)
`call`	function call
`skewness.x`, `kurtosis.x`	see Arguments
`distname`	a character string describing distribution characteristics given the target theoretical skewness/kurtosis for the input. Same information as `skewness.x` and `kurtosis.x` but human-readable.
`location.family`	see Arguments
`message`	message from the optimization method. What kind of convergence?
`method`	estimation method; here: `"IGMM"`

Author(s)

Georg M. Goerg

See Also

`delta_GMM`, `gamma_GMM`, `analyze_convergence`

Examples

```
# estimate tau for the skewed version of a Normal
y <- rLambertW(n = 100, theta = list(beta = c(2, 1), gamma = 0.2),
               distname = "normal")
fity <- IGMM(y, type = "s")
fity
summary(fity)
plot(fity)
## Not run:
# estimate tau for the skewed version of an exponential
y <- rLambertW(n = 100, theta = list(beta = 1, gamma = 0.5),
               distname = "exp")
fity <- IGMM(y, type = "s", skewness.x = 2, location.family = FALSE)
fity
summary(fity)
plot(fity)

# estimate theta for the heavy-tailed version of a Normal = Tukey's h
y <- rLambertW(n = 100, theta = list(beta = c(2, 1), delta = 0.2),
               distname = "normal")
system.time(
fity <- IGMM(y, type = "h")
)
fity
summary(fity)
plot(fity)

## End(Not run)
```

`ks_test_t` *One-sample Kolmogorov-Smirnov test for student-t distribution*

Description

Performs a two-sided KS test for $H_0 : X \sim t_\nu$ with c, scale s, and degrees of freedom ν. If parameters are not specified, the MLE given the data will be used (see `fitdistr`).

For estimated parameters of the t-distribution the p-values are incorrect and should be adjusted. See `ks.test` and the references therein (Durbin (1973)). As a more practical approach consider bootstrapping and estimating the p-value empirically.

Usage

```
ks_test_t(x, param = NULL)
```

Arguments

`x`	a numeric vector of data values.
`param`	3-dimensional named vector (`'location'`, `'scale'`, `'df'`) which parametrizes the student t distribution. Default: `param = NULL`, in which case it will be estimated from `x`.

Value

A list of class `"htest"` containing:

`statistic`	the value of the Kolomogorv-Smirnov statistic.
`p.value`	the p-value for the test.
`alternative`	a character string describing the alternative hypothesis.
`method`	the character string "One-sample Kolmogorov-Smirnov test student-t" plus rounded parameter values.
`data.name`	a character string giving the name(s) of the data.

See Also

`fitdistr`, `ks.test`

Examples

```
set.seed(1021)
beta.true <- c(location = 0, scale = 1, df = 4)
xx <- rt(n = 1000, df = beta.true['df'])
ks_test_t(xx)
ks_test_t(xx, beta.true)
```

`kurtosis` *Skewness and kurtosis*

Description

`kurtosis` estimates the fourth central, normalized moment from data.

`skewness` estimates the third central, normalized moment from data.

Usage

```
kurtosis(x)

skewness(x)
```

Arguments

`x`	a numeric vector.

See Also

Corresponding functions in the **moments** package.

`LambertW-toolkit` *Do-it-yourself toolkit for Lambert W× F distribution*

Description

IMPORTANT: This toolkit functionality is still under active development; function names, arguments, return values, etc. may change.

This do-it-yourself Lambert W× F toolkit implements the flexible input/output framework of Lambert W × F random variables (see References). Using a modular approach, it allows users to create their own Lambert W × 'MyFavoriteDistribution' RVs. See Details below.

If the distribution you inted to use is not already implemented (`get_distnames`), then you can create it:

create input: use `create_LambertW_input` with your favorite distribution,

create output: pass it as an input argument to `create_LambertW_output`,

use output: use Rs standard functionality for distributions such as random number generation (`rY`), pdf (`dY`) and cdf (`pY`), quantile function (`qY`), etc. for this newly generated Lambert W × 'MyFavoriteDistribution'.

`create_LambertW_output` converts the input `LambertW_input` representing random variable $X \sim F_X$ to the Lambert W × F_X output.

Usage

```
create_LambertW_input(
  distname = NULL,
  beta,
 input.u = list(beta2tau = NULL, d = NULL, p = NULL, r = NULL, q = NULL, distname =
    "MyFavoriteDistribution", is.non.negative = FALSE)
)

create_LambertW_output(
  LambertW.input = NULL,
  theta = NULL,
  distname = LambertW.input$distname
)
```

Arguments

`distname`	character; name of input distribution; see `get_distnames`.
`beta`	numeric vector (deprecated); parameter β of the input distribution. See `check_beta` on how to specify `beta` for each distribution.
`input.u`	optional; users can make their own 'Lambert W x F' distribution by supplying the necessary functions. See Description for details.
`LambertW.input`	an object of class `LambertW_input`
`theta`	list; a (possibly incomplete) list of parameters `alpha`, `beta`, `gamma`, `delta`. `complete_theta` fills in default values for missing entries.

Details

`create_LambertW_output` takes an object of class `LambertW_input` and creates a class `LambertW_output` for standard distributions as well as the user-defined distribution. This `LambertW_output` represents the RV Y $\sim$ Lambert W $\times$ 'MyFavoriteDistribution' with all its properties and R functionality, such as random number generation (`rY`), pdf (`dY`) and cdf (`pY`), etc.

`create_LambertW_input` allows users to define their own Lambert W$\times$ F distribution by supplying the necessary functions about the input random variable U and $\boldsymbol{\beta}$. Here U is the zero mean and/or unit variance version of $X \sim F_X(x \mid \boldsymbol{\beta})$ (see References).

The argument `input.u` must be a list containing all of the following:

`beta2tau` R function of `(beta)`: converts $\boldsymbol{\beta}$ to τ for the user defined distribution

`distname` optional; users can specify the name of their input distribution. By default it's called `"MyFavoriteDistribution"`. The distribution name will be used in plots and summaries of the Lambert W$\times$ F input (and output) object.

`is.non.negative` logical; users should specify whether the distribution is for non-negative random variables or not. This will help for plotting and theoretical quantile computation.

`d` R function of `(u, beta)`: probability density function (pdf) of U,

`p` R function of `(u, beta)`: cumulative distribution function (cdf) of U,

`q` R function of `(p, beta)`: quantile function of U,

`r` R function `(n, beta)`: random number generator for U,

Value

`create_LambertW_output` returns a list of class `LambertW_output` with values that are (for the most part) functions themselves (see Examples):

`d`	pdf of Y $\sim$ Lambert W $\times$ 'MyFavoriteDistribution',
`p`	cdf of Y,
`q`	quantile function for Y,
`r`	random number generator for Y,
`distname`	character string with the name of the new distribution. Format: "Lambert W x 'MyFavoriteDistribution'",
`beta, theta`	see Arguments,
`distname.with.beta`	name of the new distribution including the parameter `beta`. Format: "Lambert W x 'MyFavoriteDistribution'(beta)".

Author(s)

