Package ‘gkwdist’

November 13, 2025

Title Generalized Kumaraswamy Distribution Family

Version 1.0.10

Description Implements the five-parameter Generalized Kumaraswamy ('gkw') distribution proposed by 'Carrasco, Ferrari and Cordeiro (2010)' <doi:10.48550/arXiv.1004.0911> and its seven nested sub-families for modeling bounded continuous data on the unit interval (0,1). The 'gkw' distribution extends the Kumaraswamy distribution described by Jones (2009) <doi:10.1016/j.stamet.2008.04.001>. Provides density, distribution, quantile, and random generation functions, along with analytical log-likelihood, gradient, and Hessian functions implemented in 'C++' via 'RcppArmadillo' for maximum computational efficiency. Suitable for modeling proportions, rates, percentages, and indices exhibiting complex features such as asymmetry, or heavy tails and other shapes not adequately captured by standard distributions like simple Beta or Kumaraswamy.

License MIT + file LICENSE

URL `https://github.com/evandeilton/gkwdist`, `https://evandeilton.github.io/gkwdist/`

BugReports `https://github.com/evandeilton/gkwdist/issues`

Encoding UTF-8

Language en-US

RoxygenNote 7.3.3

Imports Rcpp, RcppArmadillo, magrittr, numDeriv

LinkingTo Rcpp, RcppArmadillo

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

NeedsCompilation yes

VignetteBuilder knitr

Author José Evandeilton Lopes [aut, cre] (ORCID: <`https://orcid.org/0009-0007-5887-4084`>)

Maintainer José Evandeilton Lopes <`evandeilton@gmail.com`>

Repository CRAN

Date/Publication 2025-11-13 19:00:08 UTC

Contents

`dbeta_` *Density of the Beta Distribution (gamma, delta+1 Parameterization)*

Description

Computes the probability density function (PDF) for the standard Beta distribution, using a parameterization common in generalized distribution families. The distribution is parameterized by `gamma` (γ) and `delta` (δ), corresponding to the standard Beta distribution with shape parameters `shape1 = gamma` and `shape2 = delta + 1`. The distribution is defined on the interval (0, 1).

Usage

```
dbeta_(x, gamma, delta, log_prob = FALSE)
```

Arguments

`x`	Vector of quantiles (values between 0 and 1).
`gamma`	First shape parameter (`shape1`), $\gamma > 0$. Can be a scalar or a vector. Default: 1.0.
`delta`	Second shape parameter is `delta + 1` (`shape2`), requires $\delta \geq 0$ so that `shape2 >= 1`. Can be a scalar or a vector. Default: 0.0 (leading to `shape2 = 1`).
`log_prob`	Logical; if `TRUE`, the logarithm of the density is returned ($\log(f(x))$). Default: `FALSE`.

Details

The probability density function (PDF) calculated by this function corresponds to a standard Beta distribution $Beta(\gamma, \delta + 1)$:

$$f(x;\gamma,\delta) = \frac{x^{\gamma-1}(1-x)^{(\delta+1)-1}}{B(\gamma,\delta+1)} = \frac{x^{\gamma-1}(1-x)^{\delta}}{B(\gamma,\delta+1)}$$

for $0 < x < 1$, where $B(a, b)$ is the Beta function (`beta`).

This specific parameterization arises as a special case of the five-parameter Generalized Kumaraswamy (GKw) distribution (`dgkw`) obtained by setting the parameters $\alpha = 1$, $\beta = 1$, and $\lambda = 1$. It is therefore equivalent to the McDonald (Mc)/Beta Power distribution (`dmc`) with $\lambda = 1$.

Note the difference in the second parameter compared to `dbeta`, where `dbeta(x, shape1, shape2)` uses `shape2` directly. Here, `shape1 = gamma` and `shape2 = delta + 1`.

Value

A vector of density values ($f(x)$) or log-density values ($\log(f(x))$). The length of the result is determined by the recycling rule applied to the arguments (x, gamma, delta). Returns 0 (or -Inf if log_prob = TRUE) for x outside the interval (0, 1), or NaN if parameters are invalid (e.g., gamma <= 0, delta < 0).

Author(s)

Lopes, J. E.

References

Johnson, N. L., Kotz, S., & Balakrishnan, N. (1995). *Continuous Univariate Distributions, Volume 2* (2nd ed.). Wiley.

Cordeiro, G. M., & de Castro, M. (2011). A new family of generalized distributions. *Journal of Statistical Computation and Simulation*,

See Also

dbeta (standard R implementation), dgkw (parent distribution density), dmc (McDonald/Beta Power density), pbeta_, qbeta_, rbeta_ (other functions for this parameterization, if they exist).

Examples

```
# Example values
x_vals <- c(0.2, 0.5, 0.8)
gamma_par <- 2.0 # Corresponds to shape1
delta_par <- 3.0 # Corresponds to shape2 - 1
shape1 <- gamma_par
shape2 <- delta_par + 1

# Calculate density using dbeta_
densities <- dbeta_(x_vals, gamma_par, delta_par)
print(densities)

# Compare with stats::dbeta
densities_stats <- stats::dbeta(x_vals, shape1 = shape1, shape2 = shape2)
print(paste("Max difference vs stats::dbeta:", max(abs(densities - densities_stats))))

# Compare with dgkw setting alpha=1, beta=1, lambda=1
densities_gkw <- dgkw(x_vals, alpha = 1.0, beta = 1.0, gamma = gamma_par,
                      delta = delta_par, lambda = 1.0)
print(paste("Max difference vs dgkw:", max(abs(densities - densities_gkw))))

# Compare with dmc setting lambda=1
densities_mc <- dmc(x_vals, gamma = gamma_par, delta = delta_par, lambda = 1.0)
print(paste("Max difference vs dmc:", max(abs(densities - densities_mc))))

# Calculate log-density
log_densities <- dbeta_(x_vals, gamma_par, delta_par, log_prob = TRUE)
print(log_densities)
print(stats::dbeta(x_vals, shape1 = shape1, shape2 = shape2, log = TRUE))

# Plot the density
curve_x <- seq(0.001, 0.999, length.out = 200)
curve_y <- dbeta_(curve_x, gamma = 2, delta = 3) # Beta(2, 4)
plot(curve_x, curve_y, type = "l", main = "Beta(2, 4) Density via dbeta_",
     xlab = "x", ylab = "f(x)", col = "blue")
curve(stats::dbeta(x, 2, 4), add=TRUE, col="red", lty=2)
legend("topright", legend=c("dbeta_(gamma=2, delta=3)", "stats::dbeta(shape1=2, shape2=4)"),
       col=c("blue", "red"), lty=c(1,2), bty="n")
```

dbkw *Density of the Beta-Kumaraswamy (BKw) Distribution*

Description

Computes the probability density function (PDF) for the Beta-Kumaraswamy (BKw) distribution with parameters `alpha` (α), `beta` (β), `gamma` (γ), and `delta` (δ). This distribution is defined on the interval (0, 1).

Usage

```
dbkw(x, alpha, beta, gamma, delta, log_prob = FALSE)
```

Arguments

`x`	Vector of quantiles (values between 0 and 1).
`alpha`	Shape parameter `alpha` > 0. Can be a scalar or a vector. Default: 1.0.
`beta`	Shape parameter `beta` > 0. Can be a scalar or a vector. Default: 1.0.
`gamma`	Shape parameter `gamma` > 0. Can be a scalar or a vector. Default: 1.0.
`delta`	Shape parameter `delta` >= 0. Can be a scalar or a vector. Default: 0.0.
`log_prob`	Logical; if `TRUE`, the logarithm of the density is returned ($\log(f(x))$). Default: `FALSE`.

Details

The probability density function (PDF) of the Beta-Kumaraswamy (BKw) distribution is given by:

$$f(x;\alpha,\beta,\gamma,\delta) = \frac{\alpha\beta}{B(\gamma,\delta+1)} x^{\alpha-1} \bigl(1-x^\alpha\bigr)^{\beta(\delta+1)-1} \bigl[1-\bigl(1-x^\alpha\bigr)^\beta\bigr]^{\gamma-1}$$

for $0 < x < 1$, where $B(a,b)$ is the Beta function (`beta`).

The BKw distribution is a special case of the five-parameter Generalized Kumaraswamy (GKw) distribution (`dgkw`) obtained by setting the parameter $\lambda = 1$. Numerical evaluation is performed using algorithms similar to those for dgkw, ensuring stability.

Value

A vector of density values ($f(x)$) or log-density values ($\log(f(x))$). The length of the result is determined by the recycling rule applied to the arguments (`x`, `alpha`, `beta`, `gamma`, `delta`). Returns `0` (or `-Inf` if `log_prob = TRUE`) for `x` outside the interval (0, 1), or `NaN` if parameters are invalid (e.g., `alpha <= 0`, `beta <= 0`, `gamma <= 0`, `delta < 0`).

Author(s)

Lopes, J. E.

References

Cordeiro, G. M., & de Castro, M. (2011). A new family of generalized distributions. *Journal of Statistical Computation and Simulation*

Kumaraswamy, P. (1980). A generalized probability density function for double-bounded random processes. *Journal of Hydrology*, *46*(1-2), 79-88.

See Also

`dgkw` (parent distribution density), `pbkw`, `qbkw`, `rbkw` (other BKw functions),

Examples

```
# Example values
x_vals <- c(0.2, 0.5, 0.8)
alpha_par <- 2.0
beta_par <- 1.5
gamma_par <- 1.0 # Equivalent to Kw when gamma=1
delta_par <- 0.5

# Calculate density
densities <- dbkw(x_vals, alpha_par, beta_par, gamma_par, delta_par)
print(densities)

# Calculate log-density
log_densities <- dbkw(x_vals, alpha_par, beta_par, gamma_par, delta_par,
                      log_prob = TRUE)
print(log_densities)
# Check: should match log(densities)
print(log(densities))

# Compare with dgkw setting lambda = 1
densities_gkw <- dgkw(x_vals, alpha_par, beta_par, gamma = gamma_par,
                      delta = delta_par, lambda = 1.0)
print(paste("Max difference:", max(abs(densities - densities_gkw)))) # Should be near zero

# Plot the density for different gamma values
curve_x <- seq(0.01, 0.99, length.out = 200)
curve_y1 <- dbkw(curve_x, alpha = 2, beta = 3, gamma = 0.5, delta = 1)
curve_y2 <- dbkw(curve_x, alpha = 2, beta = 3, gamma = 1.0, delta = 1)
curve_y3 <- dbkw(curve_x, alpha = 2, beta = 3, gamma = 2.0, delta = 1)
```

```
plot(curve_x, curve_y1, type = "l", main = "BKw Density Examples (alpha=2, beta=3, delta=1)",
    xlab = "x", ylab = "f(x)", col = "blue", ylim = range(0, curve_y1, curve_y2, curve_y3))
lines(curve_x, curve_y2, col = "red")
lines(curve_x, curve_y3, col = "green")
legend("topright", legend = c("gamma=0.5", "gamma=1.0", "gamma=2.0"),
       col = c("blue", "red", "green"), lty = 1, bty = "n")
```

dekw *Density of the Exponentiated Kumaraswamy (EKw) Distribution*

Description

Computes the probability density function (PDF) for the Exponentiated Kumaraswamy (EKw) distribution with parameters `alpha` (α), `beta` (β), and `lambda` (λ). This distribution is defined on the interval (0, 1).

Usage

```
dekw(x, alpha, beta, lambda, log_prob = FALSE)
```

Arguments

`x`	Vector of quantiles (values between 0 and 1).
`alpha`	Shape parameter `alpha` > 0. Can be a scalar or a vector. Default: 1.0.
`beta`	Shape parameter `beta` > 0. Can be a scalar or a vector. Default: 1.0.
`lambda`	Shape parameter `lambda` > 0 (exponent parameter). Can be a scalar or a vector. Default: 1.0.
`log_prob`	Logical; if `TRUE`, the logarithm of the density is returned ($\log(f(x))$). Default: `FALSE`.

Details

The probability density function (PDF) of the Exponentiated Kumaraswamy (EKw) distribution is given by:

$$f(x;\alpha,\beta,\lambda) = \lambda\alpha\beta x^{\alpha-1}(1-x^\alpha)^{\beta-1}\bigl[1-(1-x^\alpha)^\beta\bigr]^{\lambda-1}$$

for $0 < x < 1$.

The EKw distribution is a special case of the five-parameter Generalized Kumaraswamy (GKw) distribution (`dgkw`) obtained by setting the parameters $\gamma = 1$ and $\delta = 0$. When $\lambda = 1$, the EKw distribution reduces to the standard Kumaraswamy distribution.

Value

A vector of density values ($f(x)$) or log-density values ($\log(f(x))$). The length of the result is determined by the recycling rule applied to the arguments (`x`, `alpha`, `beta`, `lambda`). Returns `0` (or `-Inf` if `log_prob = TRUE`) for `x` outside the interval (0, 1), or `NaN` if parameters are invalid (e.g., `alpha <= 0`, `beta <= 0`, `lambda <= 0`).

Author(s)

Lopes, J. E.

References

Nadarajah, S., Cordeiro, G. M., & Ortega, E. M. (2012). The exponentiated Kumaraswamy distribution. *Journal of the Franklin Institute*, *349*(3),

Cordeiro, G. M., & de Castro, M. (2011). A new family of generalized distributions. *Journal of Statistical Computation and Simulation*,

Kumaraswamy, P. (1980). A generalized probability density function for double-bounded random processes. *Journal of Hydrology*, *46*(1-2), 79-88.

See Also

`dgkw` (parent distribution density), `pekw`, `qekw`, `rekw` (other EKw functions),

Examples

```
# Example values
x_vals <- c(0.2, 0.5, 0.8)
alpha_par <- 2.0
beta_par <- 3.0
lambda_par <- 1.5 # Exponent parameter

# Calculate density
densities <- dekw(x_vals, alpha_par, beta_par, lambda_par)
print(densities)

# Calculate log-density
log_densities <- dekw(x_vals, alpha_par, beta_par, lambda_par, log_prob = TRUE)
print(log_densities)
# Check: should match log(densities)
print(log(densities))

# Compare with dgkw setting gamma = 1, delta = 0
densities_gkw <- dgkw(x_vals, alpha_par, beta_par, gamma = 1.0, delta = 0.0,
                      lambda = lambda_par)
print(paste("Max difference:", max(abs(densities - densities_gkw)))) # Should be near zero

# Plot the density for different lambda values
curve_x <- seq(0.01, 0.99, length.out = 200)
curve_y1 <- dekw(curve_x, alpha = 2, beta = 3, lambda = 0.5) # less peaked
curve_y2 <- dekw(curve_x, alpha = 2, beta = 3, lambda = 1.0) # standard Kw
```

```
curve_y3 <- dekw(curve_x, alpha = 2, beta = 3, lambda = 2.0) # more peaked

plot(curve_x, curve_y2, type = "l", main = "EKw Density Examples (alpha=2, beta=3)",
     xlab = "x", ylab = "f(x)", col = "red", ylim = range(0, curve_y1, curve_y2, curve_y3))
lines(curve_x, curve_y1, col = "blue")
lines(curve_x, curve_y3, col = "green")
legend("topright", legend = c("lambda=0.5", "lambda=1.0 (Kw)", "lambda=2.0"),
       col = c("blue", "red", "green"), lty = 1, bty = "n")
```

dgkw — *Density of the Generalized Kumaraswamy Distribution*

Description

Computes the probability density function (PDF) for the five-parameter Generalized Kumaraswamy (GKw) distribution, defined on the interval (0, 1).

Usage

```
dgkw(x, alpha, beta, gamma, delta, lambda, log_prob = FALSE)
```

Arguments

x	Vector of quantiles (values between 0 and 1).
alpha	Shape parameter `alpha` > 0. Can be a scalar or a vector. Default: 1.0.
beta	Shape parameter `beta` > 0. Can be a scalar or a vector. Default: 1.0.
gamma	Shape parameter `gamma` > 0. Can be a scalar or a vector. Default: 1.0.
delta	Shape parameter `delta` >= 0. Can be a scalar or a vector. Default: 0.0.
lambda	Shape parameter `lambda` > 0. Can be a scalar or a vector. Default: 1.0.
log_prob	Logical; if `TRUE`, the logarithm of the density is returned. Default: `FALSE`.

Details

The probability density function of the Generalized Kumaraswamy (GKw) distribution with parameters `alpha` (α), `beta` (β), `gamma` (γ), `delta` (δ), and `lambda` (λ) is given by:

$$f(x;\alpha,\beta,\gamma,\delta,\lambda) = \frac{\lambda\alpha\beta x^{\alpha-1}(1-x^\alpha)^{\beta-1}}{B(\gamma,\delta+1)}[1-(1-x^\alpha)^\beta]^{\gamma\lambda-1}[1-[1-(1-x^\alpha)^\beta]^\lambda]^\delta$$

for $x \in (0,1)$, where $B(a,b)$ is the Beta function `beta`.

This distribution was proposed by Cordeiro & de Castro (2011) and includes several other distributions as special cases:

- Kumaraswamy (Kw): `gamma = 1`, `delta = 0`, `lambda = 1`

- Exponentiated Kumaraswamy (EKw): `gamma = 1`, `delta = 0`
- Beta-Kumaraswamy (BKw): `lambda = 1`
- Generalized Beta type 1 (GB1 - implies McDonald): `alpha = 1`, `beta = 1`
- Beta distribution: `alpha = 1`, `beta = 1`, `lambda = 1`

The function includes checks for valid parameters and input values `x`. It uses numerical stabilization for `x` close to 0 or 1.

Value

A vector of density values ($f(x)$) or log-density values ($\log(f(x))$). The length of the result is determined by the recycling rule applied to the arguments (`x`, `alpha`, `beta`, `gamma`, `delta`, `lambda`). Returns `0` (or `-Inf` if `log_prob = TRUE`) for `x` outside the interval (0, 1), or `NaN` if parameters are invalid.

Author(s)

Lopes, J. E.

References

Cordeiro, G. M., & de Castro, M. (2011). A new family of generalized distributions. *Journal of Statistical Computation and Simulation*, *81*(7), 883-898.

Kumaraswamy, P. (1980). A generalized probability density function for double-bounded random processes. *Journal of Hydrology*, *46*(1-2), 79-88.

See Also

`pgkw`, `qgkw`, `rgkw` (if these exist), `dbeta`, `integrate`

Examples

```
# Simple density evaluation at a point
dgkw(0.5, alpha = 2, beta = 3, gamma = 1, delta = 0, lambda = 1) # Kw case

# Plot the PDF for various parameter sets
x_vals <- seq(0.01, 0.99, by = 0.01)

# Standard Kumaraswamy (gamma=1, delta=0, lambda=1)
pdf_kw <- dgkw(x_vals, alpha = 2, beta = 3, gamma = 1, delta = 0, lambda = 1)

# Beta equivalent (alpha=1, beta=1, lambda=1) - Beta(gamma, delta+1)
pdf_beta <- dgkw(x_vals, alpha = 1, beta = 1, gamma = 2, delta = 3, lambda = 1)
# Compare with stats::dbeta
pdf_beta_check <- stats::dbeta(x_vals, shape1 = 2, shape2 = 3 + 1)
# max(abs(pdf_beta - pdf_beta_check)) # Should be close to zero

# Exponentiated Kumaraswamy (gamma=1, delta=0)
pdf_ekw <- dgkw(x_vals, alpha = 2, beta = 3, gamma = 1, delta = 0, lambda = 2)
```

```
plot(x_vals, pdf_kw, type = "l", ylim = range(c(pdf_kw, pdf_beta, pdf_ekw)),
     main = "GKw Densities Examples", ylab = "f(x)", xlab="x", col = "blue")
lines(x_vals, pdf_beta, col = "red")
lines(x_vals, pdf_ekw, col = "green")
legend("topright", legend = c("Kw(2,3)", "Beta(2,4) equivalent", "EKw(2,3, lambda=2)"),
       col = c("blue", "red", "green"), lty = 1, bty = "n")

# Log-density
log_pdf_val <- dgkw(0.5, 2, 3, 1, 0, 1, log_prob = TRUE)
print(log_pdf_val)
print(log(dgkw(0.5, 2, 3, 1, 0, 1))) # Should match
```

dkkw *Density of the Kumaraswamy-Kumaraswamy (kkw) Distribution*

Description

Computes the probability density function (PDF) for the Kumaraswamy-Kumaraswamy (kkw) distribution with parameters `alpha` (α), `beta` (β), `delta` (δ), and `lambda` (λ). This distribution is defined on the interval (0, 1).

Usage

```
dkkw(x, alpha, beta, delta, lambda, log_prob = FALSE)
```

Arguments

`x`	Vector of quantiles (values between 0 and 1).
`alpha`	Shape parameter `alpha` > 0. Can be a scalar or a vector. Default: 1.0.
`beta`	Shape parameter `beta` > 0. Can be a scalar or a vector. Default: 1.0.
`delta`	Shape parameter `delta` >= 0. Can be a scalar or a vector. Default: 0.0.
`lambda`	Shape parameter `lambda` > 0. Can be a scalar or a vector. Default: 1.0.
`log_prob`	Logical; if `TRUE`, the logarithm of the density is returned ($\log(f(x))$). Default: `FALSE`.