Georg M. Goerg

Examples

```
# create a Gaussian N(1, 2) input
Gauss.input <- create_LambertW_input("normal", beta = c(1, 2))

# create a heavy-tailed version of a normal
# gamma = 0, alpha = 1 are set by default; beta comes from input
params <- list(delta = c(0.3))
LW.Gauss <- create_LambertW_output(LambertW.input = Gauss.input,
                                   theta = params)
LW.Gauss

op <- par(no.readonly = TRUE)
par(mfrow = c(2, 1), mar = c(3, 3, 2, 1))
curve(LW.Gauss$d(x, params), -7, 10, col = "red")
# parameter will get detected automatically from the input
curve(LW.Gauss$d(x), -7, 10, col = "blue") # same in blue;

# compare to the input case (i.e. set delta = 0)
params.0 <- params
params.0$delta <- 0

# to evaluate the RV at a different parameter value,
# it is necessary to pass the new parameter
curve(LW.Gauss$d(x, params.0), -7, 10, add = TRUE, col = 1) #' par(op)

curve(LW.Gauss$p(x, params), -7, 10, col = "red")
curve(LW.Gauss$p(x, params.0), -7, 10, add = TRUE, col = 1)

test_normality(LW.Gauss$r(n = 100), add.legend = FALSE)

## generate a positively skewed version of a shifted, scaled t_3
t.input <- create_LambertW_input("t", beta = c(2, 1, 3))
t.input
params <- list(gamma = 0.05) # skew it
LW.t <- create_LambertW_output(LambertW.input = t.input, theta = params)
LW.t

plot(t.input$d, -7, 11, col = 1)
plot(LW.t$d, -7, 11, col = 2, add = TRUE)
abline(v = t.input$beta["location"], lty = 2)

# draw samples from the skewed t_3
yy <- LW.t$r(n = 100)
test_normality(yy)

### create a skewed exponential distribution
exp.input <- create_LambertW_input("exp", beta = 1)
plot(exp.input)
params <- list(gamma = 0.2)
LW.exp <- create_LambertW_output(exp.input, theta = params)
plot(LW.exp)
```

```
# create a heavy-tail exponential distribution
params <- list(delta = 0.2)
LW.exp <- create_LambertW_output(exp.input, theta = params)
plot(LW.exp)

# create a skewed chi-square distribution with 5 df
chi.input <- create_LambertW_input("chisq", beta = 5)
plot(chi.input)
params <- list(gamma = sqrt(2)*0.2)
LW.chi <- create_LambertW_output(chi.input, theta = params)
plot(LW.chi)


# a demo on how a user-defined U input needs to look like
user.tmp <- list(d = function(u, beta) dnorm(u),
                 r = function(n, beta) rnorm(n),
                 p = function(u, beta) pnorm(u),
                 q = function(p, beta) qnorm(p),
                 beta2tau = function(beta) {
                   c(mu_x = beta[1], sigma_x = beta[2],
                     gamma = 0, alpha = 1, delta = 0)
                   },
                 distname = "MyNormal",
                 is.non.negative = FALSE)
my.input <- create_LambertW_input(input.u = user.tmp, beta = c(0, 1))
my.input
plot(my.input)
```

`LambertW-utils` *Utilities for Lambert W× F Random Variables*

Description

Density, distribution, quantile function and random number generation for a Lambert W $\times$ $F_X(x \mid \boldsymbol{\beta})$ random variable with parameter $\theta = (\alpha, \boldsymbol{\beta}, \gamma, \delta)$.

Following the usual R `dqpr` family of functions (e.g., `rnorm`, `dnorm`, ...) the Lambert W× F utility functions work as expected: `dLambertW` evaluates the pdf at `y`, `pLambertW` evaluates the cdf at `y`, `qLambertW` is the quantile function, and `rLambertW` generates random samples from a Lambert W $\times$ $F_X(x \mid \boldsymbol{\beta})$ distribution.

`mLambertW` computes the first 4 central/standardized moments of a Lambert W × F. Works only for Gaussian distribution.

`qqLambertW` computes and plots the sample quantiles of the data `y` versus the theoretical Lambert W $\times$ F theoretical quantiles given θ.

Usage

```
dLambertW(
```

```
  y,
  distname = NULL,
  theta = NULL,
  beta = NULL,
  gamma = 0,
  delta = 0,
  alpha = 1,
  input.u = NULL,
  tau = NULL,
  use.mean.variance = TRUE,
  log = FALSE
)

mLambertW(
  theta = NULL,
  distname = c("normal"),
  beta,
  gamma = 0,
  delta = 0,
  alpha = 1
)

pLambertW(
  q,
  distname,
  theta = NULL,
  beta = NULL,
  gamma = 0,
  delta = 0,
  alpha = 1,
  input.u = NULL,
  tau = NULL,
  log = FALSE,
  lower.tail = FALSE,
  use.mean.variance = TRUE
)

qLambertW(
  p,
  distname = NULL,
  theta = NULL,
  beta = NULL,
  gamma = 0,
  delta = 0,
  alpha = 1,
  input.u = NULL,
  tau = NULL,
  is.non.negative = FALSE,
```

```
  use.mean.variance = TRUE
)

qqLambertW(
  y,
  distname,
  theta = NULL,
  beta = NULL,
  gamma = 0,
  delta = 0,
  alpha = 1,
  plot.it = TRUE,
  use.mean.variance = TRUE,
  ...
)

rLambertW(
  n,
  distname,
  theta = NULL,
  beta = NULL,
  gamma = 0,
  delta = 0,
  alpha = 1,
  return.x = FALSE,
  input.u = NULL,
  tau = NULL,
  use.mean.variance = TRUE
)
```

Arguments

`y, q`	vector of quantiles.
`distname`	character; name of input distribution; see `get_distnames`.
`theta`	list; a (possibly incomplete) list of parameters `alpha`, `beta`, `gamma`, `delta`. `complete_theta` fills in default values for missing entries.
`beta`	numeric vector (deprecated); parameter $\boldsymbol{\beta}$ of the input distribution. See `check_beta` on how to specify `beta` for each distribution.
`gamma`	scalar (deprecated); skewness parameter; default: `0`.
`delta`	scalar or vector (length 2) (deprecated); heavy-tail parameter(s); default: `0`.
`alpha`	scalar or vector (length 2) (deprecated); heavy tail exponent(s); default: `1`.
`input.u`	users can supply their own version of U (either a vector of simulated values or a function defining the pdf/cdf/quanitle function of U); default: `NULL`. If not `NULL`, `tau` must be specified as well.
`tau`	optional; if `input.u = TRUE`, then `tau` must be specified. Note that $\boldsymbol{\beta}$ is still taken from `theta`, but `"mu_x"`, `"sigma_x"`, and the other parameters (α, γ, δ)

	are all taken from `tau`. This is usually only used by the `create_LambertW_output` function; users usually don't need to supply this argument directly.
`use.mean.variance`	logical; if `TRUE` it uses mean and variance implied by $\boldsymbol{\beta}$ to do the transformation (Goerg 2011). If `FALSE`, it uses the alternative definition from Goerg (2016) with location and scale parameter.
`log`	logical; if `TRUE`, probabilities p are given as log(p).
`lower.tail`	logical; if `TRUE` (default), probabilities are $P(X \leq x)$ otherwise, $P(X > x)$.
`p`	vector of probability levels
`is.non.negative`	logical; by default it is set to `TRUE` if the distribution is not a location but a scale family.
`plot.it`	logical; should the result be plotted? Default: `TRUE`.
`...`	further arguments passed to or from other methods.
`n`	number of observations
`return.x`	logical; if `TRUE` not only the simulated Lambert W× F sample `y`, but also the corresponding simulated input `x` will be returned. Default `FALSE`. **Note:** if `TRUE` then `rLambertW` does not return a vector of length `n`, but a list of two vectors (each of length `n`).

Details

All functions here have an optional `input.u` argument where users can supply their own version corresponding to zero-mean, unit variance input U. This function usually depends on the input parameter $\boldsymbol{\beta}$; e.g., users can pass their own density function `dmydist <- function(u, beta) {...}` as `dLambertW(..., input.u = dmydist)`. `dLambertW` will then use this function to evaluate the pdf of the Lambert W x 'mydist' distribution.

Important: Make sure that all `input.u` in `dLambertW`, `pLambertW`, ... are supplied correctly and return correct values – there are no unit-tests or sanity checks for user-defined functions.

See the references for the analytic expressions of the pdf and cdf. For `"h"` or `"hh"` types and for scale-families of `type = "s"` quantiles can be computed analytically. For location (-scale) families of `type = "s"` quantiles need to be computed numerically.

Value

`mLambertW` returns a list with the 4 theoretical (central/standardized) moments of Y implied by $\boldsymbol{\theta}$ and `distname` (currrently, this only works for `distname = "normal"`):

`mean`	mean,
`sd`	standard deviation,
`skewness`	skewness,
`kurtosis`	kurtosis (not excess kurtosis, i.e., 3 for a Gaussian).

`rLambertW` returns a vector of length `n`. If `return.input = TRUE`, then it returns a list of two vectors (each of length `n`):

`x`	simulated input,
`y`	Lambert W random sample (transformed from `x` - see References and `get_output`).

`qqLambertW` returns a list of 2 vectors (analogous to `qqnorm`):

`x`	theoretical quantiles (sorted),
`y`	empirical quantiles (sorted).