Details

The Kumaraswamy-Kumaraswamy (kkw) distribution is a special case of the five-parameter Generalized Kumaraswamy distribution (`dgkw`) obtained by setting the parameter $\gamma = 1$.

The probability density function is given by:

$$f(x;\alpha,\beta,\delta,\lambda) = (\delta+1)\lambda\alpha\beta x^{\alpha-1}(1-x^{\alpha})^{\beta-1}\bigl[1-(1-x^{\alpha})^{\beta}\bigr]^{\lambda-1}\bigl\{1-\bigl[1-(1-x^{\alpha})^{\beta}\bigr]^{\lambda}\bigr\}^{\delta}$$

for $0 < x < 1$. Note that $1/(\delta+1)$ corresponds to the Beta function term $B(1,\delta+1)$ when $\gamma = 1$.

Numerical evaluation follows similar stability considerations as `dgkw`.

Value

A vector of density values ($f(x)$) or log-density values ($\log(f(x))$). The length of the result is determined by the recycling rule applied to the arguments (`x`, `alpha`, `beta`, `delta`, `lambda`). Returns `0` (or `-Inf` if `log_prob = TRUE`) for `x` outside the interval (0, 1), or `NaN` if parameters are invalid (e.g., `alpha <= 0`, `beta <= 0`, `delta < 0`, `lambda <= 0`).

Author(s)

Lopes, J. E.

References

Cordeiro, G. M., & de Castro, M. (2011). A new family of generalized distributions. *Journal of Statistical Computation and Simulation*

Kumaraswamy, P. (1980). A generalized probability density function for double-bounded random processes. *Journal of Hydrology*, *46*(1-2), 79-88.

See Also

`dgkw` (parent distribution density), `pkkw`, `qkkw`, `rkkw` (if they exist), `dbeta`

Examples

```
# Example values
x_vals <- c(0.2, 0.5, 0.8)
alpha_par <- 2.0
beta_par <- 3.0
delta_par <- 0.5
lambda_par <- 1.5

# Calculate density
densities <- dkkw(x_vals, alpha_par, beta_par, delta_par, lambda_par)
print(densities)

# Calculate log-density
log_densities <- dkkw(x_vals, alpha_par, beta_par, delta_par, lambda_par,
                       log_prob = TRUE)
print(log_densities)
# Check: should match log(densities)
print(log(densities))

# Compare with dgkw setting gamma = 1
densities_gkw <- dgkw(x_vals, alpha_par, beta_par, gamma = 1.0,
                      delta_par, lambda_par)
print(paste("Max difference:", max(abs(densities - densities_gkw)))) # Should be near zero

# Plot the density
curve_x <- seq(0.01, 0.99, length.out = 200)
curve_y <- dkkw(curve_x, alpha_par, beta_par, delta_par, lambda_par)
plot(curve_x, curve_y, type = "l", main = "kkw Density Example",
     xlab = "x", ylab = "f(x)", col = "blue")
```

`dkw` *Density of the Kumaraswamy (Kw) Distribution*

Description

Computes the probability density function (PDF) for the two-parameter Kumaraswamy (Kw) distribution with shape parameters `alpha` (α) and `beta` (β). This distribution is defined on the interval (0, 1).

Usage

```
dkw(x, alpha, beta, log_prob = FALSE)
```

Arguments

`x`	Vector of quantiles (values between 0 and 1).
`alpha`	Shape parameter `alpha` > 0. Can be a scalar or a vector. Default: 1.0.
`beta`	Shape parameter `beta` > 0. Can be a scalar or a vector. Default: 1.0.
`log_prob`	Logical; if `TRUE`, the logarithm of the density is returned ($\log(f(x))$). Default: `FALSE`.

Details

The probability density function (PDF) of the Kumaraswamy (Kw) distribution is given by:

$$f(x; \alpha, \beta) = \alpha\beta x^{\alpha-1}(1 - x^\alpha)^{\beta-1}$$

for $0 < x < 1$, $\alpha > 0$, and $\beta > 0$.

The Kumaraswamy distribution is identical to the Generalized Kumaraswamy (GKw) distribution (`dgkw`) with parameters $\gamma = 1$, $\delta = 0$, and $\lambda = 1$. It is also a special case of the Exponentiated Kumaraswamy (`dekw`) with $\lambda = 1$, and the Kumaraswamy-Kumaraswamy (`dkkw`) with $\delta = 0$ and $\lambda = 1$.

Value

A vector of density values ($f(x)$) or log-density values ($\log(f(x))$). The length of the result is determined by the recycling rule applied to the arguments (`x`, `alpha`, `beta`). Returns `0` (or `-Inf` if `log_prob = TRUE`) for `x` outside the interval (0, 1), or `NaN` if parameters are invalid (e.g., `alpha <= 0`, `beta <= 0`).

Author(s)

Lopes, J. E.

References

Kumaraswamy, P. (1980). A generalized probability density function for double-bounded random processes. *Journal of Hydrology*, *46*(1-2), 79-88.

Jones, M. C. (2009). Kumaraswamy's distribution: A beta-type distribution with some tractability advantages. *Statistical Methodology*, *6*(1), 70-81.

See Also

`dgkw` (parent distribution density), `dekw`, `dkkw`, `pkw`, `qkw`, `rkw` (other Kw functions), `dbeta`

Examples

```
# Example values
x_vals <- c(0.2, 0.5, 0.8)
alpha_par <- 2.0
beta_par <- 3.0

# Calculate density using dkw
densities <- dkw(x_vals, alpha_par, beta_par)
print(densities)

# Calculate log-density
log_densities <- dkw(x_vals, alpha_par, beta_par, log_prob = TRUE)
print(log_densities)
# Check: should match log(densities)
print(log(densities))

# Compare with dgkw setting gamma = 1, delta = 0, lambda = 1
densities_gkw <- dgkw(x_vals, alpha_par, beta_par, gamma = 1.0, delta = 0.0,
                      lambda = 1.0)
print(paste("Max difference:", max(abs(densities - densities_gkw)))) # Should be near zero

# Plot the density for different shape parameter combinations
curve_x <- seq(0.001, 0.999, length.out = 200)
plot(curve_x, dkw(curve_x, alpha = 2, beta = 3), type = "l",
     main = "Kumaraswamy Density Examples", xlab = "x", ylab = "f(x)",
     col = "blue", ylim = c(0, 4))
lines(curve_x, dkw(curve_x, alpha = 3, beta = 2), col = "red")
lines(curve_x, dkw(curve_x, alpha = 0.5, beta = 0.5), col = "green") # U-shaped
lines(curve_x, dkw(curve_x, alpha = 5, beta = 1), col = "purple") # J-shaped
lines(curve_x, dkw(curve_x, alpha = 1, beta = 3), col = "orange") # J-shaped (reversed)
legend("top", legend = c("a=2, b=3", "a=3, b=2", "a=0.5, b=0.5", "a=5, b=1", "a=1, b=3"),
       col = c("blue", "red", "green", "purple", "orange"), lty = 1, bty = "n", ncol = 2)
```

dmc *Density of the McDonald (Mc)/Beta Power Distribution Distribution*

Description

Computes the probability density function (PDF) for the McDonald (Mc) distribution (also previously referred to as Beta Power) with parameters `gamma` (γ), `delta` (δ), and `lambda` (λ). This distribution is defined on the interval (0, 1).

Usage

```
dmc(x, gamma, delta, lambda, log_prob = FALSE)
```

Arguments

`x`	Vector of quantiles (values between 0 and 1).
`gamma`	Shape parameter `gamma` > 0. Can be a scalar or a vector. Default: 1.0.
`delta`	Shape parameter `delta` >= 0. Can be a scalar or a vector. Default: 0.0.
`lambda`	Shape parameter `lambda` > 0. Can be a scalar or a vector. Default: 1.0.
`log_prob`	Logical; if `TRUE`, the logarithm of the density is returned ($\log(f(x))$). Default: `FALSE`.

Details

The probability density function (PDF) of the McDonald (Mc) distribution is given by:

$$f(x; \gamma, \delta, \lambda) = \frac{\lambda}{B(\gamma, \delta+1)} x^{\gamma\lambda - 1} (1 - x^\lambda)^\delta$$

for $0 < x < 1$, where $B(a, b)$ is the Beta function (`beta`).

The Mc distribution is a special case of the five-parameter Generalized Kumaraswamy (GKw) distribution (`dgkw`) obtained by setting the parameters $\alpha = 1$ and $\beta = 1$. It was introduced by McDonald (1984) and is related to the Generalized Beta distribution of the first kind (GB1). When $\lambda = 1$, it simplifies to the standard Beta distribution with parameters γ and $\delta + 1$.

Value

A vector of density values ($f(x)$) or log-density values ($\log(f(x))$). The length of the result is determined by the recycling rule applied to the arguments (`x`, `gamma`, `delta`, `lambda`). Returns `0` (or `-Inf` if `log_prob = TRUE`) for `x` outside the interval (0, 1), or `NaN` if parameters are invalid (e.g., `gamma <= 0`, `delta < 0`, `lambda <= 0`).

Author(s)

Lopes, J. E.

References

McDonald, J. B. (1984). Some generalized functions for the size distribution of income. *Econometrica*, 52(3), 647-663.

Cordeiro, G. M., & de Castro, M. (2011). A new family of generalized distributions. *Journal of Statistical Computation and Simulation*,

Kumaraswamy, P. (1980). A generalized probability density function for double-bounded random processes. *Journal of Hydrology*, *46*(1-2), 79-88.

See Also

`dgkw` (parent distribution density), `pmc`, `qmc`, `rmc` (other Mc functions), `dbeta`

Examples

```
# Example values
x_vals <- c(0.2, 0.5, 0.8)
gamma_par <- 2.0
delta_par <- 1.5
lambda_par <- 1.0 # Equivalent to Beta(gamma, delta+1)

# Calculate density using dmc
densities <- dmc(x_vals, gamma_par, delta_par, lambda_par)
print(densities)
# Compare with Beta density
print(stats::dbeta(x_vals, shape1 = gamma_par, shape2 = delta_par + 1))

# Calculate log-density
log_densities <- dmc(x_vals, gamma_par, delta_par, lambda_par, log_prob = TRUE)
print(log_densities)

# Compare with dgkw setting alpha = 1, beta = 1
densities_gkw <- dgkw(x_vals, alpha = 1.0, beta = 1.0, gamma = gamma_par,
                      delta = delta_par, lambda = lambda_par)
print(paste("Max difference:", max(abs(densities - densities_gkw)))) # Should be near zero

# Plot the density for different lambda values
curve_x <- seq(0.01, 0.99, length.out = 200)
curve_y1 <- dmc(curve_x, gamma = 2, delta = 3, lambda = 0.5)
curve_y2 <- dmc(curve_x, gamma = 2, delta = 3, lambda = 1.0) # Beta(2, 4)
curve_y3 <- dmc(curve_x, gamma = 2, delta = 3, lambda = 2.0)

plot(curve_x, curve_y2, type = "l", main = "McDonald (Mc) Density (gamma=2, delta=3)",
    xlab = "x", ylab = "f(x)", col = "red", ylim = range(0, curve_y1, curve_y2, curve_y3))
lines(curve_x, curve_y1, col = "blue")
lines(curve_x, curve_y3, col = "green")
legend("topright", legend = c("lambda=0.5", "lambda=1.0 (Beta)", "lambda=2.0"),
       col = c("blue", "red", "green"), lty = 1, bty = "n")
```

gkwgetstartvalues *Estimate Distribution Parameters Using Method of Moments*

Description

Estimates parameters for various distribution families from the Generalized Kumaraswamy family using the method of moments. The implementation is optimized for numerical stability and computational efficiency through Nelder-Mead optimization and adaptive numerical integration.

Usage

```
gkwgetstartvalues(x, family = "gkw", n_starts = 5L)
```

Arguments

`x`	Numeric vector of observations. All values must be in the open interval (0,1). Values outside this range will be automatically truncated to avoid numerical issues.
`family`	Character string specifying the distribution family. Valid options are: `"gkw"` (Generalized Kumaraswamy - 5 parameters), `"bkw"` (Beta-Kumaraswamy - 4 parameters), `"kkw"` (Kumaraswamy-Kumaraswamy - 4 parameters), `"ekw"` (Exponentiated Kumaraswamy - 3 parameters), `"mc"` (McDonald - 3 parameters), `"kw"` (Kumaraswamy - 2 parameters), `"beta"` (Beta - 2 parameters). The string is case-insensitive. Default is `"gkw"`.
`n_starts`	Integer specifying the number of different initial parameter values to try during optimization. More starting points increase the probability of finding the global optimum at the cost of longer computation time. Default is 5.

Details

The function uses the method of moments to estimate distribution parameters by minimizing the weighted sum of squared relative errors between theoretical and sample moments (orders 1 through 5). The optimization employs the Nelder-Mead simplex algorithm, which is derivative-free and particularly robust for this problem.

Key implementation features: logarithmic calculations for numerical stability, adaptive numerical integration using Simpson's rule with fallback to trapezoidal rule, multiple random starting points to avoid local minima, decreasing weights for higher-order moments (1.0, 0.8, 0.6, 0.4, 0.2), and automatic parameter constraint enforcement.

Parameter Constraints: All parameters are constrained to positive values. Additionally, family-specific constraints are enforced: alpha and beta in (0.1, 50.0), gamma in (0.1, 10.0) for GKw-related families or (0.1, 50.0) for Beta, delta in (0.01, 10.0), and lambda in (0.1, 20.0).

The function will issue warnings for empty input vectors, sample sizes less than 10 (unreliable estimation), or failure to find valid parameter estimates (returns defaults).

Value

Named numeric vector containing the estimated parameters for the specified distribution family. Parameter names correspond to the distribution specification. If estimation fails, returns a vector of NA values with appropriate parameter names.

References

Jones, M. C. (2009). Kumaraswamy's distribution: A beta-type distribution with some tractability advantages. Statistical Methodology, 6(1), 70-81.

Examples

```
# Generate sample data from Beta distribution
set.seed(123)
x <- rbeta(100, shape1 = 2, shape2 = 3)

# Estimate Beta parameters
params_beta <- gkwgetstartvalues(x, family = "beta")
print(params_beta)

# Estimate Kumaraswamy parameters
params_kw <- gkwgetstartvalues(x, family = "kw")
print(params_kw)

# Estimate GKw parameters with more starting points
params_gkw <- gkwgetstartvalues(x, family = "gkw", n_starts = 10)
print(params_gkw)
```

`grbeta` *Gradient of the Negative Log-Likelihood for the Beta Distribution (gamma, delta+1 Parameterization)*

Description

Computes the gradient vector (vector of first partial derivatives) of the negative log-likelihood function for the standard Beta distribution, using a parameterization common in generalized distribution families. The distribution is parameterized by `gamma` (γ) and `delta` (δ), corresponding to the standard Beta distribution with shape parameters `shape1 = gamma` and `shape2 = delta + 1`. The gradient is useful for optimization algorithms.

Usage

```
grbeta(par, data)
```

Arguments

`par`	A numeric vector of length 2 containing the distribution parameters in the order: `gamma` ($\gamma > 0$), `delta` ($\delta \geq 0$).
`data`	A numeric vector of observations. All values must be strictly between 0 and 1 (exclusive).

Details

This function calculates the gradient of the negative log-likelihood for a Beta distribution with parameters `shape1` = `gamma` (γ) and `shape2` = `delta + 1` ($\delta + 1$). The components of the gradient vector ($-\nabla \ell(\theta|\mathbf{x})$) are:

$$-\frac{\partial \ell}{\partial \gamma} = n[\psi(\gamma) - \psi(\gamma + \delta + 1)] - \sum_{i=1}^{n} \ln(x_i)$$

$$-\frac{\partial \ell}{\partial \delta} = n[\psi(\delta + 1) - \psi(\gamma + \delta + 1)] - \sum_{i=1}^{n} \ln(1 - x_i)$$

where $\psi(\cdot)$ is the digamma function (`digamma`). These formulas represent the derivatives of $-\ell(\theta)$, consistent with minimizing the negative log-likelihood. They correspond to the relevant components of the general GKw gradient (`grgkw`) evaluated at $\alpha = 1, \beta = 1, \lambda = 1$. Note the parameterization: the standard Beta shape parameters are γ and $\delta + 1$.

Value

Returns a numeric vector of length 2 containing the partial derivatives of the negative log-likelihood function $-\ell(\theta|\mathbf{x})$ with respect to each parameter: $(-\partial \ell/\partial \gamma, -\partial \ell/\partial \delta)$. Returns a vector of `NaN` if any parameter values are invalid according to their constraints, or if any value in `data` is not in the interval (0, 1).

Author(s)

Lopes, J. E.

References

Johnson, N. L., Kotz, S., & Balakrishnan, N. (1995). *Continuous Univariate Distributions, Volume 2* (2nd ed.). Wiley.

Cordeiro, G. M., & de Castro, M. (2011). A new family of generalized distributions. *Journal of Statistical Computation and Simulation*,

(Note: Specific gradient formulas might be derived or sourced from additional references).

See Also

`grgkw`, `grmc` (related gradients), `llbeta` (negative log-likelihood function), `hsbeta` (Hessian, if available), `dbeta_`, `pbeta_`, `qbeta_`, `rbeta_`, `optim`, `grad` (for numerical gradient comparison), `digamma`.

Examples

```
## Example 1: Basic Gradient Evaluation

# Generate sample data
set.seed(123)
n <- 1000
true_params <- c(gamma = 2.0, delta = 3.0)
data <- rbeta_(n, gamma = true_params[1], delta = true_params[2])

# Evaluate gradient at true parameters
grad_true <- grbeta(par = true_params, data = data)
cat("Gradient at true parameters:\n")
print(grad_true)
cat("Norm:", sqrt(sum(grad_true^2)), "\n")

# Evaluate at different parameter values
test_params <- rbind(
  c(1.5, 2.5),
  c(2.0, 3.0),
  c(2.5, 3.5)
)

grad_norms <- apply(test_params, 1, function(p) {
  g <- grbeta(p, data)
  sqrt(sum(g^2))
})

results <- data.frame(
  Gamma = test_params[, 1],
  Delta = test_params[, 2],
  Grad_Norm = grad_norms
)
print(results, digits = 4)


## Example 2: Gradient in Optimization

# Optimization with analytical gradient
fit_with_grad <- optim(
  par = c(1.5, 2.5),
  fn = llbeta,
  gr = grbeta,
  data = data,
  method = "L-BFGS-B",
  lower = c(0.01, 0.01),
  upper = c(100, 100),
  hessian = TRUE,
  control = list(trace = 0)
)

# Optimization without gradient
fit_no_grad <- optim(
  par = c(1.5, 2.5),
  fn = llbeta,
  data = data,
  method = "L-BFGS-B",
  lower = c(0.01, 0.01),
  upper = c(100, 100),
  hessian = TRUE,
  control = list(trace = 0)
)

comparison <- data.frame(
  Method = c("With Gradient", "Without Gradient"),
  Gamma = c(fit_with_grad$par[1], fit_no_grad$par[1]),
  Delta = c(fit_with_grad$par[2], fit_no_grad$par[2]),
  NegLogLik = c(fit_with_grad$value, fit_no_grad$value),
  Iterations = c(fit_with_grad$counts[1], fit_no_grad$counts[1])
)
print(comparison, digits = 4, row.names = FALSE)


## Example 3: Verifying Gradient at MLE

mle <- fit_with_grad$par
names(mle) <- c("gamma", "delta")

# At MLE, gradient should be approximately zero
gradient_at_mle <- grbeta(par = mle, data = data)
cat("\nGradient at MLE:\n")
print(gradient_at_mle)
cat("Max absolute component:", max(abs(gradient_at_mle)), "\n")
cat("Gradient norm:", sqrt(sum(gradient_at_mle^2)), "\n")


## Example 4: Numerical vs Analytical Gradient

# Manual finite difference gradient
numerical_gradient <- function(f, x, data, h = 1e-7) {
  grad <- numeric(length(x))
  for (i in seq_along(x)) {
    x_plus <- x_minus <- x
    x_plus[i] <- x[i] + h
    x_minus[i] <- x[i] - h
    grad[i] <- (f(x_plus, data) - f(x_minus, data)) / (2 * h)
  }
  return(grad)
}

# Compare at MLE
grad_analytical <- grbeta(par = mle, data = data)
grad_numerical <- numerical_gradient(llbeta, mle, data)

comparison_grad <- data.frame(
  Parameter = c("gamma", "delta"),
  Analytical = grad_analytical,
  Numerical = grad_numerical,
  Abs_Diff = abs(grad_analytical - grad_numerical),
  Rel_Error = abs(grad_analytical - grad_numerical) /
              (abs(grad_analytical) + 1e-10)
)
print(comparison_grad, digits = 8)


## Example 5: Score Test Statistic

# Score test for H0: theta = theta0
theta0 <- c(1.8, 2.8)
score_theta0 <- -grbeta(par = theta0, data = data)

# Fisher information at theta0
fisher_info <- hsbeta(par = theta0, data = data)

# Score test statistic
score_stat <- t(score_theta0) %*% solve(fisher_info) %*% score_theta0
p_value <- pchisq(score_stat, df = 2, lower.tail = FALSE)

cat("\nScore Test:\n")
cat("H0: gamma=1.8, delta=2.8\n")
cat("Test statistic:", score_stat, "\n")
cat("P-value:", format.pval(p_value, digits = 4), "\n")


## Example 6: Confidence Ellipse (Gamma vs Delta)

# Observed information
obs_info <- hsbeta(par = mle, data = data)
vcov_full <- solve(obs_info)

# Create confidence ellipse
theta <- seq(0, 2 * pi, length.out = 100)
chi2_val <- qchisq(0.95, df = 2)

eig_decomp <- eigen(vcov_full)
ellipse <- matrix(NA, nrow = 100, ncol = 2)
for (i in 1:100) {
  v <- c(cos(theta[i]), sin(theta[i]))
  ellipse[i, ] <- mle + sqrt(chi2_val) *
    (eig_decomp$vectors %*% diag(sqrt(eig_decomp$values)) %*% v)
}

# Marginal confidence intervals
se_2d <- sqrt(diag(vcov_full))
ci_gamma <- mle[1] + c(-1, 1) * 1.96 * se_2d[1]
ci_delta <- mle[2] + c(-1, 1) * 1.96 * se_2d[2]

# Plot
```

```
plot(ellipse[, 1], ellipse[, 2], type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(gamma), ylab = expression(delta),
     main = "95% Confidence Region (Gamma vs Delta)", las = 1)

# Add marginal CIs
abline(v = ci_gamma, col = "#808080", lty = 3, lwd = 1.5)
abline(h = ci_delta, col = "#808080", lty = 3, lwd = 1.5)

points(mle[1], mle[2], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[2], pch = 17, col = "#006400", cex = 1.5)

legend("topright",
       legend = c("MLE", "True", "95% CR", "Marginal 95% CI"),
       col = c("#8B0000", "#006400", "#2E4057", "#808080"),
       pch = c(19, 17, NA, NA),
       lty = c(NA, NA, 1, 3),
       lwd = c(NA, NA, 2, 1.5),
       bty = "n")
grid(col = "gray90")
```

grbkw — *Gradient of the Negative Log-Likelihood for the BKw Distribution*

Description

Computes the gradient vector (vector of first partial derivatives) of the negative log-likelihood function for the Beta-Kumaraswamy (BKw) distribution with parameters `alpha` (α), `beta` (β), `gamma` (γ), and `delta` (δ). This distribution is the special case of the Generalized Kumaraswamy (GKw) distribution where $\lambda = 1$. The gradient is typically used in optimization algorithms for maximum likelihood estimation.