Examples

```
###############################
######### mLambertW ###########
mLambertW(theta = list(beta = c(0, 1), gamma = 0.1))
mLambertW(theta = list(beta = c(1, 1), gamma = 0.1)) # mean shifted by 1
mLambertW(theta = list(beta = c(0, 1), gamma = 0)) # N(0, 1)

###############################
######### rLambertW ###########
set.seed(1)
# same as rnorm(1000)
x <- rLambertW(n=100, theta = list(beta=c(0, 1)), distname = "normal")
skewness(x) # very small skewness
medcouple_estimator(x) # also close to zero

y <- rLambertW(n=100, theta = list(beta = c(1, 3), gamma = 0.1),
               distname = "normal")
skewness(y) # high positive skewness (in theory equal to 3.70)
medcouple_estimator(y) # also the robust measure gives a high value

op <- par(no.readonly=TRUE)
par(mfrow = c(2, 2), mar = c(2, 4, 3, 1))
plot(x)
hist(x, prob=TRUE, 15)
lines(density(x))

plot(y)
hist(y, prob=TRUE, 15)
lines(density(y))
par(op)
###############################
######### dLambertW ###########
beta.s <- c(0, 1)
gamma.s <- 0.1

# x11(width=10, height=5)
par(mfrow = c(1, 2), mar = c(3, 3, 3, 1))
curve(dLambertW(x, theta = list(beta = beta.s, gamma = gamma.s),
                distname = "normal"),
     -3.5, 5, ylab = "",  main="Density function")
plot(dnorm, -3.5, 5, add = TRUE, lty = 2)
legend("topright" , c("Lambert W x Gaussian" , "Gaussian"), lty = 1:2)
abline(h=0)
```

```
###############################
######### pLambertW ###########

curve(pLambertW(x, theta = list(beta = beta.s, gamma = gamma.s),
                distname = "normal"),
      -3.5, 3.5, ylab = "", main = "Distribution function")
plot(pnorm, -3.5,3.5, add = TRUE, lty = 2)
legend("topleft" , c("Lambert W x Gaussian" , "Gaussian"), lty = 1:2)
par(op)

######## Animation
## Not run:
gamma.v <- seq(-0.15, 0.15, length = 31) # typical, empirical range of gamma
b <- get_support(gamma_01(min(gamma.v)))[2]*1.1
a <- get_support(gamma_01(max(gamma.v)))[1]*1.1

for (ii in seq_along(gamma.v)) {
  curve(dLambertW(x, beta = gamma_01(gamma.v[ii])[c("mu_x", "sigma_x")],
                  gamma = gamma.v[ii], distname="normal"),
        a, b, ylab="", lty = 2, col = 2, lwd = 2, main = "pdf",
        ylim = c(0, 0.45))
  plot(dnorm, a, b, add = TRUE, lty = 1, lwd = 2)
  legend("topright" , c("Lambert W x Gaussian" , "Gaussian"),
         lty = 2:1, lwd = 2, col = 2:1)
  abline(h=0)
  legend("topleft", cex = 1.3,
         c(as.expression(bquote(gamma == .(round(gamma.v[ii],3))))))
Sys.sleep(0.04)
}

## End(Not run)

###############################
######### qLambertW ###########

p.v <- c(0.01, 0.05, 0.5, 0.9, 0.95,0.99)
qnorm(p.v)
# same as above except for rounding errors
qLambertW(p.v, theta = list(beta = c(0, 1), gamma = 0), distname = "normal")
# positively skewed data -> quantiles are higher
qLambertW(p.v, theta = list(beta = c(0, 1), gamma = 0.1),
          distname = "normal")

###############################
######### qqLambertW ##########
## Not run:
y <- rLambertW(n=500, distname="normal",
               theta = list(beta = c(0,1), gamma = 0.1))

layout(matrix(1:2, ncol = 2))
qqnorm(y)
qqline(y)
```

```
qqLambertW(y, theta = list(beta = c(0, 1), gamma = 0.1),
           distname = "normal")

## End(Not run)
```

`LambertW_fit-methods` *Methods for Lambert W× F estimates*

Description

S3 methods (`print`, `plot`, `summary`, etc.) for `LambertW_fit` class returned by the `MLE_LambertW` or `IGMM` estimators.

`plot.LambertW_fit` plots a (1) histogram, (2) empirical density of the data `y`. These are compared (3) to the theoretical $F_X(x \mid \widehat{\boldsymbol{\beta}})$ and (4) Lambert W × $F_X(y \mid \widehat{\boldsymbol{\beta}})$ densities.

`print.LambertW_fit` prints only very basic information about $\widehat{\theta}$ (to prevent an overload of data/information in the console when executing an estimator).

`print.summary.LambertW_fit` tries to be smart about formatting the coefficients, standard errors, etc. and also displays "significance stars" (like in the output of `summary.lm`).

`summary.LambertW_fit` computes some auxiliary results from the estimate such as standard errors, theoretical support (only for `type="s"`), skewness tests (only for `type="hh"`), etc. See `print.summary.LambertW_fit` for print out in the console.

Usage

```
## S3 method for class 'LambertW_fit'
plot(x, xlim = NULL, show.qqplot = FALSE, ...)

## S3 method for class 'LambertW_fit'
print(x, ...)

## S3 method for class 'summary.LambertW_fit'
print(x, ...)

## S3 method for class 'LambertW_fit'
summary(object, ...)
```

Arguments

`x, object`	object of class `LambertW_fit`
`xlim`	lower and upper limit of x-axis for cdf and pdf plots.
`show.qqplot`	should a Lambert W× F QQ plot be displayed? Default: `FALSE`.
`...`	further arguments passed to or from other methods.

Value

`summary` returns a list of class `summary.LambertW_fit` containing

`call`	function call
`coefmat`	matrix with 4 columns: $\widehat{\theta}$, its standard errors, t-statistic, and two-sided p-values
`distname`	see Arguments
`n`	number of observations
`data`	original data (y)
`input`	back-transformed input data
`support`	support of output random variable Y
`data.range`	empirical data range
`method`	estimation method
`hessian`	Hessian at the optimum. Numerically obtained for `method = "MLE"`; for `method = "IGMM"` a diagonal-matrix approximation from covariance matrix obtained by simulations for $n = 1000$ samples in Goerg (2011).
`p_m1`, `p_m1n`	Probability that one (or n) observation were caused by input from the non-principal branch (see `p_m1`); only for `type = "s"`.
`symmetry.p.value`	p-value from Wald test of identical left and right tail parameters (see `test_symmetry`); only for `type = "hh"`.

Examples

```
# See ?LambertW-package
```

`LambertW_input_output-methods`

Methods for Lambert W input and output objects

Description

S3 methods for Lambert W input and output objects (created by `create_LambertW_input` and `create_LambertW_output`).

`plot.LambertW_input` plots the theoretical (1) pdf and (2) cdf of the input $X \sim F_X(x \mid \boldsymbol{\beta})$.

`plot.LambertW_output` plots the theoretical (1) pdf and (2) cdf of the output RV $Y \sim$ Lambert W $\times$ $F_X(x \mid \boldsymbol{\beta})$. It overlays the plot with the pdf and cdf of the input RV $X \sim F_X(x \mid \boldsymbol{\beta})$ (setting $\gamma = \delta = 0, \alpha = 1$).

`print.LambertW_input` prints an overview of the input object.

`print.LambertW_output` prints an overview of the output object.

Usage

```
## S3 method for class 'LambertW_input'
plot(x, xlim = NULL, ...)

## S3 method for class 'LambertW_output'
plot(x, xlim = NULL, ...)

## S3 method for class 'LambertW_input'
print(x, ...)

## S3 method for class 'LambertW_output'
print(x, ...)
```

Arguments

`x`	object of class `LambertW_input` or `LambertW_output`.
`xlim`	lower and upper limit of x-axis for cdf and pdf plots. If `NULL`, it tries to determine good limits based on the family type of the distribution and the quantiles. Most of the times it will show the pdf and cdf from the 0.5% to 99.5% quantile.
`...`	further arguments passed to or from other methods.

Examples

```
# create a Normal(1, 2) input
Gauss.input <- create_LambertW_input("normal", beta = c(1, 2))
plot(Gauss.input)
# make it a bit heavy tailed (beta in theta comes from Gauss.input)
LW.Gauss <- create_LambertW_output(LambertW.input = Gauss.input,
                                   theta = list(delta = c(0.3)))
LW.Gauss  # print a nice overview in the console
plot(LW.Gauss)

# draw random sample
LW.Gauss$r(n=10)
Gauss.input$r(n=10)
# quantiles
LW.Gauss$q(p=0.6)
Gauss.input$q(p=0.6)
```

`loglik-LambertW-utils` *Log-Likelihood for Lambert W× F RVs*

Description

Evaluates the log-likelihood for θ given observations `y`.

`loglik_LambertW` computes the log-likelihood of θ for a Lambert W × F distribution given observations `y`.

`loglik_input` computes the log-likelihood of various distributions for the parameter $\boldsymbol{\beta}$ given the data `x`. This can be used independently of the Lambert W x F framework to compute the log-likelihood of parameters for common distributions.

`loglik_penalty` computes the penalty for transforming the data back to the input (see Goerg 2016). This penalty is independent of the distribution specified by `distname`, but only depends on τ. If `type = "s"` then the penalty term exists if the distribution is non-negative (see `get_distname_family`) and `gamma >= 0`; otherwise, it returns `NA`.

Usage

```
loglik_LambertW(
  theta,
  y,
  distname,
  type,
  return.negative = FALSE,
  flattened.theta.names = names(theta),
  use.mean.variance = TRUE
)

loglik_input(
  beta,
  x,
  distname,
  dX = NULL,
  log.dX = function(x, beta) log(dX(x, beta))
)

loglik_penalty(tau, y, type = c("h", "hh", "s"), is.non.negative = FALSE)
```

Arguments

`theta`	list; a (possibly incomplete) list of parameters `alpha`, `beta`, `gamma`, `delta`. `complete_theta` fills in default values for missing entries.
`y`	a numeric vector of real values (the observed data).
`distname`	character; name of input distribution; see `get_distnames`.
`type`	type of Lambert W × F distribution: skewed `"s"`; heavy-tail `"h"`; or skewed heavy-tail `"hh"`.
`return.negative`	logical; if `TRUE` it returns the negative log-likelihood as a scalar (which is useful for numerical *minimization* algorithms for *maximum* likelihood estimation); otherwise it returns a list of input log-likelihood, penalty, and their sum = full likelihood. Default: `FALSE`.

`flattened.theta.names`	vector of strings with names of flattened `theta`; this is necessary for optimization functions since they drop the names of a vector, but all functions in this package use names to select elements of (the flattened) `theta`.
`use.mean.variance`	logical; if `TRUE` it uses mean and variance implied by $\boldsymbol{\beta}$ to do the transformation (Goerg 2011). If `FALSE`, it uses the alternative definition from Goerg (2016) with location and scale parameter.
`beta`	numeric vector (deprecated); parameter $\boldsymbol{\beta}$ of the input distribution. See `check_beta` on how to specify `beta` for each distribution.
`x`	a numeric vector of real values (the *input* data).
`dX`	optional; density function of `x`. Common distributions are already built-in (see `distname`). If you want to supply your own density, you **must** supply a function of `(x, beta)` and set `distname = "user"`.
`log.dX`	optional; a function that returns the logarithm of the density function of `x`. Often – in particular for exponential families – the $\log$ of $f_X(x)$ has a simpler form (and is thus faster to evaluate).
`tau`	named vector τ which defines the variable transformation. Must have at least `'mu_x'` and `'sigma_x'` element; see `complete_tau` for details.
`is.non.negative`	logical; by default it is set to `TRUE` if the distribution is not a location but a scale family.

Details

For heavy-tail Lambert W× F distributions (`type = "h"` or `type = "hh"`) the log-likelihood decomposes into an input log-likelihood plus a penalty term for transforming the data.

For skewed Lambert W × F distributions this decomposition only exists for non-negative input RVs (e.g., `"exp"`onential, `"gamma"`, `"f"`, ...). If negative values are possible (`"normal"`, `"t"`, `"unif"`, `"cauchy"`, ...) then `loglik_input` and `loglik_penalty` return `NA`, but the value of the output log-likelihood will still be returned correctly as `loglik.LambertW`.

See Goerg (2016) for details on the decomposition of the log-likelihood into a log-likelihood on the input parameters plus a penalty term for transforming the data.

Value

`loglik_input` and `loglik_penalty` return a scalar; `loglik_LambertW` returns a list with 3 values:

`loglik.input`	loglikelihood of `beta` given the transformed data,
`loglik.penalty`	penalty for transforming the data,
`loglik.LambertW`	total log-likelihood of `theta` given the observed data; if the former two values exist this is simply their sum.

Examples

```
set.seed(1)
yy <- rLambertW(n = 1000, distname = "normal",
                theta = list(beta = c(0, 1), delta = 0.2))
loglik_penalty(tau = theta2tau(list(beta = c(1, 1), delta = c(0.2, 0.2)),
                               distname = "normal"),
               y = yy, type = "hh")
# For a type = 's' Lambert W x F distribution with location family input
# such a decomposition doesn't exist; thus NA.
loglik_penalty(tau = theta2tau(list(beta = c(1, 1), gamma = 0.03),
                               distname = "normal"),
               is.non.negative = FALSE,
               y = yy, type = "s")
# For scale-family input it does exist
loglik_penalty(tau = theta2tau(list(beta = 1, gamma = 0.01),
                               distname = "exp"),
               is.non.negative = TRUE,
               y = yy, type = "s")

# evaluating the Gaussian log-likelihood
loglik_input(beta = c(0, 1), x = yy, distname = "normal") # built-in version
# or pass your own log pdf function
loglik_input(beta = c(0, 1), x = yy, distname = "user",
             log.dX = function(xx, beta = beta) {
                dnorm(xx, mean = beta[1], sd = beta[2], log = TRUE)
             })
## Not run:
# you must specify distname = 'user'; otherwise it does not work
loglik_input(beta = c(0, 1), x = yy, distname = "mydist",
             log.dX = function(xx, beta = beta) {
                dnorm(xx, mean = beta[1], sd = beta[2], log = TRUE)
                })

## End(Not run)

### loglik_LambertW returns all three values
loglik_LambertW(theta = list(beta = c(1, 1), delta = c(0.09, 0.07)),
                y = yy, type = "hh", distname ="normal")

# can also take a flattend vector; must provide names though for delta
loglik_LambertW(theta = flatten_theta(list(beta = c(1, 1),
                                           delta = c(delta_l = 0.09,
                                                     delta_r = 0.07))),
                y = yy, type = "hh", distname ="normal")
```

`lp_norm` *lp norm of a vector*

Description

Computes the ℓ^p norm of an n-dimensional (real/complex) vector $\mathbf{x} \in \mathbf{C}^n$

$$||\mathbf{x}||_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}, p \in [0, \infty],$$

where $|x_i|$ is the absolute value of x_i. For $p = 2$ this is Euclidean norm; for $p = 1$ it is Manhattan norm. For $p = 0$ it is defined as the number of non-zero elements in $\mathbf{x}$; for $p = \infty$ it is the maximum of the absolute values of $\mathbf{x}$.

The norm of $\mathbf{x}$ equals 0 if and only if $\mathbf{x} = \mathbf{0}$.

Usage

```
lp_norm(x, p = 2)
```

Arguments

`x`	n-dimensional vector (possibly complex values)
`p`	which norm? Allowed values $p \geq 0$ including `Inf`. Default: `2` (Euclidean norm).

Value

Non-negative float, the norm of $\mathbf{x}$.

Examples

```
kRealVec <- c(3, 4)
# Pythagoras
lp_norm(kRealVec)
# did not know Manhattan,
lp_norm(kRealVec, p = 1)

# so he just imagined running in circles.
kComplexVec <- exp(1i * runif(20, -pi, pi))
plot(kComplexVec)
sapply(kComplexVec, lp_norm)
```

`medcouple_estimator` *MedCouple Estimator*

Description

A robust measure of asymmetry. See References for details.

Usage

```
medcouple_estimator(x, seed = sample.int(1e+06, 1))
```

Arguments

`x`	numeric vector; if length > 3,000, it uses a random subsample (otherwise it takes too long to compute as calculations are of order N^2.)
`seed`	numeric; seed used for sampling (when `length(x)` > `3000`).

Value

float; measures the degree of asymmetry

References

Brys, G., M. Hubert, and A. Struyf (2004). “A robust measure of skewness”. Journal of Computational and Graphical Statistics 13 (4), 996 - 1017.