Usage

```
grbkw(par, data)
```

Arguments

`par`	A numeric vector of length 4 containing the distribution parameters in the order: `alpha` ($\alpha > 0$), `beta` ($\beta > 0$), `gamma` ($\gamma > 0$), `delta` ($\delta \geq 0$).
`data`	A numeric vector of observations. All values must be strictly between 0 and 1 (exclusive).

Details

The components of the gradient vector of the negative log-likelihood ($-\nabla\ell(\theta|\mathbf{x})$) for the BKw ($\lambda = 1$) model are:

$$-\frac{\partial\ell}{\partial\alpha} = -\frac{n}{\alpha} - \sum_{i=1}^{n}\ln(x_i) + \sum_{i=1}^{n}\left[x_i^{\alpha}\ln(x_i)\left(\frac{\beta(\delta+1)-1}{v_i} - \frac{(\gamma-1)\beta v_i^{\beta-1}}{w_i}\right)\right]$$

$$-\frac{\partial\ell}{\partial\beta} = -\frac{n}{\beta} - (\delta+1)\sum_{i=1}^{n}\ln(v_i) + \sum_{i=1}^{n}\left[\frac{(\gamma-1)v_i^{\beta}\ln(v_i)}{w_i}\right]$$

$$-\frac{\partial\ell}{\partial\gamma} = n[\psi(\gamma) - \psi(\gamma+\delta+1)] - \sum_{i=1}^{n}\ln(w_i)$$

$$-\frac{\partial\ell}{\partial\delta} = n[\psi(\delta+1) - \psi(\gamma+\delta+1)] - \beta\sum_{i=1}^{n}\ln(v_i)$$

where:

- $v_i = 1 - x_i^{\alpha}$
- $w_i = 1 - v_i^{\beta} = 1 - (1 - x_i^{\alpha})^{\beta}$
- $\psi(\cdot)$ is the digamma function (`digamma`).

These formulas represent the derivatives of $-\ell(\theta)$, consistent with minimizing the negative log-likelihood. They correspond to the general GKw gradient (`grgkw`) components for $\alpha, \beta, \gamma, \delta$ evaluated at $\lambda = 1$. Note that the component for λ is omitted. Numerical stability is maintained through careful implementation.

Value

Returns a numeric vector of length 4 containing the partial derivatives of the negative log-likelihood function $-\ell(\theta|\mathbf{x})$ with respect to each parameter: $(-\partial\ell/\partial\alpha, -\partial\ell/\partial\beta, -\partial\ell/\partial\gamma, -\partial\ell/\partial\delta)$. Returns a vector of `NaN` if any parameter values are invalid according to their constraints, or if any value in `data` is not in the interval (0, 1).

Author(s)

Lopes, J. E.

References

Cordeiro, G. M., & de Castro, M. (2011). A new family of generalized distributions. *Journal of Statistical Computation and Simulation*,

Kumaraswamy, P. (1980). A generalized probability density function for double-bounded random processes. *Journal of Hydrology*, *46*(1-2), 79-88.

(Note: Specific gradient formulas might be derived or sourced from additional references).

See Also

grgkw (parent distribution gradient), llbkw (negative log-likelihood for BKw), hsbkw (Hessian for BKw, if available), dbkw (density for BKw), optim, grad (for numerical gradient comparison), digamma.

Examples

```
## Example 1: Basic Gradient Evaluation
# Generate sample data
set.seed(2203)
n <- 1000
true_params <- c(alpha = 2.0, beta = 1.5, gamma = 1.5, delta = 0.5)
data <- rbkw(n, alpha = true_params[1], beta = true_params[2],
             gamma = true_params[3], delta = true_params[4])

# Evaluate gradient at true parameters
grad_true <- grbkw(par = true_params, data = data)
cat("Gradient at true parameters:\n")
print(grad_true)
cat("Norm:", sqrt(sum(grad_true^2)), "\n")

# Evaluate at different parameter values
test_params <- rbind(
  c(1.5, 1.0, 1.0, 0.3),
  c(2.0, 1.5, 1.5, 0.5),
  c(2.5, 2.0, 2.0, 0.7)
)

grad_norms <- apply(test_params, 1, function(p) {
  g <- grbkw(p, data)
  sqrt(sum(g^2))
})

results <- data.frame(
  Alpha = test_params[, 1],
  Beta = test_params[, 2],
  Gamma = test_params[, 3],
  Delta = test_params[, 4],
  Grad_Norm = grad_norms
)
print(results, digits = 4)


## Example 2: Gradient in Optimization

# Optimization with analytical gradient
fit_with_grad <- optim(
  par = c(1.8, 1.2, 1.1, 0.3),
  fn = llbkw,
  gr = grbkw,
  data = data,
  method = "Nelder-Mead",
  hessian = TRUE,
  control = list(trace = 0)
)

# Optimization without gradient
fit_no_grad <- optim(
  par = c(1.8, 1.2, 1.1, 0.3),
  fn = llbkw,
  data = data,
  method = "Nelder-Mead",
  hessian = TRUE,
  control = list(trace = 0)
)

comparison <- data.frame(
  Method = c("With Gradient", "Without Gradient"),
  Alpha = c(fit_with_grad$par[1], fit_no_grad$par[1]),
  Beta = c(fit_with_grad$par[2], fit_no_grad$par[2]),
  Gamma = c(fit_with_grad$par[3], fit_no_grad$par[3]),
  Delta = c(fit_with_grad$par[4], fit_no_grad$par[4]),
  NegLogLik = c(fit_with_grad$value, fit_no_grad$value),
  Iterations = c(fit_with_grad$counts[1], fit_no_grad$counts[1])
)
print(comparison, digits = 4, row.names = FALSE)


## Example 3: Verifying Gradient at MLE

mle <- fit_with_grad$par
names(mle) <- c("alpha", "beta", "gamma", "delta")

# At MLE, gradient should be approximately zero
gradient_at_mle <- grbkw(par = mle, data = data)
cat("\nGradient at MLE:\n")
print(gradient_at_mle)
cat("Max absolute component:", max(abs(gradient_at_mle)), "\n")
cat("Gradient norm:", sqrt(sum(gradient_at_mle^2)), "\n")


## Example 4: Numerical vs Analytical Gradient

# Manual finite difference gradient
numerical_gradient <- function(f, x, data, h = 1e-7) {
  grad <- numeric(length(x))
  for (i in seq_along(x)) {
    x_plus <- x_minus <- x
    x_plus[i] <- x[i] + h
    x_minus[i] <- x[i] - h
    grad[i] <- (f(x_plus, data) - f(x_minus, data)) / (2 * h)
  }
  return(grad)
}
```

```
# Compare at MLE
grad_analytical <- grbkw(par = mle, data = data)
grad_numerical <- numerical_gradient(llbkw, mle, data)

comparison_grad <- data.frame(
  Parameter = c("alpha", "beta", "gamma", "delta"),
  Analytical = grad_analytical,
  Numerical = grad_numerical,
  Abs_Diff = abs(grad_analytical - grad_numerical),
  Rel_Error = abs(grad_analytical - grad_numerical) /
              (abs(grad_analytical) + 1e-10)
)
print(comparison_grad, digits = 8)


## Example 5: Score Test Statistic

# Score test for H0: theta = theta0
theta0 <- c(1.8, 1.3, 1.2, 0.4)
score_theta0 <- -grbkw(par = theta0, data = data)

# Fisher information at theta0
fisher_info <- hsbkw(par = theta0, data = data)

# Score test statistic
score_stat <- t(score_theta0) %*% solve(fisher_info) %*% score_theta0
p_value <- pchisq(score_stat, df = 4, lower.tail = FALSE)

cat("\nScore Test:\n")
cat("H0: alpha=1.8, beta=1.3, gamma=1.2, delta=0.4\n")
cat("Test statistic:", score_stat, "\n")
cat("P-value:", format.pval(p_value, digits = 4), "\n")


## Example 6: Confidence Ellipses (Selected pairs)

# Observed information
obs_info <- hsbkw(par = mle, data = data)
vcov_full <- solve(obs_info)

# Create confidence ellipses
theta <- seq(0, 2 * pi, length.out = 100)
chi2_val <- qchisq(0.95, df = 2)

# Alpha vs Beta ellipse
vcov_ab <- vcov_full[1:2, 1:2]
eig_decomp_ab <- eigen(vcov_ab)
ellipse_ab <- matrix(NA, nrow = 100, ncol = 2)
for (i in 1:100) {
  v <- c(cos(theta[i]), sin(theta[i]))
  ellipse_ab[i, ] <- mle[1:2] + sqrt(chi2_val) *
    (eig_decomp_ab$vectors %*% diag(sqrt(eig_decomp_ab$values)) %*% v)
}
```

```
# Alpha vs Gamma ellipse
vcov_ag <- vcov_full[c(1, 3), c(1, 3)]
eig_decomp_ag <- eigen(vcov_ag)
ellipse_ag <- matrix(NA, nrow = 100, ncol = 2)
for (i in 1:100) {
  v <- c(cos(theta[i]), sin(theta[i]))
  ellipse_ag[i, ] <- mle[c(1, 3)] + sqrt(chi2_val) *
    (eig_decomp_ag$vectors %*% diag(sqrt(eig_decomp_ag$values)) %*% v)
}

# Beta vs Delta ellipse
vcov_bd <- vcov_full[c(2, 4), c(2, 4)]
eig_decomp_bd <- eigen(vcov_bd)
ellipse_bd <- matrix(NA, nrow = 100, ncol = 2)
for (i in 1:100) {
  v <- c(cos(theta[i]), sin(theta[i]))
  ellipse_bd[i, ] <- mle[c(2, 4)] + sqrt(chi2_val) *
    (eig_decomp_bd$vectors %*% diag(sqrt(eig_decomp_bd$values)) %*% v)
}

# Marginal confidence intervals
se_ab <- sqrt(diag(vcov_ab))
ci_alpha_ab <- mle[1] + c(-1, 1) * 1.96 * se_ab[1]
ci_beta_ab <- mle[2] + c(-1, 1) * 1.96 * se_ab[2]

se_ag <- sqrt(diag(vcov_ag))
ci_alpha_ag <- mle[1] + c(-1, 1) * 1.96 * se_ag[1]
ci_gamma_ag <- mle[3] + c(-1, 1) * 1.96 * se_ag[2]

se_bd <- sqrt(diag(vcov_bd))
ci_beta_bd <- mle[2] + c(-1, 1) * 1.96 * se_bd[1]
ci_delta_bd <- mle[4] + c(-1, 1) * 1.96 * se_bd[2]

# Plot selected ellipses

# Alpha vs Beta
plot(ellipse_ab[, 1], ellipse_ab[, 2], type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(alpha), ylab = expression(beta),
     main = "Alpha vs Beta", las = 1, xlim = range(ellipse_ab[, 1], ci_alpha_ab),
     ylim = range(ellipse_ab[, 2], ci_beta_ab))
abline(v = ci_alpha_ab, col = "#808080", lty = 3, lwd = 1.5)
abline(h = ci_beta_ab, col = "#808080", lty = 3, lwd = 1.5)
points(mle[1], mle[2], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[2], pch = 17, col = "#006400", cex = 1.5)
grid(col = "gray90")

# Alpha vs Gamma
plot(ellipse_ag[, 1], ellipse_ag[, 2], type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(alpha), ylab = expression(gamma),
     main = "Alpha vs Gamma", las = 1, xlim = range(ellipse_ag[, 1], ci_alpha_ag),
     ylim = range(ellipse_ag[, 2], ci_gamma_ag))
abline(v = ci_alpha_ag, col = "#808080", lty = 3, lwd = 1.5)
abline(h = ci_gamma_ag, col = "#808080", lty = 3, lwd = 1.5)
points(mle[1], mle[3], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[3], pch = 17, col = "#006400", cex = 1.5)
grid(col = "gray90")

# Beta vs Delta
plot(ellipse_bd[, 1], ellipse_bd[, 2], type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(beta), ylab = expression(delta),
     main = "Beta vs Delta", las = 1, xlim = range(ellipse_bd[, 1], ci_beta_bd),
     ylim = range(ellipse_bd[, 2], ci_delta_bd))
abline(v = ci_beta_bd, col = "#808080", lty = 3, lwd = 1.5)
abline(h = ci_delta_bd, col = "#808080", lty = 3, lwd = 1.5)
points(mle[2], mle[4], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[2], true_params[4], pch = 17, col = "#006400", cex = 1.5)
grid(col = "gray90")

legend("topright",
       legend = c("MLE", "True", "95% CR", "Marginal 95% CI"),
       col = c("#8B0000", "#006400", "#2E4057", "#808080"),
       pch = c(19, 17, NA, NA),
       lty = c(NA, NA, 1, 3),
       lwd = c(NA, NA, 2, 1.5),
       bty = "n", cex = 0.8)
```

grekw — *Gradient of the Negative Log-Likelihood for the EKw Distribution*

Description

Computes the gradient vector (vector of first partial derivatives) of the negative log-likelihood function for the Exponentiated Kumaraswamy (EKw) distribution with parameters `alpha` (α), `beta` (β), and `lambda` (λ). This distribution is the special case of the Generalized Kumaraswamy (GKw) distribution where $\gamma = 1$ and $\delta = 0$. The gradient is useful for optimization.

Usage

```
grekw(par, data)
```

Arguments

`par`	A numeric vector of length 3 containing the distribution parameters in the order: `alpha` ($\alpha > 0$), `beta` ($\beta > 0$), `lambda` ($\lambda > 0$).
`data`	A numeric vector of observations. All values must be strictly between 0 and 1 (exclusive).

Details

The components of the gradient vector of the negative log-likelihood ($-\nabla\ell(\theta|\mathbf{x})$) for the EKw ($\gamma = 1, \delta = 0$) model are:

$$-\frac{\partial \ell}{\partial \alpha} = -\frac{n}{\alpha} - \sum_{i=1}^{n} \ln(x_i) + \sum_{i=1}^{n} \left[x_i^{\alpha} \ln(x_i) \left(\frac{\beta - 1}{v_i} - \frac{(\lambda - 1)\beta v_i^{\beta-1}}{w_i} \right) \right]$$

$$-\frac{\partial \ell}{\partial \beta} = -\frac{n}{\beta} - \sum_{i=1}^{n} \ln(v_i) + \sum_{i=1}^{n} \left[\frac{(\lambda - 1) v_i^{\beta} \ln(v_i)}{w_i} \right]$$

$$-\frac{\partial \ell}{\partial \lambda} = -\frac{n}{\lambda} - \sum_{i=1}^{n} \ln(w_i)$$

where:

- $v_i = 1 - x_i^{\alpha}$
- $w_i = 1 - v_i^{\beta} = 1 - (1 - x_i^{\alpha})^{\beta}$

These formulas represent the derivatives of $-\ell(\theta)$, consistent with minimizing the negative log-likelihood. They correspond to the relevant components of the general GKw gradient (`grgkw`) evaluated at $\gamma = 1, \delta = 0$.

Value

Returns a numeric vector of length 3 containing the partial derivatives of the negative log-likelihood function $-\ell(\theta|\mathbf{x})$ with respect to each parameter: $(-\partial\ell/\partial\alpha, -\partial\ell/\partial\beta, -\partial\ell/\partial\lambda)$. Returns a vector of `NaN` if any parameter values are invalid according to their constraints, or if any value in `data` is not in the interval (0, 1).

Author(s)

Lopes, J. E.

References

Nadarajah, S., Cordeiro, G. M., & Ortega, E. M. (2012). The exponentiated Kumaraswamy distribution. *Journal of the Franklin Institute*, *349*(3),

Cordeiro, G. M., & de Castro, M. (2011). A new family of generalized distributions. *Journal of Statistical Computation and Simulation*,

Kumaraswamy, P. (1980). A generalized probability density function for double-bounded random processes. *Journal of Hydrology*, *46*(1-2), 79-88.

(Note: Specific gradient formulas might be derived or sourced from additional references).

See Also

`grgkw` (parent distribution gradient), `llekw` (negative log-likelihood for EKw), `hsekw` (Hessian for EKw, if available), `dekw` (density for EKw), `optim`, `grad` (for numerical gradient comparison).

Examples

```
## Example 1: Basic Gradient Evaluation

# Generate sample data
set.seed(123)
n <- 1000
true_params <- c(alpha = 2.5, beta = 3.5, lambda = 2.0)
data <- rekw(n, alpha = true_params[1], beta = true_params[2],
             lambda = true_params[3])

# Evaluate gradient at true parameters
grad_true <- grekw(par = true_params, data = data)
cat("Gradient at true parameters:\n")
print(grad_true)
cat("Norm:", sqrt(sum(grad_true^2)), "\n")

# Evaluate at different parameter values
test_params <- rbind(
  c(2.0, 3.0, 1.5),
  c(2.5, 3.5, 2.0),
  c(3.0, 4.0, 2.5)
)

grad_norms <- apply(test_params, 1, function(p) {
  g <- grekw(p, data)
  sqrt(sum(g^2))
})

results <- data.frame(
  Alpha = test_params[, 1],
  Beta = test_params[, 2],
  Lambda = test_params[, 3],
  Grad_Norm = grad_norms
)
print(results, digits = 4)


## Example 2: Gradient in Optimization

# Optimization with analytical gradient
fit_with_grad <- optim(
  par = c(2, 3, 1.5),
  fn = llekw,
  gr = grekw,
  data = data,
  method = "BFGS",
  hessian = TRUE,
  control = list(trace = 0)
)

# Optimization without gradient
fit_no_grad <- optim(
  par = c(2, 3, 1.5),
  fn = llekw,
  data = data,
  method = "BFGS",
  hessian = TRUE,
  control = list(trace = 0)
)

comparison <- data.frame(
  Method = c("With Gradient", "Without Gradient"),
  Alpha = c(fit_with_grad$par[1], fit_no_grad$par[1]),
  Beta = c(fit_with_grad$par[2], fit_no_grad$par[2]),
  Lambda = c(fit_with_grad$par[3], fit_no_grad$par[3]),
  NegLogLik = c(fit_with_grad$value, fit_no_grad$value),
  Iterations = c(fit_with_grad$counts[1], fit_no_grad$counts[1])
)
print(comparison, digits = 4, row.names = FALSE)


## Example 3: Verifying Gradient at MLE

mle <- fit_with_grad$par
names(mle) <- c("alpha", "beta", "lambda")

# At MLE, gradient should be approximately zero
gradient_at_mle <- grekw(par = mle, data = data)
cat("\nGradient at MLE:\n")
print(gradient_at_mle)
cat("Max absolute component:", max(abs(gradient_at_mle)), "\n")
cat("Gradient norm:", sqrt(sum(gradient_at_mle^2)), "\n")


## Example 4: Numerical vs Analytical Gradient

# Manual finite difference gradient
numerical_gradient <- function(f, x, data, h = 1e-7) {
  grad <- numeric(length(x))
  for (i in seq_along(x)) {
    x_plus <- x_minus <- x
    x_plus[i] <- x[i] + h
    x_minus[i] <- x[i] - h
    grad[i] <- (f(x_plus, data) - f(x_minus, data)) / (2 * h)
  }
  return(grad)
}

# Compare at MLE
grad_analytical <- grekw(par = mle, data = data)
grad_numerical <- numerical_gradient(llekw, mle, data)

comparison_grad <- data.frame(
  Parameter = c("alpha", "beta", "lambda"),
  Analytical = grad_analytical,
  Numerical = grad_numerical,
  Abs_Diff = abs(grad_analytical - grad_numerical),
  Rel_Error = abs(grad_analytical - grad_numerical) /
              (abs(grad_analytical) + 1e-10)
)
print(comparison_grad, digits = 8)


## Example 5: Score Test Statistic

# Score test for H0: theta = theta0
theta0 <- c(2.2, 3.2, 1.8)
score_theta0 <- -grekw(par = theta0, data = data)

# Fisher information at theta0
fisher_info <- hsekw(par = theta0, data = data)

# Score test statistic
score_stat <- t(score_theta0) %*% solve(fisher_info) %*% score_theta0
p_value <- pchisq(score_stat, df = 3, lower.tail = FALSE)

cat("\nScore Test:\n")
cat("H0: alpha=2.2, beta=3.2, lambda=1.8\n")
cat("Test statistic:", score_stat, "\n")
cat("P-value:", format.pval(p_value, digits = 4), "\n")


## Example 6: Confidence Ellipse (Alpha vs Beta)

# Observed information
obs_info <- hsekw(par = mle, data = data)
vcov_full <- solve(obs_info)
vcov_2d <- vcov_full[1:2, 1:2]

# Create confidence ellipse
theta <- seq(0, 2 * pi, length.out = 100)
chi2_val <- qchisq(0.95, df = 2)

eig_decomp <- eigen(vcov_2d)
ellipse <- matrix(NA, nrow = 100, ncol = 2)
for (i in 1:100) {
  v <- c(cos(theta[i]), sin(theta[i]))
  ellipse[i, ] <- mle[1:2] + sqrt(chi2_val) *
    (eig_decomp$vectors %*% diag(sqrt(eig_decomp$values)) %*% v)
}

# Marginal confidence intervals
se_2d <- sqrt(diag(vcov_2d))
ci_alpha <- mle[1] + c(-1, 1) * 1.96 * se_2d[1]
ci_beta <- mle[2] + c(-1, 1) * 1.96 * se_2d[2]

# Plot
```

```
plot(ellipse[, 1], ellipse[, 2], type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(alpha), ylab = expression(beta),
     main = "95% Confidence Region (Alpha vs Beta)", las = 1)

# Add marginal CIs
abline(v = ci_alpha, col = "#808080", lty = 3, lwd = 1.5)
abline(h = ci_beta, col = "#808080", lty = 3, lwd = 1.5)

points(mle[1], mle[2], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[2], pch = 17, col = "#006400", cex = 1.5)

legend("topright",
       legend = c("MLE", "True", "95% CR", "Marginal 95% CI"),
       col = c("#8B0000", "#006400", "#2E4057", "#808080"),
       pch = c(19, 17, NA, NA),
       lty = c(NA, NA, 1, 3),
       lwd = c(NA, NA, 2, 1.5),
       bty = "n")
grid(col = "gray90")


## Example 7: Confidence Ellipse (Alpha vs Lambda)

# Extract 2x2 submatrix for alpha and lambda
vcov_2d_al <- vcov_full[c(1, 3), c(1, 3)]

# Create confidence ellipse
eig_decomp_al <- eigen(vcov_2d_al)
ellipse_al <- matrix(NA, nrow = 100, ncol = 2)
for (i in 1:100) {
  v <- c(cos(theta[i]), sin(theta[i]))
  ellipse_al[i, ] <- mle[c(1, 3)] + sqrt(chi2_val) *
    (eig_decomp_al$vectors %*% diag(sqrt(eig_decomp_al$values)) %*% v)
}

# Marginal confidence intervals
se_2d_al <- sqrt(diag(vcov_2d_al))
ci_alpha_2 <- mle[1] + c(-1, 1) * 1.96 * se_2d_al[1]
ci_lambda <- mle[3] + c(-1, 1) * 1.96 * se_2d_al[2]

# Plot

plot(ellipse_al[, 1], ellipse_al[, 2], type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(alpha), ylab = expression(lambda),
     main = "95% Confidence Region (Alpha vs Lambda)", las = 1)

# Add marginal CIs
abline(v = ci_alpha_2, col = "#808080", lty = 3, lwd = 1.5)
abline(h = ci_lambda, col = "#808080", lty = 3, lwd = 1.5)

points(mle[1], mle[3], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[3], pch = 17, col = "#006400", cex = 1.5)
```

```
legend("topright",
       legend = c("MLE", "True", "95% CR", "Marginal 95% CI"),
       col = c("#8B0000", "#006400", "#2E4057", "#808080"),
       pch = c(19, 17, NA, NA),
       lty = c(NA, NA, 1, 3),
       lwd = c(NA, NA, 2, 1.5),
       bty = "n")
grid(col = "gray90")


## Example 8: Confidence Ellipse (Beta vs Lambda)

# Extract 2x2 submatrix for beta and lambda
vcov_2d_bl <- vcov_full[2:3, 2:3]

# Create confidence ellipse
eig_decomp_bl <- eigen(vcov_2d_bl)
ellipse_bl <- matrix(NA, nrow = 100, ncol = 2)
for (i in 1:100) {
  v <- c(cos(theta[i]), sin(theta[i]))
  ellipse_bl[i, ] <- mle[2:3] + sqrt(chi2_val) *
    (eig_decomp_bl$vectors %*% diag(sqrt(eig_decomp_bl$values)) %*% v)
}

# Marginal confidence intervals
se_2d_bl <- sqrt(diag(vcov_2d_bl))
ci_beta_2 <- mle[2] + c(-1, 1) * 1.96 * se_2d_bl[1]
ci_lambda_2 <- mle[3] + c(-1, 1) * 1.96 * se_2d_bl[2]

# Plot

plot(ellipse_bl[, 1], ellipse_bl[, 2], type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(beta), ylab = expression(lambda),
     main = "95% Confidence Region (Beta vs Lambda)", las = 1)

# Add marginal CIs
abline(v = ci_beta_2, col = "#808080", lty = 3, lwd = 1.5)
abline(h = ci_lambda_2, col = "#808080", lty = 3, lwd = 1.5)

points(mle[2], mle[3], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[2], true_params[3], pch = 17, col = "#006400", cex = 1.5)

legend("topright",
       legend = c("MLE", "True", "95% CR", "Marginal 95% CI"),
       col = c("#8B0000", "#006400", "#2E4057", "#808080"),
       pch = c(19, 17, NA, NA),
       lty = c(NA, NA, 1, 3),
       lwd = c(NA, NA, 2, 1.5),
       bty = "n")
grid(col = "gray90")
```

`grgkw` *Gradient of the Negative Log-Likelihood for the GKw Distribution*

Description

Computes the gradient vector (vector of partial derivatives) of the negative log-likelihood function for the five-parameter Generalized Kumaraswamy (GKw) distribution. This provides the analytical gradient, often used for efficient optimization via maximum likelihood estimation.

Usage

```
grgkw(par, data)
```

Arguments

`par`	A numeric vector of length 5 containing the distribution parameters in the order: `alpha` ($\alpha > 0$), `beta` ($\beta > 0$), `gamma` ($\gamma > 0$), `delta` ($\delta \geq 0$), `lambda` ($\lambda > 0$).
`data`	A numeric vector of observations. All values must be strictly between 0 and 1 (exclusive).

Details

The components of the gradient vector of the negative log-likelihood ($-\nabla\ell(\theta|\mathbf{x})$) are:

$$-\frac{\partial\ell}{\partial\alpha} = -\frac{n}{\alpha} - \sum_{i=1}^{n}\ln(x_i) + \sum_{i=1}^{n}\left[x_i^{\alpha}\ln(x_i)\left(\frac{\beta-1}{v_i} - \frac{(\gamma\lambda-1)\beta v_i^{\beta-1}}{w_i} + \frac{\delta\lambda\beta v_i^{\beta-1}w_i^{\lambda-1}}{z_i}\right)\right]$$

$$-\frac{\partial\ell}{\partial\beta} = -\frac{n}{\beta} - \sum_{i=1}^{n}\ln(v_i) + \sum_{i=1}^{n}\left[v_i^{\beta}\ln(v_i)\left(\frac{\gamma\lambda-1}{w_i} - \frac{\delta\lambda w_i^{\lambda-1}}{z_i}\right)\right]$$

$$-\frac{\partial\ell}{\partial\gamma} = n[\psi(\gamma) - \psi(\gamma+\delta+1)] - \lambda\sum_{i=1}^{n}\ln(w_i)$$

$$-\frac{\partial\ell}{\partial\delta} = n[\psi(\delta+1) - \psi(\gamma+\delta+1)] - \sum_{i=1}^{n}\ln(z_i)$$

$$-\frac{\partial\ell}{\partial\lambda} = -\frac{n}{\lambda} - \gamma\sum_{i=1}^{n}\ln(w_i) + \delta\sum_{i=1}^{n}\frac{w_i^{\lambda}\ln(w_i)}{z_i}$$

where:

- $v_i = 1 - x_i^{\alpha}$
- $w_i = 1 - v_i^{\beta} = 1 - (1 - x_i^{\alpha})^{\beta}$
- $z_i = 1 - w_i^{\lambda} = 1 - [1 - (1 - x_i^{\alpha})^{\beta}]^{\lambda}$
- $\psi(\cdot)$ is the digamma function (`digamma`).

Numerical stability is ensured through careful implementation, including checks for valid inputs and handling of intermediate calculations involving potentially small or large numbers, often leveraging the Armadillo C++ library for efficiency.

Value

Returns a numeric vector of length 5 containing the partial derivatives of the negative log-likelihood function $-\ell(\theta|\mathbf{x})$ with respect to each parameter: $(-\partial\ell/\partial\alpha, -\partial\ell/\partial\beta, -\partial\ell/\partial\gamma, -\partial\ell/\partial\delta, -\partial\ell/\partial\lambda)$. Returns a vector of `NaN` if any parameter values are invalid according to their constraints, or if any value in `data` is not in the interval (0, 1).

Author(s)

Lopes, J. E.

References

Cordeiro, G. M., & de Castro, M. (2011). A new family of generalized distributions. *Journal of Statistical Computation and Simulation*

Kumaraswamy, P. (1980). A generalized probability density function for double-bounded random processes. *Journal of Hydrology*, *46*(1-2), 79-88.

See Also

`llgkw` (negative log-likelihood), `hsgkw` (Hessian matrix), `dgkw` (density), `optim`, `grad` (for numerical gradient comparison), `digamma`

Examples

```
## Example 1: Basic Gradient Evaluation


# Generate sample data
set.seed(123)
n <- 1000
true_params <- c(alpha = 2.0, beta = 3.0, gamma = 1.5, delta = 2.0, lambda = 1.8)
data <- rgkw(n, alpha = true_params[1], beta = true_params[2],
             gamma = true_params[3], delta = true_params[4],
             lambda = true_params[5])

# Evaluate gradient at true parameters
grad_true <- grgkw(par = true_params, data = data)
cat("Gradient at true parameters:\n")
print(grad_true)
cat("Norm:", sqrt(sum(grad_true^2)), "\n")

# Evaluate at different parameter values
test_params <- rbind(
  c(1.5, 2.5, 1.2, 1.5, 1.5),
  c(2.0, 3.0, 1.5, 2.0, 1.8),
  c(2.5, 3.5, 1.8, 2.5, 2.0)
)

grad_norms <- apply(test_params, 1, function(p) {
  g <- grgkw(p, data)
  sqrt(sum(g^2))
})

results <- data.frame(
  Alpha = test_params[, 1],
  Beta = test_params[, 2],
  Gamma = test_params[, 3],
  Delta = test_params[, 4],
  Lambda = test_params[, 5],
  Grad_Norm = grad_norms
)
print(results, digits = 4)


## Example 2: Gradient in Optimization

# Optimization with analytical gradient
fit_with_grad <- optim(
  par = c(1.5, 2.5, 1.2, 1.5, 1.5),
  fn = llgkw,
  gr = grgkw,
  data = data,
  method = "BFGS",
  hessian = TRUE,
  control = list(trace = 0, maxit = 1000)
)

# Optimization without gradient
fit_no_grad <- optim(
  par = c(1.5, 2.5, 1.2, 1.5, 1.5),
  fn = llgkw,
  data = data,
  method = "BFGS",
  hessian = TRUE,
  control = list(trace = 0, maxit = 1000)
)

comparison <- data.frame(
  Method = c("With Gradient", "Without Gradient"),
  Alpha = c(fit_with_grad$par[1], fit_no_grad$par[1]),
  Beta = c(fit_with_grad$par[2], fit_no_grad$par[2]),
  Gamma = c(fit_with_grad$par[3], fit_no_grad$par[3]),
  Delta = c(fit_with_grad$par[4], fit_no_grad$par[4]),
  Lambda = c(fit_with_grad$par[5], fit_no_grad$par[5]),
  NegLogLik = c(fit_with_grad$value, fit_no_grad$value),
  Iterations = c(fit_with_grad$counts[1], fit_no_grad$counts[1])
)
print(comparison, digits = 4, row.names = FALSE)
```

```
## Example 3: Verifying Gradient at MLE

mle <- fit_with_grad$par
names(mle) <- c("alpha", "beta", "gamma", "delta", "lambda")

# At MLE, gradient should be approximately zero
gradient_at_mle <- grgkw(par = mle, data = data)
cat("\nGradient at MLE:\n")
print(gradient_at_mle)
cat("Max absolute component:", max(abs(gradient_at_mle)), "\n")
cat("Gradient norm:", sqrt(sum(gradient_at_mle^2)), "\n")


## Example 4: Numerical vs Analytical Gradient

# Manual finite difference gradient
numerical_gradient <- function(f, x, data, h = 1e-7) {
  grad <- numeric(length(x))
  for (i in seq_along(x)) {
    x_plus <- x_minus <- x
    x_plus[i] <- x[i] + h
    x_minus[i] <- x[i] - h
    grad[i] <- (f(x_plus, data) - f(x_minus, data)) / (2 * h)
  }
  return(grad)
}

# Compare at MLE
grad_analytical <- grgkw(par = mle, data = data)
grad_numerical <- numerical_gradient(llgkw, mle, data)

comparison_grad <- data.frame(
  Parameter = c("alpha", "beta", "gamma", "delta", "lambda"),
  Analytical = grad_analytical,
  Numerical = grad_numerical,
  Abs_Diff = abs(grad_analytical - grad_numerical),
  Rel_Error = abs(grad_analytical - grad_numerical) /
              (abs(grad_analytical) + 1e-10)
)
print(comparison_grad, digits = 8)


## Example 5: Score Test Statistic

# Score test for H0: theta = theta0
theta0 <- c(1.8, 2.8, 1.3, 1.8, 1.6)
score_theta0 <- grgkw(par = theta0, data = data)

# Fisher information at theta0
fisher_info <- hsgkw(par = theta0, data = data)

# Score test statistic
```

```
score_stat <- t(score_theta0) %*% solve(fisher_info) %*% score_theta0
p_value <- pchisq(score_stat, df = 5, lower.tail = FALSE)

cat("\nScore Test:\n")
cat("H0: alpha=1.8, beta=2.8, gamma=1.3, delta=1.8, lambda=1.6\n")
cat("Test statistic:", score_stat, "\n")
cat("P-value:", format.pval(p_value, digits = 4), "\n")


## Example 6: Confidence Ellipse (Alpha vs Beta)

# Observed information
obs_info <- hsgkw(par = mle, data = data)
vcov_full <- solve(obs_info)
vcov_2d <- vcov_full[1:2, 1:2]

# Create confidence ellipse
theta <- seq(0, 2 * pi, length.out = round(n/4))
chi2_val <- qchisq(0.95, df = 2)

eig_decomp <- eigen(vcov_2d)
ellipse <- matrix(NA, nrow = round(n/4), ncol = 2)
for (i in 1:round(n/4)) {
  v <- c(cos(theta[i]), sin(theta[i]))
  ellipse[i, ] <- mle[1:2] + sqrt(chi2_val) *
    (eig_decomp$vectors %*% diag(sqrt(eig_decomp$values)) %*% v)
}

# Marginal confidence intervals
se_2d <- sqrt(diag(vcov_2d))
ci_alpha <- mle[1] + c(-1, 1) * 1.96 * se_2d[1]
ci_beta <- mle[2] + c(-1, 1) * 1.96 * se_2d[2]

# Plot
plot(ellipse[, 1], ellipse[, 2], type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(alpha), ylab = expression(beta),
     main = "95% Confidence Region (Alpha vs Beta)", las = 1)

# Add marginal CIs
abline(v = ci_alpha, col = "#808080", lty = 3, lwd = 1.5)
abline(h = ci_beta, col = "#808080", lty = 3, lwd = 1.5)

points(mle[1], mle[2], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[2], pch = 17, col = "#006400", cex = 1.5)

legend("topright",
       legend = c("MLE", "True", "95% CR", "Marginal 95% CI"),
       col = c("#8B0000", "#006400", "#2E4057", "#808080"),
       pch = c(19, 17, NA, NA),
       lty = c(NA, NA, 1, 3),
       lwd = c(NA, NA, 2, 1.5),
       bty = "n")
grid(col = "gray90")
```

```
## Example 7: Confidence Ellipse (Gamma vs Delta)

# Extract 2x2 submatrix for gamma and delta
vcov_2d_gd <- vcov_full[3:4, 3:4]

# Create confidence ellipse
eig_decomp_gd <- eigen(vcov_2d_gd)
ellipse_gd <- matrix(NA, nrow = round(n/4), ncol = 2)
for (i in 1:round(n/4)) {
  v <- c(cos(theta[i]), sin(theta[i]))
  ellipse_gd[i, ] <- mle[3:4] + sqrt(chi2_val) *
    (eig_decomp_gd$vectors %*% diag(sqrt(eig_decomp_gd$values)) %*% v)
}

# Marginal confidence intervals
se_2d_gd <- sqrt(diag(vcov_2d_gd))
ci_gamma <- mle[3] + c(-1, 1) * 1.96 * se_2d_gd[1]
ci_delta <- mle[4] + c(-1, 1) * 1.96 * se_2d_gd[2]

# Plot
plot(ellipse_gd[, 1], ellipse_gd[, 2], type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(gamma), ylab = expression(delta),
     main = "95% Confidence Region (Gamma vs Delta)", las = 1)

# Add marginal CIs
abline(v = ci_gamma, col = "#808080", lty = 3, lwd = 1.5)
abline(h = ci_delta, col = "#808080", lty = 3, lwd = 1.5)

points(mle[3], mle[4], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[3], true_params[4], pch = 17, col = "#006400", cex = 1.5)

legend("topright",
       legend = c("MLE", "True", "95% CR", "Marginal 95% CI"),
       col = c("#8B0000", "#006400", "#2E4057", "#808080"),
       pch = c(19, 17, NA, NA),
       lty = c(NA, NA, 1, 3),
       lwd = c(NA, NA, 2, 1.5),
       bty = "n")
grid(col = "gray90")
```

grkkw *Gradient of the Negative Log-Likelihood for the kkw Distribution*

Description

Computes the gradient vector (vector of first partial derivatives) of the negative log-likelihood function for the Kumaraswamy-Kumaraswamy (kkw) distribution with parameters `alpha` (α), `beta` (β), `delta` (δ), and `lambda` (λ). This distribution is the special case of the Generalized Kumaraswamy (GKw) distribution where $\gamma = 1$. The gradient is typically used in optimization algorithms for maximum likelihood estimation.

Usage

```
grkkw(par, data)
```

Arguments

`par`	A numeric vector of length 4 containing the distribution parameters in the order: `alpha` ($\alpha > 0$), `beta` ($\beta > 0$), `delta` ($\delta \geq 0$), `lambda` ($\lambda > 0$).
`data`	A numeric vector of observations. All values must be strictly between 0 and 1 (exclusive).