See Also

`test_symmetry`

Examples

```
# a simulation
kNumSim <- 100
kNumObs <- 200

################# Gaussian (Symmetric) ####
A <- t(replicate(kNumSim, {xx <- rnorm(kNumObs); c(skewness(xx), medcouple_estimator(xx))}))
########### skewed LambertW x Gaussian ####
tau.s <- gamma_01(0.2) # zero mean, unit variance, but positive skewness
rbind(mLambertW(theta = list(beta = tau.s[c("mu_x", "sigma_x")],
                             gamma = tau.s["gamma"]),
                distname="normal"))
B <- t(replicate(kNumSim,
                 {
                   xx <- rLambertW(n = kNumObs,
                                   theta = list(beta = tau.s[c("mu_x", "sigma_x")],
                                                gamma = tau.s["gamma"]),
                                   distname="normal")
                   c(skewness(xx), medcouple_estimator(xx))
                 }))

colnames(A) <- colnames(B) <- c("MedCouple", "Pearson Skewness")

layout(matrix(1:4, ncol = 2))
plot(A, main = "Gaussian")
boxplot(A)
abline(h = 0)
```

```
plot(B, main = "Skewed Lambert W x Gaussian")
boxplot(B)
abline(h = mLambertW(theta = list(beta = tau.s[c("mu_x", "sigma_x")],
                                  gamma = tau.s["gamma"]),
                     distname="normal")["skewness"])

colMeans(A)
apply(A, 2, sd)

colMeans(B)
apply(B, 2, sd)
```

`MLE_LambertW` *Maximum Likelihood Estimation for Lambert W× F distributions*

Description

Maximum Likelihood Estimation (MLE) for Lambert W $\times F$ distributions computes $\widehat{\theta}_{MLE}$.

For `type = "s"`, the skewness parameter γ is estimated and $\delta = 0$ is held fixed; for `type = "h"` the one-dimensional δ is estimated and $\gamma = 0$ is held fixed; and for `type = "hh"` the 2-dimensional δ is estimated and $\gamma = 0$ is held fixed.

By default $\alpha = 1$ is fixed for any `type`. If you want to also estimate α (for `type = "h"` or `"hh"`) set `theta.fixed = list()`.

Usage

```
MLE_LambertW(
  y,
  distname,
  type = c("h", "s", "hh"),
  theta.fixed = list(alpha = 1),
  use.mean.variance = TRUE,
 theta.init = get_initial_theta(y, distname = distname, type = type, theta.fixed =
    theta.fixed, use.mean.variance = use.mean.variance, method = "IGMM"),
  hessian = TRUE,
  return.estimate.only = FALSE,
  optim.fct = c("optim", "nlm", "solnp"),
  not.negative = FALSE
)
```

Arguments

`y`	a numeric vector of real values.
`distname`	character; name of input distribution; see `get_distnames`.
`type`	type of Lambert W × F distribution: skewed `"s"`; heavy-tail `"h"`; or skewed heavy-tail `"hh"`.

`theta.fixed`	a list of fixed parameters in the optimization; default only `alpha = 1`.
`use.mean.variance`	logical; if `TRUE` it uses mean and variance implied by $\boldsymbol{\beta}$ to do the transformation (Goerg 2011). If `FALSE`, it uses the alternative definition from Goerg (2016) with location and scale parameter.
`theta.init`	a list containing the starting values of $(\alpha, \boldsymbol{\beta}, \gamma, \delta)$ for the numerical optimization; default: see `get_initial_theta`.
`hessian`	indicator for returning the (numerically obtained) Hessian at the optimum; default: `TRUE`. If the **numDeriv** package is available it uses `numDeriv::hessian()`; otherwise `stats::optim(..., hessian = TRUE)`.
`return.estimate.only`	logical; if `TRUE`, only a named flattened vector of $\widehat{\theta}_{MLE}$ will be returned (only the estimated, non-fixed values). This is useful for simulations where it is usually not necessary to give a nicely organized output, but only the estimated parameter. Default: `FALSE`.
`optim.fct`	character; which R optimization function should be used. Either `'optim'` (default), `'nlm'`, or `'solnp'` from the **Rsolnp** package (if available). Note that if `'nlm'` is used, then `not.negative = TRUE` will be set automatically.
`not.negative`	logical; if `TRUE`, it restricts `delta` or `gamma` to the non-negative reals. See `theta2unbounded` for details.

Value

A list of class `LambertW_fit`:

`data`	data `y`,
`loglik`	scalar; log-likelihood evaluated at the optimum $\widehat{\theta}_{MLE}$,
`theta.init`	list; starting values for numerical optimization,
`beta`	estimated $\boldsymbol{\beta}$ vector of the input distribution via Lambert W MLE (In general this is not exactly identical to $\widehat{\boldsymbol{\beta}}_{MLE}$ for the input data),
`theta`	list; MLE for θ,
`type`	see Arguments,
`hessian`	Hessian matrix; used to calculate standard errors (only if `hessian = TRUE`, otherwise `NULL`),
`call`	function call,
`distname`	see Arguments,
`message`	message from the optimization method. What kind of convergence?,
`method`	estimation method; here `"MLE"`.

Examples

```
# See ?LambertW-package
```

---

p_m1 *Non-principal branch probability*

Description

Computes the probability that (at least) one (out of n) observation(s) of the latent variable U lies in the non-principal branch region. The 'm1' in `p_m1` stands for 'minus 1', i.e, the non-principal branch.

See Goerg (2011) and Details for mathematical derivations.

Usage

```
p_m1(gamma, beta, distname, n = 1, use.mean.variance = TRUE)
```

Arguments

`gamma`	scalar; skewness parameter.
`beta`	numeric vector (deprecated); parameter $\boldsymbol{\beta}$ of the input distribution. See `check_beta` on how to specify `beta` for each distribution.
`distname`	character; name of input distribution; see `get_distnames`.
`n`	number of RVs/observations.
`use.mean.variance`	logical; if `TRUE` it uses mean and variance implied by $\boldsymbol{\beta}$ to do the transformation (Goerg 2011). If `FALSE`, it uses the alternative definition from Goerg (2016) with location and scale parameter.

Details

The probability that one observation of the latent RV U lies in the non-principal region equals at most

$$p_{-1}(\gamma, n = 1) = P\left(U < -\frac{1}{|\gamma|}\right),$$

where U is the zero-mean, unit variance version of the input $X \sim F_X(x \mid \boldsymbol{\beta})$ – see References.

For N independent RVs $U_1, \ldots, U_N$, the probability that at least one data point came from the non-principal region equals

$$p_{-1}(\gamma, n = N) = P\left(U_i < -\frac{1}{|\gamma|} \; for\ at\ least\ one\ i\right)$$

This equals (assuming independence)

$$P\left(U_i < -\frac{1}{|\gamma|} \; for\ at\ least\ one\ i\right) = 1 - P\left(U_i \geq -\frac{1}{|\gamma|}, \forall i\right) = 1 - \prod_{i=1}^{N} P\left(U_i \geq -\frac{1}{|\gamma|}\right)$$

$$= 1 - \prod_{i=1}^{N} (1 - p_{-1}(\gamma, n = 1)) = 1 - (1 - p_{-1}(\gamma, n = 1))^N.$$

For improved numerical stability the cdf of a geometric RV (`pgeom`) is used to evaluate the last expression. Nevertheless, numerical problems can occur for $|\gamma| < 0.03$ (returns `0` due to rounding errors).

Note that $1 - (1 - p_{-1}(\gamma, n = 1))^N$ reduces to $p_{-1}(\gamma)$ for $N = 1$.

Value

non-negative float; the probability p_{-1} for `n` observations.

Examples

```
beta.01 <- c(mu = 0, sigma = 1)
# for n=1 observation
p_m1(0, beta = beta.01, distname = "normal") # identical to 0
# in theory != 0; but machine precision too low
p_m1(0.01, beta = beta.01, distname = "normal")
p_m1(0.05, beta = beta.01, distname = "normal") # extremely small
p_m1(0.1, beta = beta.01, distname = "normal") # != 0, but very small
# 1 out of 4 samples is a non-principal input;
p_m1(1.5, beta = beta.01, distname = "normal")
# however, gamma=1.5 is not common in practice

# for n=100 observations
p_m1(0, n=100, beta = beta.01, distname = "normal") # == 0
p_m1(0.1, n=100, beta = beta.01, distname = "normal") # still small
p_m1(0.3, n=100, beta = beta.01, distname = "normal") # a bit more likely
p_m1(1.5, n=100, beta = beta.01, distname = "normal")
# Here we can be almost 100% sure (rounding errors) that at least one
# y_i was caused by an input in the non-principal branch.
```

`tau-utils` *Utilities for transformation vector tau*

Description

All functions here are for the transformation parameter vector $\tau = (\mu_x, \sigma_x, \gamma, \delta, \alpha)$.