Details

The components of the gradient vector of the negative log-likelihood ($-\nabla\ell(\theta|\mathbf{x})$) for the kkw ($\gamma = 1$) model are:

$$-\frac{\partial\ell}{\partial\alpha} = -\frac{n}{\alpha} - \sum_{i=1}^{n}\ln(x_i) + (\beta-1)\sum_{i=1}^{n}\frac{x_i^{\alpha}\ln(x_i)}{v_i} - (\lambda-1)\sum_{i=1}^{n}\frac{\beta v_i^{\beta-1}x_i^{\alpha}\ln(x_i)}{w_i} + \delta\sum_{i=1}^{n}\frac{\lambda w_i^{\lambda-1}\beta v_i^{\beta-1}x_i^{\alpha}\ln(x_i)}{z_i}$$

$$-\frac{\partial\ell}{\partial\beta} = -\frac{n}{\beta} - \sum_{i=1}^{n}\ln(v_i) + (\lambda-1)\sum_{i=1}^{n}\frac{v_i^{\beta}\ln(v_i)}{w_i} - \delta\sum_{i=1}^{n}\frac{\lambda w_i^{\lambda-1}v_i^{\beta}\ln(v_i)}{z_i}$$

$$-\frac{\partial\ell}{\partial\delta} = -\frac{n}{\delta+1} - \sum_{i=1}^{n}\ln(z_i)$$

$$-\frac{\partial\ell}{\partial\lambda} = -\frac{n}{\lambda} - \sum_{i=1}^{n}\ln(w_i) + \delta\sum_{i=1}^{n}\frac{w_i^{\lambda}\ln(w_i)}{z_i}$$

where:

- $v_i = 1 - x_i^{\alpha}$
- $w_i = 1 - v_i^{\beta} = 1 - (1 - x_i^{\alpha})^{\beta}$
- $z_i = 1 - w_i^{\lambda} = 1 - [1 - (1 - x_i^{\alpha})^{\beta}]^{\lambda}$

These formulas represent the derivatives of $-\ell(\theta)$, consistent with minimizing the negative log-likelihood. They correspond to the general GKw gradient (`grgkw`) components for $\alpha, \beta, \delta, \lambda$ evaluated at $\gamma = 1$. Note that the component for γ is omitted. Numerical stability is maintained through careful implementation.

Value

Returns a numeric vector of length 4 containing the partial derivatives of the negative log-likelihood function $-\ell(\theta|\mathbf{x})$ with respect to each parameter: $(-\partial\ell/\partial\alpha, -\partial\ell/\partial\beta, -\partial\ell/\partial\delta, -\partial\ell/\partial\lambda)$. Returns a vector of NaN if any parameter values are invalid according to their constraints, or if any value in data is not in the interval (0, 1).

Author(s)

Lopes, J. E.

References

Cordeiro, G. M., & de Castro, M. (2011). A new family of generalized distributions. *Journal of Statistical Computation and Simulation*,

Kumaraswamy, P. (1980). A generalized probability density function for double-bounded random processes. *Journal of Hydrology*, *46*(1-2), 79-88.

See Also

grgkw (parent distribution gradient), llkkw (negative log-likelihood for kkw), hskkw (Hessian for kkw), dkkw (density for kkw), optim, grad (for numerical gradient comparison).

Examples

```
## Example 1: Basic Gradient Evaluation

# Generate sample data
set.seed(123)
n <- 1000
true_params <- c(alpha = 2.0, beta = 3.0, delta = 1.5, lambda = 2.0)
data <- rkkw(n, alpha = true_params[1], beta = true_params[2],
             delta = true_params[3], lambda = true_params[4])

# Evaluate gradient at true parameters
grad_true <- grkkw(par = true_params, data = data)
cat("Gradient at true parameters:\n")
print(grad_true)
cat("Norm:", sqrt(sum(grad_true^2)), "\n")

# Evaluate at different parameter values
test_params <- rbind(
  c(1.5, 2.5, 1.0, 1.5),
  c(2.0, 3.0, 1.5, 2.0),
  c(2.5, 3.5, 2.0, 2.5)
)

grad_norms <- apply(test_params, 1, function(p) {
  g <- grkkw(p, data)
  sqrt(sum(g^2))
})
```

```
results <- data.frame(
  Alpha = test_params[, 1],
  Beta = test_params[, 2],
  Delta = test_params[, 3],
  Lambda = test_params[, 4],
  Grad_Norm = grad_norms
)
print(results, digits = 4)


## Example 2: Gradient in Optimization

# Optimization with analytical gradient
fit_with_grad <- optim(
  par = c(1.5, 2.5, 1.0, 1.5),
  fn = llkkw,
  gr = grkkw,
  data = data,
  method = "BFGS",
  hessian = TRUE,
  control = list(trace = 0)
)

# Optimization without gradient
fit_no_grad <- optim(
  par = c(1.5, 2.5, 1.0, 1.5),
  fn = llkkw,
  data = data,
  method = "BFGS",
  hessian = TRUE,
  control = list(trace = 0)
)

comparison <- data.frame(
  Method = c("With Gradient", "Without Gradient"),
  Alpha = c(fit_with_grad$par[1], fit_no_grad$par[1]),
  Beta = c(fit_with_grad$par[2], fit_no_grad$par[2]),
  Delta = c(fit_with_grad$par[3], fit_no_grad$par[3]),
  Lambda = c(fit_with_grad$par[4], fit_no_grad$par[4]),
  NegLogLik = c(fit_with_grad$value, fit_no_grad$value),
  Iterations = c(fit_with_grad$counts[1], fit_no_grad$counts[1])
)
print(comparison, digits = 4, row.names = FALSE)


## Example 3: Verifying Gradient at MLE

mle <- fit_with_grad$par
names(mle) <- c("alpha", "beta", "delta", "lambda")

# At MLE, gradient should be approximately zero
gradient_at_mle <- grkkw(par = mle, data = data)
cat("\nGradient at MLE:\n")
```

```
print(gradient_at_mle)
cat("Max absolute component:", max(abs(gradient_at_mle)), "\n")
cat("Gradient norm:", sqrt(sum(gradient_at_mle^2)), "\n")


## Example 4: Numerical vs Analytical Gradient

# Manual finite difference gradient
numerical_gradient <- function(f, x, data, h = 1e-7) {
  grad <- numeric(length(x))
  for (i in seq_along(x)) {
    x_plus <- x_minus <- x
    x_plus[i] <- x[i] + h
    x_minus[i] <- x[i] - h
    grad[i] <- (f(x_plus, data) - f(x_minus, data)) / (2 * h)
  }
  return(grad)
}

# Compare at MLE
grad_analytical <- grkkw(par = mle, data = data)
grad_numerical <- numerical_gradient(llkkw, mle, data)

comparison_grad <- data.frame(
  Parameter = c("alpha", "beta", "delta", "lambda"),
  Analytical = grad_analytical,
  Numerical = grad_numerical,
  Abs_Diff = abs(grad_analytical - grad_numerical),
  Rel_Error = abs(grad_analytical - grad_numerical) /
              (abs(grad_analytical) + 1e-10)
)
print(comparison_grad, digits = 8)


## Example 5: Score Test Statistic

# Score test for H0: theta = theta0
theta0 <- c(1.8, 2.8, 1.3, 1.8)
score_theta0 <- -grkkw(par = theta0, data = data)

# Fisher information at theta0
fisher_info <- hskkw(par = theta0, data = data)

# Score test statistic
score_stat <- t(score_theta0) %*% solve(fisher_info) %*% score_theta0
p_value <- pchisq(score_stat, df = 4, lower.tail = FALSE)

cat("\nScore Test:\n")
cat("H0: alpha=1.8, beta=2.8, delta=1.3, lambda=1.8\n")
cat("Test statistic:", score_stat, "\n")
cat("P-value:", format.pval(p_value, digits = 4), "\n")
```

```
## Example 6: Confidence Ellipse with Gradient Information

# For visualization, use first two parameters (alpha, beta)
# Observed information
obs_info <- hskkw(par = mle, data = data)
vcov_full <- solve(obs_info)
vcov_2d <- vcov_full[1:2, 1:2]

# Create confidence ellipse
theta <- seq(0, 2 * pi, length.out = 100)
chi2_val <- qchisq(0.95, df = 2)

eig_decomp <- eigen(vcov_2d)
ellipse <- matrix(NA, nrow = 100, ncol = 2)
for (i in 1:100) {
  v <- c(cos(theta[i]), sin(theta[i]))
  ellipse[i, ] <- mle[1:2] + sqrt(chi2_val) *
    (eig_decomp$vectors %*% diag(sqrt(eig_decomp$values)) %*% v)
}

# Marginal confidence intervals
se_2d <- sqrt(diag(vcov_2d))
ci_alpha <- mle[1] + c(-1, 1) * 1.96 * se_2d[1]
ci_beta <- mle[2] + c(-1, 1) * 1.96 * se_2d[2]

# Plot
plot(ellipse[, 1], ellipse[, 2], type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(alpha), ylab = expression(beta),
     main = "95% Confidence Region (Alpha vs Beta)", las = 1)

# Add marginal CIs
abline(v = ci_alpha, col = "#808080", lty = 3, lwd = 1.5)
abline(h = ci_beta, col = "#808080", lty = 3, lwd = 1.5)

points(mle[1], mle[2], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[2], pch = 17, col = "#006400", cex = 1.5)

legend("topright",
       legend = c("MLE", "True", "95% CR", "Marginal 95% CI"),
       col = c("#8B0000", "#006400", "#2E4057", "#808080"),
       pch = c(19, 17, NA, NA),
       lty = c(NA, NA, 1, 3),
       lwd = c(NA, NA, 2, 1.5),
       bty = "n")
grid(col = "gray90")
```

grkw *Gradient of the Negative Log-Likelihood for the Kumaraswamy (Kw) Distribution*

Description

Computes the gradient vector (vector of first partial derivatives) of the negative log-likelihood function for the two-parameter Kumaraswamy (Kw) distribution with parameters `alpha` (α) and `beta` (β). This provides the analytical gradient often used for efficient optimization via maximum likelihood estimation.

Usage

```
grkw(par, data)
```

Arguments

`par`	A numeric vector of length 2 containing the distribution parameters in the order: `alpha` ($\alpha > 0$), `beta` ($\beta > 0$).
`data`	A numeric vector of observations. All values must be strictly between 0 and 1 (exclusive).

Details

The components of the gradient vector of the negative log-likelihood ($-\nabla \ell(\theta|\mathbf{x})$) for the Kw model are:

$$-\frac{\partial \ell}{\partial \alpha} = -\frac{n}{\alpha} - \sum_{i=1}^{n} \ln(x_i) + (\beta - 1) \sum_{i=1}^{n} \frac{x_i^{\alpha} \ln(x_i)}{v_i}$$

$$-\frac{\partial \ell}{\partial \beta} = -\frac{n}{\beta} - \sum_{i=1}^{n} \ln(v_i)$$

where $v_i = 1 - x_i^{\alpha}$. These formulas represent the derivatives of $-\ell(\theta)$, consistent with minimizing the negative log-likelihood. They correspond to the relevant components of the general GKw gradient (`grgkw`) evaluated at $\gamma = 1, \delta = 0, \lambda = 1$.

Value

Returns a numeric vector of length 2 containing the partial derivatives of the negative log-likelihood function $-\ell(\theta|\mathbf{x})$ with respect to each parameter: $(-\partial \ell/\partial \alpha, -\partial \ell/\partial \beta)$. Returns a vector of `NaN` if any parameter values are invalid according to their constraints, or if any value in `data` is not in the interval (0, 1).

Author(s)

Lopes, J. E.

References

Kumaraswamy, P. (1980). A generalized probability density function for double-bounded random processes. *Journal of Hydrology*, *46*(1-2), 79-88.

Jones, M. C. (2009). Kumaraswamy's distribution: A beta-type distribution with some tractability advantages. *Statistical Methodology*, *6*(1), 70-81.

(Note: Specific gradient formulas might be derived or sourced from additional references).

See Also

`grgkw` (parent distribution gradient), `llkw` (negative log-likelihood for Kw), `hskw` (Hessian for Kw, if available), `dkw` (density for Kw), `optim`, `grad` (for numerical gradient comparison).

Examples

```
## Example 1: Basic Gradient Evaluation

# Generate sample data
set.seed(123)
n <- 1000
true_params <- c(alpha = 2.5, beta = 3.5)
data <- rkw(n, alpha = true_params[1], beta = true_params[2])

# Evaluate gradient at true parameters
grad_true <- grkw(par = true_params, data = data)
cat("Gradient at true parameters:\n")
print(grad_true)
cat("Norm:", sqrt(sum(grad_true^2)), "\n")

# Evaluate at different parameter values
test_params <- rbind(
  c(1.5, 2.5),
  c(2.0, 3.0),
  c(2.5, 3.5),
  c(3.0, 4.0)
)

grad_norms <- apply(test_params, 1, function(p) {
  g <- grkw(p, data)
  sqrt(sum(g^2))
})

results <- data.frame(
  Alpha = test_params[, 1],
  Beta = test_params[, 2],
  Grad_Norm = grad_norms
)
print(results, digits = 4)


## Example 2: Gradient in Optimization
```

```
# Optimization with analytical gradient
fit_with_grad <- optim(
  par = c(2, 2),
  fn = llkw,
  gr = grkw,
  data = data,
  method = "BFGS",
  hessian = TRUE,
  control = list(trace = 0)
)

# Optimization without gradient
fit_no_grad <- optim(
  par = c(2, 2),
  fn = llkw,
  data = data,
  method = "BFGS",
  hessian = TRUE,
  control = list(trace = 0)
)

comparison <- data.frame(
  Method = c("With Gradient", "Without Gradient"),
  Alpha = c(fit_with_grad$par[1], fit_no_grad$par[1]),
  Beta = c(fit_with_grad$par[2], fit_no_grad$par[2]),
  NegLogLik = c(fit_with_grad$value, fit_no_grad$value),
  Iterations = c(fit_with_grad$counts[1], fit_no_grad$counts[1])
)
print(comparison, digits = 4, row.names = FALSE)


## Example 3: Verifying Gradient at MLE

mle <- fit_with_grad$par
names(mle) <- c("alpha", "beta")

# At MLE, gradient should be approximately zero
gradient_at_mle <- grkw(par = mle, data = data)
cat("\nGradient at MLE:\n")
print(gradient_at_mle)
cat("Max absolute component:", max(abs(gradient_at_mle)), "\n")
cat("Gradient norm:", sqrt(sum(gradient_at_mle^2)), "\n")


## Example 4: Numerical vs Analytical Gradient

# Manual finite difference gradient
numerical_gradient <- function(f, x, data, h = 1e-7) {
  grad <- numeric(length(x))
  for (i in seq_along(x)) {
    x_plus <- x_minus <- x
    x_plus[i] <- x[i] + h
    x_minus[i] <- x[i] - h
    grad[i] <- (f(x_plus, data) - f(x_minus, data)) / (2 * h)
  }
  return(grad)
}

# Compare at several points
test_points <- rbind(
  c(1.5, 2.5),
  c(2.0, 3.0),
  mle,
  c(3.0, 4.0)
)

cat("\nNumerical vs Analytical Gradient Comparison:\n")
for (i in 1:nrow(test_points)) {
  grad_analytical <- grkw(par = test_points[i, ], data = data)
  grad_numerical <- numerical_gradient(llkw, test_points[i, ], data)

  cat("\nPoint", i, ": alpha =", test_points[i, 1],
      ", beta =", test_points[i, 2], "\n")

  comparison <- data.frame(
    Parameter = c("alpha", "beta"),
    Analytical = grad_analytical,
    Numerical = grad_numerical,
    Abs_Diff = abs(grad_analytical - grad_numerical),
    Rel_Error = abs(grad_analytical - grad_numerical) /
                (abs(grad_analytical) + 1e-10)
  )
  print(comparison, digits = 8)
}


## Example 5: Gradient Path Visualization

# Create grid
alpha_grid <- seq(mle[1] - 1, mle[1] + 1, length.out = 20)
beta_grid <- seq(mle[2] - 1, mle[2] + 1, length.out = 20)
alpha_grid <- alpha_grid[alpha_grid > 0]
beta_grid <- beta_grid[beta_grid > 0]

# Compute gradient vectors
grad_alpha <- matrix(NA, nrow = length(alpha_grid), ncol = length(beta_grid))
grad_beta <- matrix(NA, nrow = length(alpha_grid), ncol = length(beta_grid))

for (i in seq_along(alpha_grid)) {
  for (j in seq_along(beta_grid)) {
    g <- grkw(c(alpha_grid[i], beta_grid[j]), data)
    grad_alpha[i, j] <- -g[1]  # Negative for gradient ascent
    grad_beta[i, j] <- -g[2]
  }
}
```

```
# Plot gradient field

plot(mle[1], mle[2], pch = 19, col = "#8B0000", cex = 1.5,
     xlim = range(alpha_grid), ylim = range(beta_grid),
     xlab = expression(alpha), ylab = expression(beta),
     main = "Gradient Vector Field", las = 1)

# Subsample for clearer visualization
step <- 2
for (i in seq(1, length(alpha_grid), by = step)) {
  for (j in seq(1, length(beta_grid), by = step)) {
    arrows(alpha_grid[i], beta_grid[j],
           alpha_grid[i] + 0.05 * grad_alpha[i, j],
           beta_grid[j] + 0.05 * grad_beta[i, j],
           length = 0.05, col = "#2E4057", lwd = 1)
  }
}

points(true_params[1], true_params[2], pch = 17, col = "#006400", cex = 1.5)
legend("topright",
       legend = c("MLE", "True"),
       col = c("#8B0000", "#006400"),
       pch = c(19, 17), bty = "n")
grid(col = "gray90")


## Example 6: Score Test Statistic

# Score test for H0: theta = theta0
theta0 <- c(2, 3)
score_theta0 <- -grkw(par = theta0, data = data)  # Score is negative gradient

# Fisher information at theta0 (using Hessian)
fisher_info <- hskw(par = theta0, data = data)

# Score test statistic
score_stat <- t(score_theta0) %*% solve(fisher_info) %*% score_theta0
p_value <- pchisq(score_stat, df = 2, lower.tail = FALSE)

cat("\nScore Test:\n")
cat("H0: alpha = 2, beta = 3\n")
cat("Score vector:", score_theta0, "\n")
cat("Test statistic:", score_stat, "\n")
cat("P-value:", format.pval(p_value, digits = 4), "\n")
```

grmc *Gradient of the Negative Log-Likelihood for the McDonald (Mc)/Beta Power Distribution*

Description

Computes the gradient vector (vector of first partial derivatives) of the negative log-likelihood function for the McDonald (Mc) distribution (also known as Beta Power) with parameters `gamma` (γ), `delta` (δ), and `lambda` (λ). This distribution is the special case of the Generalized Kumaraswamy (GKw) distribution where $\alpha = 1$ and $\beta = 1$. The gradient is useful for optimization.

Usage

```
grmc(par, data)
```

Arguments

`par`	A numeric vector of length 3 containing the distribution parameters in the order: `gamma` ($\gamma > 0$), `delta` ($\delta \geq 0$), `lambda` ($\lambda > 0$).
`data`	A numeric vector of observations. All values must be strictly between 0 and 1 (exclusive).

Details

The components of the gradient vector of the negative log-likelihood ($-\nabla \ell(\theta|\mathbf{x})$) for the Mc ($\alpha = 1, \beta = 1$) model are:

$$-\frac{\partial \ell}{\partial \gamma} = n[\psi(\gamma + \delta + 1) - \psi(\gamma)] - \lambda \sum_{i=1}^{n} \ln(x_i)$$

$$-\frac{\partial \ell}{\partial \delta} = n[\psi(\gamma + \delta + 1) - \psi(\delta + 1)] - \sum_{i=1}^{n} \ln(1 - x_i^{\lambda})$$

$$-\frac{\partial \ell}{\partial \lambda} = -\frac{n}{\lambda} - \gamma \sum_{i=1}^{n} \ln(x_i) + \delta \sum_{i=1}^{n} \frac{x_i^{\lambda} \ln(x_i)}{1 - x_i^{\lambda}}$$

where $\psi(\cdot)$ is the digamma function (`digamma`). These formulas represent the derivatives of $-\ell(\theta)$, consistent with minimizing the negative log-likelihood. They correspond to the relevant components of the general GKw gradient (`grgkw`) evaluated at $\alpha = 1, \beta = 1$.

Value

Returns a numeric vector of length 3 containing the partial derivatives of the negative log-likelihood function $-\ell(\theta|\mathbf{x})$ with respect to each parameter: $(-\partial \ell/\partial \gamma, -\partial \ell/\partial \delta, -\partial \ell/\partial \lambda)$. Returns a vector of `NaN` if any parameter values are invalid according to their constraints, or if any value in `data` is not in the interval (0, 1).

Author(s)

Lopes, J. E.

References

McDonald, J. B. (1984). Some generalized functions for the size distribution of income. *Econometrica*, 52(3), 647-663.

Cordeiro, G. M., & de Castro, M. (2011). A new family of generalized distributions. *Journal of Statistical Computation and Simulation*,

(Note: Specific gradient formulas might be derived or sourced from additional references).

See Also

grgkw (parent distribution gradient), llmc (negative log-likelihood for Mc), hsmc (Hessian for Mc, if available), dmc (density for Mc), optim, grad (for numerical gradient comparison), digamma.