`check_tau` checks if τ is correctly specified (correct names, non-negativity constraints, etc.)

`complete_tau` completes missing values so users don't have to specify every element of τ explicitly. `'mu_x'` and `'sigma_x'` must be specified, but `alpha = 1`, `gamma = 0`, and `delta = 0` will be set automatically if missing.

`get_initial_tau` provides starting estimates for τ.

`normalize_by_tau` shifts and scales data given the `tau` vector as

$$(data - \mu_x)/\sigma_x.$$

Parameters μ_x and σ_x are not necessarily mean and standard deviation in the τ vector; that depends on the family type and `use.mean.variance` (for location families they usually are mean and standard deviation if `use.mean.variance = TRUE`; for scale and non-location non-scale families they are just location/scale parameters for the transformation).

`tau2theta` converts τ to the parameter list θ (inverse of `theta2tau`).

`tau2type` guesses the type (`'s'`, `'h'`, `'hh'`) from the names of `tau` vector; thus make sure `tau` is named correctly.

Usage

```
check_tau(tau)

complete_tau(tau, type = tau2type(tau))

get_initial_tau(y, type = c("h", "hh", "s"), location.family = TRUE)

normalize_by_tau(data, tau, inverse = FALSE)

tau2theta(tau, beta)

tau2type(tau)
```

Arguments

`tau`	named vector τ which defines the variable transformation. Must have at least `'mu_x'` and `'sigma_x'` element; see `complete_tau` for details.
`type`	type of Lambert W $\times$ F distribution: skewed `"s"`; heavy-tail `"h"`; or skewed heavy-tail `"hh"`.
`y`	a numeric vector of real values (the observed data).
`location.family`	logical; if `FALSE` it sets `mu_x` to 0 and only estimates `sigma_x`; if `TRUE` (default), it estimates `mu_x` as well.
`data`	numeric; a numeric object in R. Usually this is either `y` or `x` (or `z` and `u` if `inverse = TRUE`.)
`inverse`	logical; if `TRUE` it applies the inverse transformation $data \cdot \sigma_x + \mu_x$
`beta`	numeric vector (deprecated); parameter $\boldsymbol{\beta}$ of the input distribution. See `check_beta` on how to specify `beta` for each distribution.

Value

`check_tau` throws an error if τ does not define a proper transformation.

`complete_tau` returns a named numeric vector.

`get_initial_tau` returns a named numeric vector.

`tau2theta` returns a list with entries `alpha`, `beta`, `gamma`, and `delta`.

`tau2type` returns a string: either `"s"`, `"h"`, or `"hh"`.

`test_normality` *Visual and statistical Gaussianity check*

Description

Graphical and statistical check if data is Gaussian (three common Normality tests, QQ-plots, histograms, etc).

`test_normality` does not show the autocorrelation function (ACF) estimate for lag 0, since it always equals 1. Thus removing it does not lose any information, but greatly improves the y-axis scale for higher order lags (which are usually very small compared to 1).

`test_norm` is a shortcut for `test_normality`.

Usage

```
test_normality(
  data,
  show.volatility = FALSE,
  plot = TRUE,
  pch = 1,
  add.legend = TRUE,
  seed = sample(1e+06, 1)
)

test_norm(...)
```

Arguments

`data`	a numeric vector of data values.
`show.volatility`	logical; if `TRUE` the squared (centered) data and its ACF are also shown. Useful for time series data to see if squares exhibit dependence (for financial data they typically do); default: `FALSE`.
`plot`	Should visual checks (histogram, densities, qqplot, ACF) be plotted? Default `TRUE`; otherwise only hypothesis test results are returned.
`pch`	a vector of plotting characters or symbols; default `pch = 1`.
`add.legend`	logical; if `TRUE` (default) a legend is placed in histogram/density plot.
`seed`	optional; if sample size > 5,000, then some normality tests fail to run. In this case it uses a subsample of size 5,000. For reproducibility, the seed can be specified by user. By default it uses a random seed.
`...`	arguments as in `test_normality`.

Value

A list with results of 3 normality tests (each of class `htest`) and the seed used for subsampling:

`anderson.darling`	Anderson Darling (if **nortest** package is available),
`shapiro.francia`	Shapiro-Francia (if **nortest** package is available),
`shapiro.wilk`	Shapiro-Wilk,
`seed`	seed for subsampling (only used if sample size > 5,000).

References

Thode Jr., H.C. (2002): “Testing for Normality”. Marcel Dekker, New York.

See Also

`shapiro.test` in the **stats** package; `ad.test`, `sf.test` in the **nortest** package.

Examples

```
y <- rLambertW(n = 1000, theta = list(beta = c(3, 4), gamma = 0.3),
               distname = "normal")
test_normality(y)

x <- rnorm(n = 1000)
test_normality(x)

# mixture of exponential and normal
test_normality(c(rexp(100), rnorm(100, mean = -5)))
```

`test_symmetry` *Test symmetry based on Lambert W heavy tail(s)*

Description

Performs a test for the null hypothesis of symmetry, $H_0 : \delta_l = \delta_r$, versus the alternative of asymmetry. This can be done using a Wald test of the linear restriction $H_0 : \delta_l - \delta_r = 0$ or a likelihood ratio test.

By default it uses `"Wald"` test since this only requires the Hessian of the `"hh"` Lambert W fit. The `"LR"` test requires the log-likelihood values for both MLEs (type `"h"` and `"hh"`) and thus takes longer to compute.

Usage

```
test_symmetry(LambertW.fit, method = c("Wald", "LR"))
```

Arguments

`LambertW.fit`	an object of class `LambertW_fit` with `type = "hh"` or a numeric vector (observed data). If it is data, then an asymmetric Lambert W × Gaussian distribution (`distname = "normal"`) with two tail parameters (`"hh"`) will be fit to the data internally and then used as the new `LambertW.fit`.
`method`	test methodology: `"Wald"` (default) or a likelihood ratio `"LR"` test

Value

A list of class `"htest"` containing:

`statistic`	value of the test statistic,
`p.value`	p-value for the test,
`method`	character string describing the test,
`data.name`	a character string giving the name(s) of the data.

Examples

```
## Not run:
# skewed
yy <- rLambertW(n = 500, theta = list(delta = c(0.1, 0.25), beta = c(2, 1)),
                distname = "normal")
fit.ml <- MLE_LambertW(yy, type = "hh", distname = "normal",
                       hessian = TRUE)
summary(fit.ml)
test_symmetry(fit.ml, "LR")
test_symmetry(fit.ml, "Wald")

# symmetric
yy <- rLambertW(n = 500, theta = list(delta = c(0.2, 0.2), beta = c(2, 1)),
                distname = "normal")
fit.ml <- MLE_LambertW(yy, type = "hh", distname = "normal")
summary(fit.ml)
test_symmetry(fit.ml, "LR")
test_symmetry(fit.ml, "Wald")

## End(Not run)
```

`theta-utils` *Utilities for the parameter vector of Lambert W× F distributions*

Description

These functions work with $\boldsymbol{\theta} = (\boldsymbol{\beta}, \gamma, \delta, \alpha)$, which fully parametrizes Lambert W× F distributions.

See Details for more background information on some functions.

`check_theta` checks if $\theta = (\alpha, \boldsymbol{\beta}, \gamma, \delta)$ describes a well-defined Lambert W distribution.

`complete_theta` completes missing values in a parameters list so users don't have to specify everything in detail. If not supplied, then `alpha = 1`, `gamma = 0`, and `delta = 0` will be set by default.

`flatten_theta` and `unflatten_theta` convert between the list `theta` and its vector-style flattened type. The flattened version is required for several optimization routines, since they optimize over multivariate vectors – not lists.

`get_initial_theta` provides initial estimates for α, $\boldsymbol{\beta}$, γ, and δ, which are then used in maximum likelihood (ML) estimation (`MLE_LambertW`).

`get_theta_bounds` returns lower and upper bounds for θ (necessary for optimization such as `MLE_LambertW`).

`theta2tau` converts θ to the transformation vector $\tau = (\mu_x, \sigma_x, \gamma, \delta, \alpha)$.

`theta2unbounded` transforms θ from the bounded space to an unrestricted space (by $\log$-transformation on σ_x, δ, and α; note that this restricts $\gamma \geq 0$, $\delta \geq 0$, and $\alpha \geq 0$.).

Usage

```
check_theta(theta, distname)

complete_theta(theta = list(), LambertW.input = NULL)

flatten_theta(theta)

get_initial_theta(
  y,
  distname,
  type = c("h", "hh", "s"),
  theta.fixed = list(alpha = 1),
  method = c("Taylor", "IGMM"),
  use.mean.variance = TRUE
)

get_theta_bounds(
  distname,
  beta,
  type = c("s", "h", "hh"),
  not.negative = FALSE
)

theta2tau(theta = list(beta = c(0, 1)), distname, use.mean.variance = TRUE)

theta2unbounded(theta, distname, type = c("h", "hh", "s"), inverse = FALSE)

unflatten_theta(theta.flattened, distname, type)
```

Arguments

`theta`	list; a (possibly incomplete) list of parameters `alpha`, `beta`, `gamma`, `delta`. `complete_theta` fills in default values for missing entries.

`distname`	character; name of input distribution; see `get_distnames`.
`LambertW.input`	optional; if `beta` is missing in `theta`, `LambertW.input` (which has a `beta` element) must be specified.
`y`	a numeric vector of real values (the observed data).
`type`	type of Lambert W × F distribution: skewed `"s"`; heavy-tail `"h"`; or skewed heavy-tail `"hh"`.
`theta.fixed`	list; fixed parameters for the optimization; default: `alpha = 1`.
`method`	character; should a fast `"Taylor"` (default) approximation be used (`delta_Taylor` or `gamma_Taylor`) to estimate δ or γ, or should `"IGMM"` (`IGMM`) estimates be used. Use `"Taylor"` as initial values for `IGMM`; `IGMM` improves upon it and should be used for `MLE_LambertW`. Do **not** use `"IGMM"` as initial values for `IGMM` – this will run `IGMM` twice.
`use.mean.variance`	logical; if `TRUE` it uses mean and variance implied by $\boldsymbol{\beta}$ to do the transformation (Goerg 2011). If `FALSE`, it uses the alternative definition from Goerg (2016) with location and scale parameter.
`beta`	numeric vector (deprecated); parameter $\boldsymbol{\beta}$ of the input distribution. See `check_beta` on how to specify `beta` for each distribution.
`not.negative`	logical; if `TRUE` it sets the lower bounds for `alpha` and `delta` to `0`. Default: `FALSE`.
`inverse`	logical; if `TRUE`, it transforms the unbounded `theta` back to the original, bounded space. Default: `FALSE`.
`theta.flattened`	named vector; flattened version of list `theta`.

Details

`get_initial_theta` obtains a quick initial estimate of θ by first finding the (approximate) input $\widehat{\boldsymbol{x}}_{\widehat{\theta}}$ by `IGMM`, and then estimating $\boldsymbol{\beta}$ for this input data $\widehat{\boldsymbol{x}}_{\widehat{\theta}} \sim F_X(x \mid \boldsymbol{\beta})$ (see `estimate_beta`).

Converting `theta` to an unbounded space is especially useful for optimization routines (like `nlm`), which can be performed over an unconstrained space. The obtained optimum can be converted back to the original space using the inverse transformation (set `inverse = TRUE` transforms it via $\exp$) – this guarantees that the estimate satisfies non-negativity constraints (if required). The main advantage is that this avoids using optimization routines with boundary constraints – since they are much slower compared to unconstrained optimization.

Value

`check_theta` throws an error if list `theta` does not define a proper Lambert W × F distribution; does nothing otherwise.

`complete_theta` returns a list containing:

`alpha`	heavy tail exponent(s),
`beta`	named vector $\boldsymbol{\beta}$ of the input distribution,
`gamma`	skewness parameter,

`delta`	heavy-tail parameter(s).

`get_initial_theta` returns a list containing:

`alpha`	heavy tail exponent; default: `1`,
`beta`	named vector $\boldsymbol{\beta}$ of the input distribution; estimated from the recovered input data $\widehat{\mathbf{x}}_{\widehat{\tau}}$,
`gamma`	skewness parameter; if `type` is `"h"` or `"hh"` `gamma` = `0`; estimated from `IGMM`,
`delta`	heavy-tail parameter; estimated from `IGMM`. If `type = "s"`, then `delta` = `0`.

`get_theta_bounds` returns a list containing two vectors:

`lower`	flattened vector of lower bounds for valid θ,
`upper`	flattened vector of upper bounds for valid θ.

See Also

`check_beta`

`estimate_beta`, `get_initial_tau`

`beta2tau`

Examples

```
## Not run:
check_theta(theta = list(beta =  c(1, 1, -1)), distname = "t")

## End(Not run)

check_theta(theta = list(beta =  c(1, 1)), distname = "normal") # ok

params <- list(beta = c(2, 1), delta = 0.3) # alpha and gamma are missing
complete_theta(params) # added default values

params <- list(beta = c(2, 1), delta = 0.3, alpha = c(1, 2))
params <- complete_theta(params)
check_theta(params, distname = 'normal')

###
x <- rnorm(1000)
get_initial_theta(x, distname = "normal", type = "h")
get_initial_theta(x, distname = "normal", type = "s")

# starting values for the skewed version of an exponential
y <- rLambertW(n = 1000, distname = "exp", theta=list(beta = 2, gamma = 0.1))
get_initial_theta(y, distname = "exp", type = "s")

# starting values for the heavy-tailed version of a Normal = Tukey's h
y <- rLambertW(n = 1000, distname="normal", theta=list(beta = c(2, 1), delta = 0.2))
get_initial_theta(y, distname = "normal", type = "h")#'

###
```

```
get_theta_bounds(type = "hh", distname = "normal", beta = c(0, 1))

###
theta.restr <- theta2unbounded(list(beta = c(-1, 0.1),
                                    delta = c(0.2, 0.2)),
                                    distname = "normal")
theta.restr
# returns again the beta and delta from above
theta2unbounded(theta.restr, inverse = TRUE, distname = "normal")
```

`U-utils` *Zero-mean, unit-variance version of standard distributions*

Description

Density, distribution function, quantile function and random number generation for the shifted and scaled U of the (location-)scale family input $X \sim F_X(x \mid \boldsymbol{\beta})$ - see References.

Since the normalized random variable U is one of the main building blocks of Lambert W × F distributions, these functions are wrappers used by other functions such as `dLambertW` or `rLambertW`.

Usage

```
dU(u, beta, distname, use.mean.variance = TRUE)

pU(u, beta, distname, use.mean.variance = TRUE)

qU(p, beta, distname, use.mean.variance = TRUE)

rU(n, beta, distname, use.mean.variance = TRUE)
```

Arguments

`u`	vector of quantiles.
`beta`	numeric vector (deprecated); parameter $\boldsymbol{\beta}$ of the input distribution. See `check_beta` on how to specify `beta` for each distribution.
`distname`	character; name of input distribution; see `get_distnames`.
`use.mean.variance`	logical; if `TRUE` it uses mean and variance implied by $\boldsymbol{\beta}$ to do the transformation (Goerg 2011). If `FALSE`, it uses the alternative definition from Goerg (2016) with location and scale parameter.
`p`	vector of probability levels
`n`	number of samples

Value

`dU` evaluates the pdf at `y`, `pU` evaluates the cdf, `qU` is the quantile function, and `rU` generates random samples from U.

Examples

```
# a zero-mean, unit variance version of the t_3 distribution.
curve(dU(x, beta = c(1, 1, 3), distname = "t"), -4, 4,
      ylab = "pdf", xlab = "u",
      main = "student-t \n zero-mean, unit variance")
# cdf of unit-variance version of an exp(3) -> just an exp(1)
curve(pU(x, beta = 3, distname = "exp"), 0, 4, ylab = "cdf", xlab = "u",
      main = "Exponential \n unit variance", col = 2, lwd = 2)
curve(pexp(x, rate = 1), 0, 4, add = TRUE, lty = 2)
# all have (empirical) variance 1
var(rU(n = 1000, distname = "chisq", beta = 2))
var(rU(n = 1000, distname = "normal", beta = c(3, 3)))
var(rU(n = 1000, distname = "exp", beta = 1))
var(rU(n = 1000, distname = "unif", beta = c(0, 10)))
```

`W` *Lambert W function, its logarithm and derivative*

Description

The Lambert W function $W(z) = u$ is defined as the inverse of (see `xexp`)

$$u \exp(u) = z,$$

i.e., it satisfies $W(z) \exp(W(z)) = z$.