Examples

```
## Example 1: Basic Examples

# Generate sample data with more stable parameters
set.seed(123)
n <- 1000
true_params <- c(gamma = 2.0, delta = 2.5, lambda = 1.5)
data <- rmc(n, gamma = true_params[1], delta = true_params[2],
            lambda = true_params[3])

# Evaluate Hessian at true parameters
hess_true <- hsmc(par = true_params, data = data)
cat("Hessian matrix at true parameters:\n")
print(hess_true, digits = 4)

# Check symmetry
cat("\nSymmetry check (max |H - H^T|):",
    max(abs(hess_true - t(hess_true))), "\n")


## Example 2: Hessian Properties at MLE

# Fit model
fit <- optim(
  par = c(1.5, 2.0, 1.0),
  fn = llmc,
  gr = grmc,
  data = data,
  method = "BFGS",
  hessian = TRUE
)

mle <- fit$par
names(mle) <- c("gamma", "delta", "lambda")

# Hessian at MLE
hessian_at_mle <- hsmc(par = mle, data = data)
cat("\nHessian at MLE:\n")
```

```
print(hessian_at_mle, digits = 4)

# Compare with optim's numerical Hessian
cat("\nComparison with optim Hessian:\n")
cat("Max absolute difference:",
    max(abs(hessian_at_mle - fit$hessian)), "\n")

# Eigenvalue analysis
eigenvals <- eigen(hessian_at_mle, only.values = TRUE)$values
cat("\nEigenvalues:\n")
print(eigenvals)

cat("\nPositive definite:", all(eigenvals > 0), "\n")
cat("Condition number:", max(eigenvals) / min(eigenvals), "\n")


## Example 3: Standard Errors and Confidence Intervals

# Observed information matrix
obs_info <- hessian_at_mle

# Variance-covariance matrix
vcov_matrix <- solve(obs_info)
cat("\nVariance-Covariance Matrix:\n")
print(vcov_matrix, digits = 6)

# Standard errors
se <- sqrt(diag(vcov_matrix))
names(se) <- c("gamma", "delta", "lambda")

# Correlation matrix
corr_matrix <- cov2cor(vcov_matrix)
cat("\nCorrelation Matrix:\n")
print(corr_matrix, digits = 4)

# Confidence intervals
z_crit <- qnorm(0.975)
results <- data.frame(
  Parameter = c("gamma", "delta", "lambda"),
  True = true_params,
  MLE = mle,
  SE = se,
  CI_Lower = mle - z_crit * se,
  CI_Upper = mle + z_crit * se
)
print(results, digits = 4)


## Example 4: Determinant and Trace Analysis

# Compute at different points
test_params <- rbind(
  c(1.5, 2.0, 1.0),
  c(2.0, 2.5, 1.5),
  mle,
  c(2.5, 3.0, 2.0)
)

hess_properties <- data.frame(
  Gamma = numeric(),
  Delta = numeric(),
  Lambda = numeric(),
  Determinant = numeric(),
  Trace = numeric(),
  Min_Eigenval = numeric(),
  Max_Eigenval = numeric(),
  Cond_Number = numeric(),
  stringsAsFactors = FALSE
)

for (i in 1:nrow(test_params)) {
  H <- hsmc(par = test_params[i, ], data = data)
  eigs <- eigen(H, only.values = TRUE)$values

  hess_properties <- rbind(hess_properties, data.frame(
    Gamma = test_params[i, 1],
    Delta = test_params[i, 2],
    Lambda = test_params[i, 3],
    Determinant = det(H),
    Trace = sum(diag(H)),
    Min_Eigenval = min(eigs),
    Max_Eigenval = max(eigs),
    Cond_Number = max(eigs) / min(eigs)
  ))
}

cat("\nHessian Properties at Different Points:\n")
print(hess_properties, digits = 4, row.names = FALSE)


## Example 5: Curvature Visualization (All pairs side by side)

# Create grids around MLE with wider range (±1.5)
gamma_grid <- seq(mle[1] - 1.5, mle[1] + 1.5, length.out = 25)
delta_grid <- seq(mle[2] - 1.5, mle[2] + 1.5, length.out = 25)
lambda_grid <- seq(mle[3] - 1.5, mle[3] + 1.5, length.out = 25)

gamma_grid <- gamma_grid[gamma_grid > 0]
delta_grid <- delta_grid[delta_grid > 0]
lambda_grid <- lambda_grid[lambda_grid > 0]

# Compute curvature measures for all pairs
determinant_surface_gd <- matrix(NA, nrow = length(gamma_grid), ncol = length(delta_grid))
trace_surface_gd <- matrix(NA, nrow = length(gamma_grid), ncol = length(delta_grid))

determinant_surface_gl <- matrix(NA, nrow = length(gamma_grid), ncol = length(lambda_grid))
```

```
trace_surface_gl <- matrix(NA, nrow = length(gamma_grid), ncol = length(lambda_grid))

determinant_surface_dl <- matrix(NA, nrow = length(delta_grid), ncol = length(lambda_grid))
trace_surface_dl <- matrix(NA, nrow = length(delta_grid), ncol = length(lambda_grid))

# Gamma vs Delta
for (i in seq_along(gamma_grid)) {
  for (j in seq_along(delta_grid)) {
    H <- hsmc(c(gamma_grid[i], delta_grid[j], mle[3]), data)
    determinant_surface_gd[i, j] <- det(H)
    trace_surface_gd[i, j] <- sum(diag(H))
  }
}

# Gamma vs Lambda
for (i in seq_along(gamma_grid)) {
  for (j in seq_along(lambda_grid)) {
    H <- hsmc(c(gamma_grid[i], mle[2], lambda_grid[j]), data)
    determinant_surface_gl[i, j] <- det(H)
    trace_surface_gl[i, j] <- sum(diag(H))
  }
}

# Delta vs Lambda
for (i in seq_along(delta_grid)) {
  for (j in seq_along(lambda_grid)) {
    H <- hsmc(c(mle[1], delta_grid[i], lambda_grid[j]), data)
    determinant_surface_dl[i, j] <- det(H)
    trace_surface_dl[i, j] <- sum(diag(H))
  }
}

# Plot

# Determinant plots
contour(gamma_grid, delta_grid, determinant_surface_gd,
        xlab = expression(gamma), ylab = expression(delta),
        main = "Determinant: Gamma vs Delta", las = 1,
        col = "#2E4057", lwd = 1.5, nlevels = 15)
points(mle[1], mle[2], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[2], pch = 17, col = "#006400", cex = 1.5)
grid(col = "gray90")

contour(gamma_grid, lambda_grid, determinant_surface_gl,
        xlab = expression(gamma), ylab = expression(lambda),
        main = "Determinant: Gamma vs Lambda", las = 1,
        col = "#2E4057", lwd = 1.5, nlevels = 15)
points(mle[1], mle[3], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[3], pch = 17, col = "#006400", cex = 1.5)
grid(col = "gray90")

contour(delta_grid, lambda_grid, determinant_surface_dl,
        xlab = expression(delta), ylab = expression(lambda),
        main = "Determinant: Delta vs Lambda", las = 1,
        col = "#2E4057", lwd = 1.5, nlevels = 15)
points(mle[2], mle[3], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[2], true_params[3], pch = 17, col = "#006400", cex = 1.5)
grid(col = "gray90")

# Trace plots
contour(gamma_grid, delta_grid, trace_surface_gd,
        xlab = expression(gamma), ylab = expression(delta),
        main = "Trace: Gamma vs Delta", las = 1,
        col = "#2E4057", lwd = 1.5, nlevels = 15)
points(mle[1], mle[2], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[2], pch = 17, col = "#006400", cex = 1.5)
grid(col = "gray90")

contour(gamma_grid, lambda_grid, trace_surface_gl,
        xlab = expression(gamma), ylab = expression(lambda),
        main = "Trace: Gamma vs Lambda", las = 1,
        col = "#2E4057", lwd = 1.5, nlevels = 15)
points(mle[1], mle[3], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[3], pch = 17, col = "#006400", cex = 1.5)
grid(col = "gray90")

contour(delta_grid, lambda_grid, trace_surface_dl,
        xlab = expression(delta), ylab = expression(lambda),
        main = "Trace: Delta vs Lambda", las = 1,
        col = "#2E4057", lwd = 1.5, nlevels = 15)
points(mle[2], mle[3], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[2], true_params[3], pch = 17, col = "#006400", cex = 1.5)
grid(col = "gray90")

legend("topright",
       legend = c("MLE", "True"),
       col = c("#8B0000", "#006400"),
       pch = c(19, 17),
       bty = "n", cex = 0.8)


## Example 6: Confidence Ellipses (All pairs side by side)

# Extract all 2x2 submatrices
vcov_gd <- vcov_matrix[1:2, 1:2]
vcov_gl <- vcov_matrix[c(1, 3), c(1, 3)]
vcov_dl <- vcov_matrix[2:3, 2:3]

# Create confidence ellipses
theta <- seq(0, 2 * pi, length.out = 100)
chi2_val <- qchisq(0.95, df = 2)

# Gamma vs Delta ellipse
eig_decomp_gd <- eigen(vcov_gd)
ellipse_gd <- matrix(NA, nrow = 100, ncol = 2)
for (i in 1:100) {
  v <- c(cos(theta[i]), sin(theta[i]))
  ellipse_gd[i, ] <- mle[1:2] + sqrt(chi2_val) *
    (eig_decomp_gd$vectors %*% diag(sqrt(eig_decomp_gd$values)) %*% v)
}

# Gamma vs Lambda ellipse
eig_decomp_gl <- eigen(vcov_gl)
ellipse_gl <- matrix(NA, nrow = 100, ncol = 2)
for (i in 1:100) {
  v <- c(cos(theta[i]), sin(theta[i]))
  ellipse_gl[i, ] <- mle[c(1, 3)] + sqrt(chi2_val) *
    (eig_decomp_gl$vectors %*% diag(sqrt(eig_decomp_gl$values)) %*% v)
}

# Delta vs Lambda ellipse
eig_decomp_dl <- eigen(vcov_dl)
ellipse_dl <- matrix(NA, nrow = 100, ncol = 2)
for (i in 1:100) {
  v <- c(cos(theta[i]), sin(theta[i]))
  ellipse_dl[i, ] <- mle[2:3] + sqrt(chi2_val) *
    (eig_decomp_dl$vectors %*% diag(sqrt(eig_decomp_dl$values)) %*% v)
}

# Marginal confidence intervals
se_gd <- sqrt(diag(vcov_gd))
ci_gamma_gd <- mle[1] + c(-1, 1) * 1.96 * se_gd[1]
ci_delta_gd <- mle[2] + c(-1, 1) * 1.96 * se_gd[2]

se_gl <- sqrt(diag(vcov_gl))
ci_gamma_gl <- mle[1] + c(-1, 1) * 1.96 * se_gl[1]
ci_lambda_gl <- mle[3] + c(-1, 1) * 1.96 * se_gl[2]

se_dl <- sqrt(diag(vcov_dl))
ci_delta_dl <- mle[2] + c(-1, 1) * 1.96 * se_dl[1]
ci_lambda_dl <- mle[3] + c(-1, 1) * 1.96 * se_dl[2]

# Plot

# Gamma vs Delta
plot(ellipse_gd[, 1], ellipse_gd[, 2], type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(gamma), ylab = expression(delta),
     main = "Gamma vs Delta", las = 1, xlim = range(ellipse_gd[, 1], ci_gamma_gd),
     ylim = range(ellipse_gd[, 2], ci_delta_gd))
abline(v = ci_gamma_gd, col = "#808080", lty = 3, lwd = 1.5)
abline(h = ci_delta_gd, col = "#808080", lty = 3, lwd = 1.5)
points(mle[1], mle[2], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[2], pch = 17, col = "#006400", cex = 1.5)
grid(col = "gray90")

# Gamma vs Lambda
plot(ellipse_gl[, 1], ellipse_gl[, 2], type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(gamma), ylab = expression(lambda),
     main = "Gamma vs Lambda", las = 1, xlim = range(ellipse_gl[, 1], ci_gamma_gl),
     ylim = range(ellipse_gl[, 2], ci_lambda_gl))
abline(v = ci_gamma_gl, col = "#808080", lty = 3, lwd = 1.5)
abline(h = ci_lambda_gl, col = "#808080", lty = 3, lwd = 1.5)
points(mle[1], mle[3], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[3], pch = 17, col = "#006400", cex = 1.5)
grid(col = "gray90")

# Delta vs Lambda
plot(ellipse_dl[, 1], ellipse_dl[, 2], type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(delta), ylab = expression(lambda),
     main = "Delta vs Lambda", las = 1, xlim = range(ellipse_dl[, 1], ci_delta_dl),
     ylim = range(ellipse_dl[, 2], ci_lambda_dl))
abline(v = ci_delta_dl, col = "#808080", lty = 3, lwd = 1.5)
abline(h = ci_lambda_dl, col = "#808080", lty = 3, lwd = 1.5)
points(mle[2], mle[3], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[2], true_params[3], pch = 17, col = "#006400", cex = 1.5)
grid(col = "gray90")

legend("topright",
       legend = c("MLE", "True", "95% CR", "Marginal 95% CI"),
       col = c("#8B0000", "#006400", "#2E4057", "#808080"),
       pch = c(19, 17, NA, NA),
       lty = c(NA, NA, 1, 3),
       lwd = c(NA, NA, 2, 1.5),
       bty = "n", cex = 0.8)
```

hsbeta *Hessian Matrix of the Negative Log-Likelihood for the Beta Distribution (gamma, delta+1 Parameterization)*

Description

Computes the analytic 2x2 Hessian matrix (matrix of second partial derivatives) of the negative log-likelihood function for the standard Beta distribution, using a parameterization common in generalized distribution families. The distribution is parameterized by `gamma` (γ) and `delta` (δ), corresponding to the standard Beta distribution with shape parameters `shape1 = gamma` and `shape2 = delta + 1`. The Hessian is useful for estimating standard errors and in optimization algorithms.

Usage

```
hsbeta(par, data)
```

Arguments

`par`	A numeric vector of length 2 containing the distribution parameters in the order: `gamma` ($\gamma > 0$), `delta` ($\delta \geq 0$).
`data`	A numeric vector of observations. All values must be strictly between 0 and 1 (exclusive).

Details

This function calculates the analytic second partial derivatives of the negative log-likelihood function ($-\ell(\theta|\mathbf{x})$) for a Beta distribution with parameters `shape1 = gamma` (γ) and `shape2 = delta + 1` ($\delta + 1$). The components of the Hessian matrix ($-\mathbf{H}(\theta)$) are:

$$-\frac{\partial^2 \ell}{\partial \gamma^2} = n[\psi'(\gamma) - \psi'(\gamma + \delta + 1)]$$

$$-\frac{\partial^2 \ell}{\partial \gamma \partial \delta} = -n\psi'(\gamma + \delta + 1)$$

$$-\frac{\partial^2 \ell}{\partial \delta^2} = n[\psi'(\delta + 1) - \psi'(\gamma + \delta + 1)]$$

where $\psi'(\cdot)$ is the trigamma function (`trigamma`). These formulas represent the second derivatives of $-\ell(\theta)$, consistent with minimizing the negative log-likelihood. They correspond to the relevant 2x2 submatrix of the general GKw Hessian (`hsgkw`) evaluated at $\alpha = 1, \beta = 1, \lambda = 1$. Note the parameterization difference from the standard Beta distribution (`shape2 = delta + 1`).

The returned matrix is symmetric.

Value

Returns a 2x2 numeric matrix representing the Hessian matrix of the negative log-likelihood function, $-\partial^2 \ell / (\partial \theta_i \partial \theta_j)$, where $\theta = (\gamma, \delta)$. Returns a 2x2 matrix populated with `NaN` if any parameter values are invalid according to their constraints, or if any value in `data` is not in the interval (0, 1).

Author(s)

Lopes, J. E.

References

Johnson, N. L., Kotz, S., & Balakrishnan, N. (1995). *Continuous Univariate Distributions, Volume 2* (2nd ed.). Wiley.

Cordeiro, G. M., & de Castro, M. (2011). A new family of generalized distributions. *Journal of Statistical Computation and Simulation*,

(Note: Specific Hessian formulas might be derived or sourced from additional references).

See Also

`hsgkw`, `hsmc` (related Hessians), `llbeta` (negative log-likelihood function), `grbeta` (gradient, if available), `dbeta_`, `pbeta_`, `qbeta_`, `rbeta_`, `optim`, `hessian` (for numerical Hessian comparison), `trigamma`.

Examples

```
## Example 1: Basic Hessian Evaluation

# Generate sample data
set.seed(123)
n <- 1000
true_params <- c(gamma = 2.0, delta = 3.0)
data <- rbeta_(n, gamma = true_params[1], delta = true_params[2])

# Evaluate Hessian at true parameters
hess_true <- hsbeta(par = true_params, data = data)
cat("Hessian matrix at true parameters:\n")
print(hess_true, digits = 4)

# Check symmetry
cat("\nSymmetry check (max |H - H^T|):",
    max(abs(hess_true - t(hess_true))), "\n")


## Example 2: Hessian Properties at MLE

# Fit model
fit <- optim(
  par = c(1.5, 2.5),
  fn = llbeta,
  gr = grbeta,
  data = data,
  method = "L-BFGS-B",
  lower = c(0.01, 0.01),
  upper = c(100, 100),
  hessian = TRUE
)

mle <- fit$par
names(mle) <- c("gamma", "delta")

# Hessian at MLE
hessian_at_mle <- hsbeta(par = mle, data = data)
cat("\nHessian at MLE:\n")
print(hessian_at_mle, digits = 4)

# Compare with optim's numerical Hessian
cat("\nComparison with optim Hessian:\n")
cat("Max absolute difference:",
    max(abs(hessian_at_mle - fit$hessian)), "\n")

# Eigenvalue analysis
eigenvals <- eigen(hessian_at_mle, only.values = TRUE)$values
cat("\nEigenvalues:\n")
print(eigenvals)

cat("\nPositive definite:", all(eigenvals > 0), "\n")
cat("Condition number:", max(eigenvals) / min(eigenvals), "\n")


## Example 3: Standard Errors and Confidence Intervals

# Observed information matrix
obs_info <- hessian_at_mle

# Variance-covariance matrix
vcov_matrix <- solve(obs_info)
cat("\nVariance-Covariance Matrix:\n")
print(vcov_matrix, digits = 6)

# Standard errors
se <- sqrt(diag(vcov_matrix))
names(se) <- c("gamma", "delta")

# Correlation matrix
corr_matrix <- cov2cor(vcov_matrix)
cat("\nCorrelation Matrix:\n")
print(corr_matrix, digits = 4)

# Confidence intervals
z_crit <- qnorm(0.975)
results <- data.frame(
  Parameter = c("gamma", "delta"),
  True = true_params,
  MLE = mle,
  SE = se,
  CI_Lower = mle - z_crit * se,
  CI_Upper = mle + z_crit * se
)
print(results, digits = 4)

cat(sprintf("\nMLE corresponds approx to Beta(%.2f, %.2f)\n",
    mle[1], mle[2] + 1))
cat("True corresponds to Beta(%.2f, %.2f)\n",
    true_params[1], true_params[2] + 1)


## Example 4: Determinant and Trace Analysis

# Compute at different points
test_params <- rbind(
  c(1.5, 2.5),
  c(2.0, 3.0),
  mle,
  c(2.5, 3.5)
)

hess_properties <- data.frame(
  Gamma = numeric(),
  Delta = numeric(),
  Determinant = numeric(),
  Trace = numeric(),
  Min_Eigenval = numeric(),
  Max_Eigenval = numeric(),
  Cond_Number = numeric(),
  stringsAsFactors = FALSE
)

for (i in 1:nrow(test_params)) {
  H <- hsbeta(par = test_params[i, ], data = data)
  eigs <- eigen(H, only.values = TRUE)$values

  hess_properties <- rbind(hess_properties, data.frame(
    Gamma = test_params[i, 1],
    Delta = test_params[i, 2],
    Determinant = det(H),
    Trace = sum(diag(H)),
    Min_Eigenval = min(eigs),
    Max_Eigenval = max(eigs),
    Cond_Number = max(eigs) / min(eigs)
  ))
}

cat("\nHessian Properties at Different Points:\n")
print(hess_properties, digits = 4, row.names = FALSE)


## Example 5: Curvature Visualization (Gamma vs Delta)

# Create grid around MLE
gamma_grid <- seq(mle[1] - 1.5, mle[1] + 1.5, length.out = 25)
delta_grid <- seq(mle[2] - 1.5, mle[2] + 1.5, length.out = 25)
gamma_grid <- gamma_grid[gamma_grid > 0]
delta_grid <- delta_grid[delta_grid > 0]

# Compute curvature measures
determinant_surface <- matrix(NA, nrow = length(gamma_grid),
                               ncol = length(delta_grid))
trace_surface <- matrix(NA, nrow = length(gamma_grid),
                         ncol = length(delta_grid))

for (i in seq_along(gamma_grid)) {
  for (j in seq_along(delta_grid)) {
    H <- hsbeta(c(gamma_grid[i], delta_grid[j]), data)
    determinant_surface[i, j] <- det(H)
    trace_surface[i, j] <- sum(diag(H))
  }
}

# Plot

contour(gamma_grid, delta_grid, determinant_surface,
        xlab = expression(gamma), ylab = expression(delta),
        main = "Hessian Determinant", las = 1,
        col = "#2E4057", lwd = 1.5, nlevels = 15)
points(mle[1], mle[2], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[2], pch = 17, col = "#006400", cex = 1.5)
grid(col = "gray90")

contour(gamma_grid, delta_grid, trace_surface,
        xlab = expression(gamma), ylab = expression(delta),
        main = "Hessian Trace", las = 1,
        col = "#2E4057", lwd = 1.5, nlevels = 15)
points(mle[1], mle[2], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[2], pch = 17, col = "#006400", cex = 1.5)
grid(col = "gray90")


## Example 6: Confidence Ellipse (Gamma vs Delta)

# Extract 2x2 submatrix (full matrix in this case)
vcov_2d <- vcov_matrix

# Create confidence ellipse
theta <- seq(0, 2 * pi, length.out = 100)
chi2_val <- qchisq(0.95, df = 2)

eig_decomp <- eigen(vcov_2d)
ellipse <- matrix(NA, nrow = 100, ncol = 2)
for (i in 1:100) {
  v <- c(cos(theta[i]), sin(theta[i]))
  ellipse[i, ] <- mle + sqrt(chi2_val) *
    (eig_decomp$vectors %*% diag(sqrt(eig_decomp$values)) %*% v)
}

# Marginal confidence intervals
se_2d <- sqrt(diag(vcov_2d))
ci_gamma <- mle[1] + c(-1, 1) * 1.96 * se_2d[1]
ci_delta <- mle[2] + c(-1, 1) * 1.96 * se_2d[2]

# Plot

plot(ellipse[, 1], ellipse[, 2], type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(gamma), ylab = expression(delta),
     main = "95% Confidence Ellipse (Gamma vs Delta)", las = 1)

# Add marginal CIs
abline(v = ci_gamma, col = "#808080", lty = 3, lwd = 1.5)
abline(h = ci_delta, col = "#808080", lty = 3, lwd = 1.5)

points(mle[1], mle[2], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[2], pch = 17, col = "#006400", cex = 1.5)

legend("topright",
       legend = c("MLE", "True", "95% CR", "Marginal 95% CI"),
       col = c("#8B0000", "#006400", "#2E4057", "#808080"),
       pch = c(19, 17, NA, NA),
       lty = c(NA, NA, 1, 3),
       lwd = c(NA, NA, 2, 1.5),
       bty = "n")
grid(col = "gray90")
```

---

hsbkw *Hessian Matrix of the Negative Log-Likelihood for the BKw Distribution*

Description

Computes the analytic 4x4 Hessian matrix (matrix of second partial derivatives) of the negative log-likelihood function for the Beta-Kumaraswamy (BKw) distribution with parameters `alpha` (α), `beta` (β), `gamma` (γ), and `delta` (δ). This distribution is the special case of the Generalized Kumaraswamy (GKw) distribution where $\lambda = 1$. The Hessian is useful for estimating standard errors and in optimization algorithms.

Usage

```
hsbkw(par, data)
```

Arguments

`par`	A numeric vector of length 4 containing the distribution parameters in the order: `alpha` ($\alpha > 0$), `beta` ($\beta > 0$), `gamma` ($\gamma > 0$), `delta` ($\delta \geq 0$).
`data`	A numeric vector of observations. All values must be strictly between 0 and 1 (exclusive).

Details

This function calculates the analytic second partial derivatives of the negative log-likelihood function based on the BKw log-likelihood ($\lambda = 1$ case of GKw, see `llbkw`):

$$\ell(\theta|\mathbf{x}) = n[\ln(\alpha)+\ln(\beta)-\ln B(\gamma,\delta+1)]+\sum_{i=1}^{n}[(\alpha-1)\ln(x_i)+(\beta(\delta+1)-1)\ln(v_i)+(\gamma-1)\ln(w_i)]$$

where $\theta = (\alpha, \beta, \gamma, \delta)$, $B(a, b)$ is the Beta function (`beta`), and intermediate terms are:

- $v_i = 1 - x_i^{\alpha}$
- $w_i = 1 - v_i^{\beta} = 1 - (1 - x_i^{\alpha})^{\beta}$

The Hessian matrix returned contains the elements $-\frac{\partial^2 \ell(\theta|\mathbf{x})}{\partial\theta_i \partial\theta_j}$ for $\theta_i, \theta_j \in \{\alpha, \beta, \gamma, \delta\}$.

Key properties of the returned matrix:

- Dimensions: 4x4.
- Symmetry: The matrix is symmetric.
- Ordering: Rows and columns correspond to the parameters in the order $\alpha, \beta, \gamma, \delta$.
- Content: Analytic second derivatives of the *negative* log-likelihood.

This corresponds to the relevant 4x4 submatrix of the 5x5 GKw Hessian (hsgkw) evaluated at $\lambda = 1$. The exact analytical formulas are implemented directly.

Value

Returns a 4x4 numeric matrix representing the Hessian matrix of the negative log-likelihood function, $-\partial^2\ell/(\partial\theta_i\partial\theta_j)$, where $\theta = (\alpha, \beta, \gamma, \delta)$. Returns a 4x4 matrix populated with `NaN` if any parameter values are invalid according to their constraints, or if any value in `data` is not in the interval (0, 1).

Author(s)

Lopes, J. E.

References

Cordeiro, G. M., & de Castro, M. (2011). A new family of generalized distributions. *Journal of Statistical Computation and Simulation*,

Kumaraswamy, P. (1980). A generalized probability density function for double-bounded random processes. *Journal of Hydrology*, *46*(1-2), 79-88.

(Note: Specific Hessian formulas might be derived or sourced from additional references).

See Also

hsgkw (parent distribution Hessian), llbkw (negative log-likelihood for BKw), grbkw (gradient for BKw, if available), dbkw (density for BKw), optim, hessian (for numerical Hessian comparison).

Examples

```
## Example 1: Basic Hessian Evaluation
# Generate sample data
set.seed(2203)
n <- 1000
true_params <- c(alpha = 2.0, beta = 1.5, gamma = 1.5, delta = 0.5)
data <- rbkw(n, alpha = true_params[1], beta = true_params[2],
             gamma = true_params[3], delta = true_params[4])

# Evaluate Hessian at true parameters
hess_true <- hsbkw(par = true_params, data = data)
cat("Hessian matrix at true parameters:\n")
print(hess_true, digits = 4)

# Check symmetry
cat("\nSymmetry check (max |H - H^T|):",
    max(abs(hess_true - t(hess_true))), "\n")
```

```
## Example 2: Hessian Properties at MLE

# Fit model
fit <- optim(
  par = c(1.8, 1.2, 1.1, 0.3),
  fn = llbkw,
  gr = grbkw,
  data = data,
  method = "Nelder-Mead",
  hessian = TRUE
)

mle <- fit$par
names(mle) <- c("alpha", "beta", "gamma", "delta")

# Hessian at MLE
hessian_at_mle <- hsbkw(par = mle, data = data)
cat("\nHessian at MLE:\n")
print(hessian_at_mle, digits = 4)

# Compare with optim's numerical Hessian
cat("\nComparison with optim Hessian:\n")
cat("Max absolute difference:",
    max(abs(hessian_at_mle - fit$hessian)), "\n")

# Eigenvalue analysis
eigenvals <- eigen(hessian_at_mle, only.values = TRUE)$values
cat("\nEigenvalues:\n")
print(eigenvals)

cat("\nPositive definite:", all(eigenvals > 0), "\n")
cat("Condition number:", max(eigenvals) / min(eigenvals), "\n")


## Example 3: Standard Errors and Confidence Intervals

# Observed information matrix
obs_info <- hessian_at_mle

# Variance-covariance matrix
vcov_matrix <- solve(obs_info)
cat("\nVariance-Covariance Matrix:\n")
print(vcov_matrix, digits = 6)

# Standard errors
se <- sqrt(diag(vcov_matrix))
names(se) <- c("alpha", "beta", "gamma", "delta")

# Correlation matrix
corr_matrix <- cov2cor(vcov_matrix)
cat("\nCorrelation Matrix:\n")
```

```
print(corr_matrix, digits = 4)

# Confidence intervals
z_crit <- qnorm(0.975)
results <- data.frame(
  Parameter = c("alpha", "beta", "gamma", "delta"),
  True = true_params,
  MLE = mle,
  SE = se,
  CI_Lower = mle - z_crit * se,
  CI_Upper = mle + z_crit * se
)
print(results, digits = 4)


## Example 4: Determinant and Trace Analysis

# Compute at different points
test_params <- rbind(
  c(1.5, 1.0, 1.0, 0.3),
  c(2.0, 1.5, 1.5, 0.5),
  mle,
  c(2.5, 2.0, 2.0, 0.7)
)

hess_properties <- data.frame(
  Alpha = numeric(),
  Beta = numeric(),
  Gamma = numeric(),
  Delta = numeric(),
  Determinant = numeric(),
  Trace = numeric(),
  Min_Eigenval = numeric(),
  Max_Eigenval = numeric(),
  Cond_Number = numeric(),
  stringsAsFactors = FALSE
)

for (i in 1:nrow(test_params)) {
  H <- hsbkw(par = test_params[i, ], data = data)
  eigs <- eigen(H, only.values = TRUE)$values

  hess_properties <- rbind(hess_properties, data.frame(
    Alpha = test_params[i, 1],
    Beta = test_params[i, 2],
    Gamma = test_params[i, 3],
    Delta = test_params[i, 4],
    Determinant = det(H),
    Trace = sum(diag(H)),
    Min_Eigenval = min(eigs),
    Max_Eigenval = max(eigs),
    Cond_Number = max(eigs) / min(eigs)
  ))
```

```
}

cat("\nHessian Properties at Different Points:\n")
print(hess_properties, digits = 4, row.names = FALSE)


## Example 5: Curvature Visualization (Selected pairs)

# Create grids around MLE with wider range (±1.5)
alpha_grid <- seq(mle[1] - 1.5, mle[1] + 1.5, length.out = 25)
beta_grid <- seq(mle[2] - 1.5, mle[2] + 1.5, length.out = 25)
gamma_grid <- seq(mle[3] - 1.5, mle[3] + 1.5, length.out = 25)
delta_grid <- seq(mle[4] - 1.5, mle[4] + 1.5, length.out = 25)

alpha_grid <- alpha_grid[alpha_grid > 0]
beta_grid <- beta_grid[beta_grid > 0]
gamma_grid <- gamma_grid[gamma_grid > 0]
delta_grid <- delta_grid[delta_grid > 0]

# Compute curvature measures for selected pairs
determinant_surface_ab <- matrix(NA, nrow = length(alpha_grid), ncol = length(beta_grid))
trace_surface_ab <- matrix(NA, nrow = length(alpha_grid), ncol = length(beta_grid))

determinant_surface_ag <- matrix(NA, nrow = length(alpha_grid), ncol = length(gamma_grid))
trace_surface_ag <- matrix(NA, nrow = length(alpha_grid), ncol = length(gamma_grid))

determinant_surface_bd <- matrix(NA, nrow = length(beta_grid), ncol = length(delta_grid))
trace_surface_bd <- matrix(NA, nrow = length(beta_grid), ncol = length(delta_grid))

# Alpha vs Beta
for (i in seq_along(alpha_grid)) {
  for (j in seq_along(beta_grid)) {
    H <- hsbkw(c(alpha_grid[i], beta_grid[j], mle[3], mle[4]), data)
    determinant_surface_ab[i, j] <- det(H)
    trace_surface_ab[i, j] <- sum(diag(H))
  }
}

# Alpha vs Gamma
for (i in seq_along(alpha_grid)) {
  for (j in seq_along(gamma_grid)) {
    H <- hsbkw(c(alpha_grid[i], mle[2], gamma_grid[j], mle[4]), data)
    determinant_surface_ag[i, j] <- det(H)
    trace_surface_ag[i, j] <- sum(diag(H))
  }
}

# Beta vs Delta
for (i in seq_along(beta_grid)) {
  for (j in seq_along(delta_grid)) {
    H <- hsbkw(c(mle[1], beta_grid[i], mle[3], delta_grid[j]), data)
    determinant_surface_bd[i, j] <- det(H)
    trace_surface_bd[i, j] <- sum(diag(H))
  }
}

# Plot selected curvature surfaces

# Determinant plots
contour(alpha_grid, beta_grid, determinant_surface_ab,
        xlab = expression(alpha), ylab = expression(beta),
        main = "Determinant: Alpha vs Beta", las = 1,
        col = "#2E4057", lwd = 1.5, nlevels = 15)
points(mle[1], mle[2], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[2], pch = 17, col = "#006400", cex = 1.5)
grid(col = "gray90")

contour(alpha_grid, gamma_grid, determinant_surface_ag,
        xlab = expression(alpha), ylab = expression(gamma),
        main = "Determinant: Alpha vs Gamma", las = 1,
        col = "#2E4057", lwd = 1.5, nlevels = 15)
points(mle[1], mle[3], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[3], pch = 17, col = "#006400", cex = 1.5)
grid(col = "gray90")

contour(beta_grid, delta_grid, determinant_surface_bd,
        xlab = expression(beta), ylab = expression(delta),
        main = "Determinant: Beta vs Delta", las = 1,
        col = "#2E4057", lwd = 1.5, nlevels = 15)
points(mle[2], mle[4], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[2], true_params[4], pch = 17, col = "#006400", cex = 1.5)
grid(col = "gray90")

# Trace plots
contour(alpha_grid, beta_grid, trace_surface_ab,
        xlab = expression(alpha), ylab = expression(beta),
        main = "Trace: Alpha vs Beta", las = 1,
        col = "#2E4057", lwd = 1.5, nlevels = 15)
points(mle[1], mle[2], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[2], pch = 17, col = "#006400", cex = 1.5)
grid(col = "gray90")

contour(alpha_grid, gamma_grid, trace_surface_ag,
        xlab = expression(alpha), ylab = expression(gamma),
        main = "Trace: Alpha vs Gamma", las = 1,
        col = "#2E4057", lwd = 1.5, nlevels = 15)
points(mle[1], mle[3], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[3], pch = 17, col = "#006400", cex = 1.5)
grid(col = "gray90")

contour(beta_grid, delta_grid, trace_surface_bd,
        xlab = expression(beta), ylab = expression(delta),
        main = "Trace: Beta vs Delta", las = 1,
        col = "#2E4057", lwd = 1.5, nlevels = 15)
points(mle[2], mle[4], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[2], true_params[4], pch = 17, col = "#006400", cex = 1.5)
```

```
grid(col = "gray90")

legend("topright",
       legend = c("MLE", "True"),
       col = c("#8B0000", "#006400"),
       pch = c(19, 17),
       bty = "n", cex = 0.8)


## Example 6: Confidence Ellipses (Selected pairs)

# Extract selected 2x2 submatrices
vcov_ab <- vcov_matrix[1:2, 1:2]
vcov_ag <- vcov_matrix[c(1, 3), c(1, 3)]
vcov_bd <- vcov_matrix[c(2, 4), c(2, 4)]

# Create confidence ellipses
theta <- seq(0, 2 * pi, length.out = 100)
chi2_val <- qchisq(0.95, df = 2)

# Alpha vs Beta ellipse
eig_decomp_ab <- eigen(vcov_ab)
ellipse_ab <- matrix(NA, nrow = 100, ncol = 2)
for (i in 1:100) {
  v <- c(cos(theta[i]), sin(theta[i]))
  ellipse_ab[i, ] <- mle[1:2] + sqrt(chi2_val) *
    (eig_decomp_ab$vectors %*% diag(sqrt(eig_decomp_ab$values)) %*% v)
}

# Alpha vs Gamma ellipse
eig_decomp_ag <- eigen(vcov_ag)
ellipse_ag <- matrix(NA, nrow = 100, ncol = 2)
for (i in 1:100) {
  v <- c(cos(theta[i]), sin(theta[i]))
  ellipse_ag[i, ] <- mle[c(1, 3)] + sqrt(chi2_val) *
    (eig_decomp_ag$vectors %*% diag(sqrt(eig_decomp_ag$values)) %*% v)
}

# Beta vs Delta ellipse
eig_decomp_bd <- eigen(vcov_bd)
ellipse_bd <- matrix(NA, nrow = 100, ncol = 2)
for (i in 1:100) {
  v <- c(cos(theta[i]), sin(theta[i]))
  ellipse_bd[i, ] <- mle[c(2, 4)] + sqrt(chi2_val) *
    (eig_decomp_bd$vectors %*% diag(sqrt(eig_decomp_bd$values)) %*% v)
}

# Marginal confidence intervals
se_ab <- sqrt(diag(vcov_ab))
ci_alpha_ab <- mle[1] + c(-1, 1) * 1.96 * se_ab[1]
ci_beta_ab <- mle[2] + c(-1, 1) * 1.96 * se_ab[2]

se_ag <- sqrt(diag(vcov_ag))
```

```
ci_alpha_ag <- mle[1] + c(-1, 1) * 1.96 * se_ag[1]
ci_gamma_ag <- mle[3] + c(-1, 1) * 1.96 * se_ag[2]

se_bd <- sqrt(diag(vcov_bd))
ci_beta_bd <- mle[2] + c(-1, 1) * 1.96 * se_bd[1]
ci_delta_bd <- mle[4] + c(-1, 1) * 1.96 * se_bd[2]

# Plot selected ellipses side by side

# Alpha vs Beta
plot(ellipse_ab[, 1], ellipse_ab[, 2], type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(alpha), ylab = expression(beta),
     main = "Alpha vs Beta", las = 1, xlim = range(ellipse_ab[, 1], ci_alpha_ab),
     ylim = range(ellipse_ab[, 2], ci_beta_ab))
abline(v = ci_alpha_ab, col = "#808080", lty = 3, lwd = 1.5)
abline(h = ci_beta_ab, col = "#808080", lty = 3, lwd = 1.5)
points(mle[1], mle[2], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[2], pch = 17, col = "#006400", cex = 1.5)
grid(col = "gray90")

# Alpha vs Gamma
plot(ellipse_ag[, 1], ellipse_ag[, 2], type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(alpha), ylab = expression(gamma),
     main = "Alpha vs Gamma", las = 1, xlim = range(ellipse_ag[, 1], ci_alpha_ag),
     ylim = range(ellipse_ag[, 2], ci_gamma_ag))
abline(v = ci_alpha_ag, col = "#808080", lty = 3, lwd = 1.5)
abline(h = ci_gamma_ag, col = "#808080", lty = 3, lwd = 1.5)
points(mle[1], mle[3], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[3], pch = 17, col = "#006400", cex = 1.5)
grid(col = "gray90")

# Beta vs Delta
plot(ellipse_bd[, 1], ellipse_bd[, 2], type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(beta), ylab = expression(delta),
     main = "Beta vs Delta", las = 1, xlim = range(ellipse_bd[, 1], ci_beta_bd),
     ylim = range(ellipse_bd[, 2], ci_delta_bd))
abline(v = ci_beta_bd, col = "#808080", lty = 3, lwd = 1.5)
abline(h = ci_delta_bd, col = "#808080", lty = 3, lwd = 1.5)
points(mle[2], mle[4], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[2], true_params[4], pch = 17, col = "#006400", cex = 1.5)
grid(col = "gray90")

legend("topright",
       legend = c("MLE", "True", "95% CR", "Marginal 95% CI"),
       col = c("#8B0000", "#006400", "#2E4057", "#808080"),
       pch = c(19, 17, NA, NA),
       lty = c(NA, NA, 1, 3),
       lwd = c(NA, NA, 2, 1.5),
       bty = "n", cex = 0.8)
```

`hsekw` *Hessian Matrix of the Negative Log-Likelihood for the EKw Distribution*

Description

Computes the analytic 3x3 Hessian matrix (matrix of second partial derivatives) of the negative log-likelihood function for the Exponentiated Kumaraswamy (EKw) distribution with parameters `alpha` (α), `beta` (β), and `lambda` (λ). This distribution is the special case of the Generalized Kumaraswamy (GKw) distribution where $\gamma = 1$ and $\delta = 0$. The Hessian is useful for estimating standard errors and in optimization algorithms.

Usage

```
hsekw(par, data)
```

Arguments

`par`	A numeric vector of length 3 containing the distribution parameters in the order: `alpha` ($\alpha > 0$), `beta` ($\beta > 0$), `lambda` ($\lambda > 0$).
`data`	A numeric vector of observations. All values must be strictly between 0 and 1 (exclusive).