`W` evaluates the Lambert W function (`W`), its first derivative (`deriv_W`), and its logarithm (`log_W`). All of them have a principal (`branch = 0` (default)) and non-principal branch (`branch = -1`) solution.

`W` is a wrapper for `lambertW0` and `lambertWm1` in the **lamW** package.

Usage

```
W(z, branch = 0)

deriv_W(z, branch = 0, W.z = W(z, branch = branch))

log_deriv_W(z, branch = 0, W.z = W(z, branch = branch))

deriv_log_W(z, branch = 0, W.z = W(z, branch = branch))

log_W(z, branch = 0, W.z = W(z, branch = branch))
```

Arguments

`z`	a numeric vector of real values; note that `W(Inf, branch = 0) = Inf`.
`branch`	either `0` or `-1` for the principal or non-principal branch solution.
`W.z`	Lambert W function evaluated at `z`; see Details below for why this is useful.

Details

Depending on the argument z of $W(z)$ one can distinguish 3 cases:

- $z \geq 0$ solution is unique `W(z) = W(z, branch = 0)`;
- $-1/e \leq z < 0$ two solutions: the principal (`W(z,branch = 0)`) and non-principal (`W(z, branch = -1)`) branch;
- $z < -1/e$ no solution exists in the reals.

`log_W` computes the natural logarithm of $W(z)$. This can be done efficiently since $\log W(z) = \log z - W(z)$. Similarly, the derivative can be expressed as a function of $W(z)$:

$$W'(z) = \frac{1}{(1 + W(z)) \exp(W(z))} = \frac{W(z)}{z(1 + W(z))}.$$

Note that $W'(0) = 1$ and $W'(-1/e) = \infty$.

Moreover, by taking logs on both sides we can even simplify further to

$$\log W'(z) = \log W(z) - \log z - \log(1 + W(z))$$

which, since $\log W(z) = \log z - W(z)$, simplifies to

$$\log W'(z) = -W(z) - \log(1 + W(z)).$$

For this reason it is numerically faster to pass the value of $W(z)$ as an argument to `deriv_W` since `W(z)` often has already been evaluated in a previous step.

Value

numeric; same dimensions/size as `z`.

`W` returns numeric, `Inf` (for `z = Inf`), or `NA` if $z < -1/e$.

Note that `W` handles `NaN` differently to `lambertW0` / `lambertWm1` in the **lamW** package; it returns `NA`.

References

Corless, R. M., G. H. Gonnet, D. E. G. Hare, D. J. Jeffrey and D. E. Knuth (1996). “On the Lambert W function”. Advances in Computational Mathematics, pp. 329-359.

See Also

`lambertW0` / `lambertWm1`in the **lamW** package; `xexp`.

Examples

```
W(-0.25) # "reasonable" input event
W(-0.25, branch = -1) # "extreme" input event

curve(W(x, branch = -1), -1, 2, type = "l", col = 2, lwd = 2)
curve(W(x), -1, 2, type = "l", add = TRUE, lty = 2)
abline(v = - 1 / exp(1))

# For lower values, the principal branch gives the 'wrong' solution;
# the non-principal must be used.
xexp(-10)
W(xexp(-10), branch = 0)
W(xexp(-10), branch = -1)

curve(log(x), 0.1, 5, lty = 2, col = 1, ylab = "")
curve(W(x), 0, 5, add = TRUE, col = "red")
curve(log_W(x), 0.1, 5, add = TRUE, col = "blue")
grid()
legend("bottomright", c("log(x)", "W(x)", "log(W(x))"),
       col = c("black", "red", "blue"), lty = c(2, 1, 1))
```

`W_delta` *Inverse transformation for heavy-tail Lambert W RVs*

Description

Inverse transformation `W_delta_alpha` for heavy-tail Lambert W RVs and its derivative. This is the inverse of Tukey's h transformation as a special case of `alpha = 1`.

Usage

```
W_delta(z, delta = 0)

W_delta_alpha(z, delta = 0, alpha = 1)

W_2delta(z, delta = c(0, 1/5))

W_2delta_2alpha(z, delta = c(0, 0), alpha = c(1, 1))

deriv_W_delta(z, delta = 0)

deriv_W_delta_alpha(z, delta = 1, alpha = 1)
```

Arguments

`z`	a numeric vector of real values.

`delta`	heavy-tail parameter(s); by default `delta = 0`, which implies `W_delta(z) = z`. If a vector of length 2 is supplied, then `delta[1]` on the left and `delta[2]` on the right (of the center) will be used.
`alpha`	heavy-tail exponent(s) in $(u^2)^\alpha$; default: `alpha = 1`.

Value

Computes $\mathrm{sgn}(z) \left(\frac{1}{\alpha\delta} W(\alpha\delta(z^2)^\alpha)\right)^{1/2\alpha}$. If z is a vector, so is the output.

Examples

```
G_delta(0)
W_delta(0)

# W_delta is the inverse of G_delta
u.v <- -2:2
W_delta(G_delta(u.v, delta = 0.3), delta = 0.3)

# with alpha too
G_delta_alpha(u.v, delta = 1, alpha = 0.33)
W_delta_alpha(G_delta_alpha(u.v, delta = 1, alpha = 0.33),
              delta = 1, alpha = 0.33) # the inverse
```

`W_gamma` *Inverse transformation for skewed Lambert W RVs*

Description

Inverse transformation for skewed Lambert W RVs and its derivative.

Usage

```
W_gamma(z, gamma = 0, branch = 0)

deriv_W_gamma(z, gamma = 0, branch = 0)
```

Arguments

`z`	a numeric vector of real values; note that `W(Inf, branch = 0) = Inf`.
`gamma`	skewness parameter; by default `gamma = 0`, which implies `W_gamma(z) = z`.
`branch`	either `0` or `-1` for the principal or non-principal branch solution.

Details

A skewed Lambert W× F RV Z (for simplicity assume zero mean, unit variance input) is defined by the transformation (see `H_gamma`)

$$z = U \exp(\gamma U) =: H_{\gamma}(U), \quad \gamma \in \mathbf{R},$$

where U is a zero-mean and/or unit-variance version of the distribution F.

The inverse transformation is $W_{\gamma}(z) := \frac{W(\gamma z)}{\gamma}$, where W is the Lambert W function.

`W_gamma(z, gamma, branch = 0)` (and `W_gamma(z, gamma, branch = -1)`) implement this inverse.

If $\gamma = 0$, then $z = u$ and the inverse also equals the identity.

If $\gamma \neq 0$, the inverse transformation can be computed by

$$W_{\gamma}(z) = \frac{1}{\gamma} W(\gamma z).$$

Same holds for `W_gamma(z, gamma, branch = -1)`.

The derivative of $W_{\gamma}(z)$ with respect to z simplifies to

$$\frac{d}{dz} W_{\gamma}(z) = \frac{1}{\gamma} \cdot W'(\gamma z) \cdot \gamma = W'(\gamma z)$$

`deriv_W_gamma` implements this derivative (for both branches).

Value

numeric; if z is a vector, so is the output.

See Also

`H_gamma`

`xexp` *Transformation that defines the Lambert W function and its derivative*

Description

The Lambert W function $W(z)$ is the inverse of $u \exp(u) = z$.

In versions < 0.6.0 of the package this function was denoted as `H`. It is now replaced with the more descriptive `xexp` (and `H` is deprecated).

Usage

```
xexp(x)

deriv_xexp(x, degree = 1)
```

Arguments

`x`	a numeric vector of real/complex values.
`degree`	non-negative integer; degree of the derivative

Details

The n-th derviative of $x \cdot \exp(x)$ is available in closed for as

$$\exp(x) \cdot (x + n).$$

Value

Returns $z = x \exp(x)$ for $x \in C$. If x is a vector/matrix, so is z.

See Also

`W`

Examples

```
plot(xexp, -5, 0.5, type="l", xlab="u", ylab="z")
grid()
abline(h=0, lty = 2)
abline(v=0, lty = 2)
```

Index