Details

This function calculates the analytic second partial derivatives of the negative log-likelihood function based on the EKw log-likelihood ($\gamma = 1, \delta = 0$ case of GKw, see `llekw`):

$$\ell(\theta|\mathbf{x}) = n[\ln(\lambda) + \ln(\alpha) + \ln(\beta)] + \sum_{i=1}^{n}[(\alpha - 1)\ln(x_i) + (\beta - 1)\ln(v_i) + (\lambda - 1)\ln(w_i)]$$

where $\theta = (\alpha, \beta, \lambda)$ and intermediate terms are:

- $v_i = 1 - x_i^\alpha$
- $w_i = 1 - v_i^\beta = 1 - (1 - x_i^\alpha)^\beta$

The Hessian matrix returned contains the elements $-\frac{\partial^2 \ell(\theta|\mathbf{x})}{\partial\theta_i \partial\theta_j}$ for $\theta_i, \theta_j \in \{\alpha, \beta, \lambda\}$.

Key properties of the returned matrix:

- Dimensions: 3x3.
- Symmetry: The matrix is symmetric.
- Ordering: Rows and columns correspond to the parameters in the order α, β, λ.
- Content: Analytic second derivatives of the *negative* log-likelihood.

This corresponds to the relevant 3x3 submatrix of the 5x5 GKw Hessian (`hsgkw`) evaluated at $\gamma = 1, \delta = 0$. The exact analytical formulas are implemented directly.

Value

Returns a 3x3 numeric matrix representing the Hessian matrix of the negative log-likelihood function, $-\partial^2 \ell/(\partial\theta_i \partial\theta_j)$, where $\theta = (\alpha, \beta, \lambda)$. Returns a 3x3 matrix populated with `NaN` if any parameter values are invalid according to their constraints, or if any value in `data` is not in the interval (0, 1).

Author(s)

Lopes, J. E.

References

Nadarajah, S., Cordeiro, G. M., & Ortega, E. M. (2012). The exponentiated Kumaraswamy distribution. *Journal of the Franklin Institute*, *349*(3),

Cordeiro, G. M., & de Castro, M. (2011). A new family of generalized distributions. *Journal of Statistical Computation and Simulation*,

Kumaraswamy, P. (1980). A generalized probability density function for double-bounded random processes. *Journal of Hydrology*, *46*(1-2), 79-88.

(Note: Specific Hessian formulas might be derived or sourced from additional references).

See Also

`hsgkw` (parent distribution Hessian), `llekw` (negative log-likelihood for EKw), `grekw` (gradient for EKw, if available), `dekw` (density for EKw), `optim`, `hessian` (for numerical Hessian comparison).

Examples

```
## Example 1: Basic Hessian Evaluation

# Generate sample data
set.seed(123)
n <- 1000
true_params <- c(alpha = 2.5, beta = 3.5, lambda = 2.0)
data <- rekw(n, alpha = true_params[1], beta = true_params[2],
             lambda = true_params[3])

# Evaluate Hessian at true parameters
hess_true <- hsekw(par = true_params, data = data)
cat("Hessian matrix at true parameters:\n")
print(hess_true, digits = 4)

# Check symmetry
cat("\nSymmetry check (max |H - H^T|):",
    max(abs(hess_true - t(hess_true))), "\n")


## Example 2: Hessian Properties at MLE

# Fit model
fit <- optim(
  par = c(2, 3, 1.5),
  fn = llekw,
  gr = grekw,
  data = data,
  method = "BFGS",
  hessian = TRUE
)

mle <- fit$par
names(mle) <- c("alpha", "beta", "lambda")

# Hessian at MLE
hessian_at_mle <- hsekw(par = mle, data = data)
cat("\nHessian at MLE:\n")
print(hessian_at_mle, digits = 4)

# Compare with optim's numerical Hessian
cat("\nComparison with optim Hessian:\n")
cat("Max absolute difference:",
    max(abs(hessian_at_mle - fit$hessian)), "\n")

# Eigenvalue analysis
eigenvals <- eigen(hessian_at_mle, only.values = TRUE)$values
cat("\nEigenvalues:\n")
print(eigenvals)

cat("\nPositive definite:", all(eigenvals > 0), "\n")
cat("Condition number:", max(eigenvals) / min(eigenvals), "\n")


## Example 3: Standard Errors and Confidence Intervals

# Observed information matrix
obs_info <- hessian_at_mle

# Variance-covariance matrix
vcov_matrix <- solve(obs_info)
cat("\nVariance-Covariance Matrix:\n")
print(vcov_matrix, digits = 6)

# Standard errors
se <- sqrt(diag(vcov_matrix))
names(se) <- c("alpha", "beta", "lambda")

# Correlation matrix
corr_matrix <- cov2cor(vcov_matrix)
cat("\nCorrelation Matrix:\n")
print(corr_matrix, digits = 4)

# Confidence intervals
z_crit <- qnorm(0.975)
results <- data.frame(
  Parameter = c("alpha", "beta", "lambda"),
  True = true_params,
  MLE = mle,
  SE = se,
  CI_Lower = mle - z_crit * se,
  CI_Upper = mle + z_crit * se
)
print(results, digits = 4)


## Example 4: Determinant and Trace Analysis

# Compute at different points
test_params <- rbind(
  c(2.0, 3.0, 1.5),
  c(2.5, 3.5, 2.0),
  mle,
  c(3.0, 4.0, 2.5)
)

hess_properties <- data.frame(
  Alpha = numeric(),
  Beta = numeric(),
  Lambda = numeric(),
  Determinant = numeric(),
  Trace = numeric(),
  Min_Eigenval = numeric(),
  Max_Eigenval = numeric(),
  Cond_Number = numeric(),
  stringsAsFactors = FALSE
)

for (i in 1:nrow(test_params)) {
  H <- hsekw(par = test_params[i, ], data = data)
  eigs <- eigen(H, only.values = TRUE)$values

  hess_properties <- rbind(hess_properties, data.frame(
    Alpha = test_params[i, 1],
    Beta = test_params[i, 2],
    Lambda = test_params[i, 3],
    Determinant = det(H),
    Trace = sum(diag(H)),
    Min_Eigenval = min(eigs),
    Max_Eigenval = max(eigs),
    Cond_Number = max(eigs) / min(eigs)
  ))
}

cat("\nHessian Properties at Different Points:\n")
print(hess_properties, digits = 4, row.names = FALSE)


## Example 5: Curvature Visualization (Alpha vs Beta)
```

```
# Create grid around MLE
alpha_grid <- seq(mle[1] - 0.5, mle[1] + 0.5, length.out = 25)
beta_grid <- seq(mle[2] - 0.5, mle[2] + 0.5, length.out = 25)
alpha_grid <- alpha_grid[alpha_grid > 0]
beta_grid <- beta_grid[beta_grid > 0]

# Compute curvature measures
determinant_surface <- matrix(NA, nrow = length(alpha_grid),
                               ncol = length(beta_grid))
trace_surface <- matrix(NA, nrow = length(alpha_grid),
                         ncol = length(beta_grid))

for (i in seq_along(alpha_grid)) {
  for (j in seq_along(beta_grid)) {
    H <- hsekw(c(alpha_grid[i], beta_grid[j], mle[3]), data)
    determinant_surface[i, j] <- det(H)
    trace_surface[i, j] <- sum(diag(H))
  }
}

# Plot

contour(alpha_grid, beta_grid, determinant_surface,
        xlab = expression(alpha), ylab = expression(beta),
        main = "Hessian Determinant", las = 1,
        col = "#2E4057", lwd = 1.5, nlevels = 15)
points(mle[1], mle[2], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[2], pch = 17, col = "#006400", cex = 1.5)
grid(col = "gray90")

contour(alpha_grid, beta_grid, trace_surface,
        xlab = expression(alpha), ylab = expression(beta),
        main = "Hessian Trace", las = 1,
        col = "#2E4057", lwd = 1.5, nlevels = 15)
points(mle[1], mle[2], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[2], pch = 17, col = "#006400", cex = 1.5)
grid(col = "gray90")

## Example 6: Confidence Ellipse (Alpha vs Beta)

# Extract 2x2 submatrix for alpha and beta
vcov_2d <- vcov_matrix[1:2, 1:2]

# Create confidence ellipse
theta <- seq(0, 2 * pi, length.out = 100)
chi2_val <- qchisq(0.95, df = 2)

eig_decomp <- eigen(vcov_2d)
ellipse <- matrix(NA, nrow = 100, ncol = 2)
for (i in 1:100) {
  v <- c(cos(theta[i]), sin(theta[i]))
  ellipse[i, ] <- mle[1:2] + sqrt(chi2_val) *
    (eig_decomp$vectors %*% diag(sqrt(eig_decomp$values)) %*% v)
```

```
}

# Marginal confidence intervals
se_2d <- sqrt(diag(vcov_2d))
ci_alpha <- mle[1] + c(-1, 1) * 1.96 * se_2d[1]
ci_beta <- mle[2] + c(-1, 1) * 1.96 * se_2d[2]

# Plot

plot(ellipse[, 1], ellipse[, 2], type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(alpha), ylab = expression(beta),
     main = "95% Confidence Ellipse (Alpha vs Beta)", las = 1)

# Add marginal CIs
abline(v = ci_alpha, col = "#808080", lty = 3, lwd = 1.5)
abline(h = ci_beta, col = "#808080", lty = 3, lwd = 1.5)

points(mle[1], mle[2], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[2], pch = 17, col = "#006400", cex = 1.5)

legend("topright",
       legend = c("MLE", "True", "95% CR", "Marginal 95% CI"),
       col = c("#8B0000", "#006400", "#2E4057", "#808080"),
       pch = c(19, 17, NA, NA),
       lty = c(NA, NA, 1, 3),
       lwd = c(NA, NA, 2, 1.5),
       bty = "n")
grid(col = "gray90")


## Example 7: Confidence Ellipse (Alpha vs Lambda)

# Extract 2x2 submatrix for alpha and lambda
vcov_2d_al <- vcov_matrix[c(1, 3), c(1, 3)]

# Create confidence ellipse
eig_decomp_al <- eigen(vcov_2d_al)
ellipse_al <- matrix(NA, nrow = 100, ncol = 2)
for (i in 1:100) {
  v <- c(cos(theta[i]), sin(theta[i]))
  ellipse_al[i, ] <- mle[c(1, 3)] + sqrt(chi2_val) *
    (eig_decomp_al$vectors %*% diag(sqrt(eig_decomp_al$values)) %*% v)
}

# Marginal confidence intervals
se_2d_al <- sqrt(diag(vcov_2d_al))
ci_alpha_2 <- mle[1] + c(-1, 1) * 1.96 * se_2d_al[1]
ci_lambda <- mle[3] + c(-1, 1) * 1.96 * se_2d_al[2]

# Plot

plot(ellipse_al[, 1], ellipse_al[, 2], type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(alpha), ylab = expression(lambda),
     main = "95% Confidence Ellipse (Alpha vs Lambda)", las = 1)

# Add marginal CIs
abline(v = ci_alpha_2, col = "#808080", lty = 3, lwd = 1.5)
abline(h = ci_lambda, col = "#808080", lty = 3, lwd = 1.5)

points(mle[1], mle[3], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[3], pch = 17, col = "#006400", cex = 1.5)

legend("topright",
       legend = c("MLE", "True", "95% CR", "Marginal 95% CI"),
       col = c("#8B0000", "#006400", "#2E4057", "#808080"),
       pch = c(19, 17, NA, NA),
       lty = c(NA, NA, 1, 3),
       lwd = c(NA, NA, 2, 1.5),
       bty = "n")
grid(col = "gray90")


## Example 8: Confidence Ellipse (Beta vs Lambda)

# Extract 2x2 submatrix for beta and lambda
vcov_2d_bl <- vcov_matrix[2:3, 2:3]

# Create confidence ellipse
eig_decomp_bl <- eigen(vcov_2d_bl)
ellipse_bl <- matrix(NA, nrow = 100, ncol = 2)
for (i in 1:100) {
  v <- c(cos(theta[i]), sin(theta[i]))
  ellipse_bl[i, ] <- mle[2:3] + sqrt(chi2_val) *
    (eig_decomp_bl$vectors %*% diag(sqrt(eig_decomp_bl$values)) %*% v)
}

# Marginal confidence intervals
se_2d_bl <- sqrt(diag(vcov_2d_bl))
ci_beta_2 <- mle[2] + c(-1, 1) * 1.96 * se_2d_bl[1]
ci_lambda_2 <- mle[3] + c(-1, 1) * 1.96 * se_2d_bl[2]

# Plot

plot(ellipse_bl[, 1], ellipse_bl[, 2], type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(beta), ylab = expression(lambda),
     main = "95% Confidence Ellipse (Beta vs Lambda)", las = 1)

# Add marginal CIs
abline(v = ci_beta_2, col = "#808080", lty = 3, lwd = 1.5)
abline(h = ci_lambda_2, col = "#808080", lty = 3, lwd = 1.5)

points(mle[2], mle[3], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[2], true_params[3], pch = 17, col = "#006400", cex = 1.5)

legend("topright",
       legend = c("MLE", "True", "95% CR", "Marginal 95% CI"),
       col = c("#8B0000", "#006400", "#2E4057", "#808080"),
       pch = c(19, 17, NA, NA),
       lty = c(NA, NA, 1, 3),
       lwd = c(NA, NA, 2, 1.5),
       bty = "n")
grid(col = "gray90")
```

`hsgkw` *Hessian Matrix of the Negative Log-Likelihood for the GKw Distribution*

Description

Computes the analytic Hessian matrix (matrix of second partial derivatives) of the negative log-likelihood function for the five-parameter Generalized Kumaraswamy (GKw) distribution. This is typically used to estimate standard errors of maximum likelihood estimates or in optimization algorithms.

Usage

```
hsgkw(par, data)
```

Arguments

`par`	A numeric vector of length 5 containing the distribution parameters in the order: `alpha` ($\alpha > 0$), `beta` ($\beta > 0$), `gamma` ($\gamma > 0$), `delta` ($\delta \geq 0$), `lambda` ($\lambda > 0$).
`data`	A numeric vector of observations. All values must be strictly between 0 and 1 (exclusive).

Details

This function calculates the analytic second partial derivatives of the negative log-likelihood function based on the GKw PDF (see `dgkw`). The log-likelihood function $\ell(\theta|\mathbf{x})$ is given by:

$$\ell(\theta) = n\ln(\lambda\alpha\beta) - n\ln B(\gamma, \delta+1) + \sum_{i=1}^{n}[(\alpha-1)\ln(x_i) + (\beta-1)\ln(v_i) + (\gamma\lambda-1)\ln(w_i) + \delta\ln(z_i)]$$

where $\theta = (\alpha, \beta, \gamma, \delta, \lambda)$, $B(a,b)$ is the Beta function (`beta`), and intermediate terms are:

- $v_i = 1 - x_i^{\alpha}$
- $w_i = 1 - v_i^{\beta} = 1 - (1 - x_i^{\alpha})^{\beta}$
- $z_i = 1 - w_i^{\lambda} = 1 - [1 - (1 - x_i^{\alpha})^{\beta}]^{\lambda}$

The Hessian matrix returned contains the elements $-\frac{\partial^2 \ell(\theta|\mathbf{x})}{\partial\theta_i \partial\theta_j}$.

Key properties of the returned matrix:

- Dimensions: 5x5.
- Symmetry: The matrix is symmetric.
- Ordering: Rows and columns correspond to the parameters in the order $\alpha, \beta, \gamma, \delta, \lambda$.
- Content: Analytic second derivatives of the *negative* log-likelihood.

The exact analytical formulas for the second derivatives are implemented directly (often derived using symbolic differentiation) for accuracy and efficiency, typically using C++.

Value

Returns a 5x5 numeric matrix representing the Hessian matrix of the negative log-likelihood function, i.e., the matrix of second partial derivatives $-\partial^2 \ell/(\partial\theta_i \partial\theta_j)$. Returns a 5x5 matrix populated with `NaN` if any parameter values are invalid according to their constraints, or if any value in `data` is not in the interval (0, 1).

Author(s)

Lopes, J. E.

References

Cordeiro, G. M., & de Castro, M. (2011). A new family of generalized distributions. *Journal of Statistical Computation and Simulation*

Kumaraswamy, P. (1980). A generalized probability density function for double-bounded random processes. *Journal of Hydrology*, *46*(1-2), 79-88.

See Also

`llgkw` (negative log-likelihood function), `grgkw` (gradient vector), `dgkw` (density function), `optim`, `hessian` (for numerical Hessian comparison).

Examples

```
## Example 1: Basic Hessian Evaluation

# Generate sample data
set.seed(2323)
n <- 1000
true_params <- c(alpha = 1.5, beta = 2.0, gamma = 0.8, delta = 1.2, lambda = 0.5)
data <- rgkw(n, alpha = true_params[1], beta = true_params[2],
             gamma = true_params[3], delta = true_params[4],
             lambda = true_params[5])

# Evaluate Hessian at true parameters
hess_true <- hsgkw(par = true_params, data = data)
cat("Hessian matrix at true parameters:\n")
print(hess_true, digits = 4)

# Check symmetry
cat("\nSymmetry check (max |H - H^T|):",
    max(abs(hess_true - t(hess_true))), "\n")


## Example 2: Hessian Properties at MLE

# Fit model
fit <- optim(
  par = c(1.2, 2.0, 0.5, 1.5, 0.2),
  fn = llgkw,
  gr = grgkw,
  data = data,
  method = "Nelder-Mead",
  hessian = TRUE,
  control = list(
    maxit = 2000,
    factr = 1e-15,
    pgtol = 1e-15,
    trace = FALSE
    )
)

mle <- fit$par
names(mle) <- c("alpha", "beta", "gamma", "delta", "lambda")

# Hessian at MLE
hessian_at_mle <- hsgkw(par = mle, data = data)
cat("\nHessian at MLE:\n")
print(hessian_at_mle, digits = 4)

# Compare with optim's numerical Hessian
cat("\nComparison with optim Hessian:\n")
cat("Max absolute difference:",
    max(abs(hessian_at_mle - fit$hessian)), "\n")

# Eigenvalue analysis
eigenvals <- eigen(hessian_at_mle, only.values = TRUE)$values
cat("\nEigenvalues:\n")
print(eigenvals)

cat("\nPositive definite:", all(eigenvals > 0), "\n")
cat("Condition number:", max(eigenvals) / min(eigenvals), "\n")


## Example 3: Standard Errors and Confidence Intervals

# Observed information matrix
obs_info <- hessian_at_mle

# Variance-covariance matrix
vcov_matrix <- solve(obs_info)
cat("\nVariance-Covariance Matrix:\n")
print(vcov_matrix, digits = 6)
```

```
# Standard errors
se <- sqrt(diag(vcov_matrix))
names(se) <- c("alpha", "beta", "gamma", "delta", "lambda")

# Correlation matrix
corr_matrix <- cov2cor(vcov_matrix)
cat("\nCorrelation Matrix:\n")
print(corr_matrix, digits = 4)

# Confidence intervals
z_crit <- qnorm(0.975)
results <- data.frame(
  Parameter = c("alpha", "beta", "gamma", "delta", "lambda"),
  True = true_params,
  MLE = mle,
  SE = se,
  CI_Lower = mle - z_crit * se,
  CI_Upper = mle + z_crit * se
)
print(results, digits = 4)


## Example 4: Determinant and Trace Analysis

# Compute at different points
test_params <- rbind(
  c(1.5, 2.5, 1.2, 1.5, 1.5),
  c(2.0, 3.0, 1.5, 2.0, 1.8),
  mle,
  c(2.5, 3.5, 1.8, 2.5, 2.0)
)

hess_properties <- data.frame(
  Alpha = numeric(),
  Beta = numeric(),
  Gamma = numeric(),
  Delta = numeric(),
  Lambda = numeric(),
  Determinant = numeric(),
  Trace = numeric(),
  Min_Eigenval = numeric(),
  Max_Eigenval = numeric(),
  Cond_Number = numeric(),
  stringsAsFactors = FALSE
)

for (i in 1:nrow(test_params)) {
  H <- hsgkw(par = test_params[i, ], data = data)
  eigs <- eigen(H, only.values = TRUE)$values

  hess_properties <- rbind(hess_properties, data.frame(
    Alpha = test_params[i, 1],
    Beta = test_params[i, 2],
    Gamma = test_params[i, 3],
    Delta = test_params[i, 4],
    Lambda = test_params[i, 5],
    Determinant = det(H),
    Trace = sum(diag(H)),
    Min_Eigenval = min(eigs),
    Max_Eigenval = max(eigs),
    Cond_Number = max(eigs) / min(eigs)
  ))
}

cat("\nHessian Properties at Different Points:\n")
print(hess_properties, digits = 4, row.names = FALSE)


## Example 5: Curvature Visualization (Alpha vs Beta)

xd <- 2
# Create grid around MLE
alpha_grid <- seq(mle[1] - xd, mle[1] + xd, length.out = round(n/4))
beta_grid <- seq(mle[2] - xd, mle[2] + xd, length.out = round(n/4))
alpha_grid <- alpha_grid[alpha_grid > 0]
beta_grid <- beta_grid[beta_grid > 0]

# Compute curvature measures
determinant_surface <- matrix(NA, nrow = length(alpha_grid),
                               ncol = length(beta_grid))
trace_surface <- matrix(NA, nrow = length(alpha_grid),
                         ncol = length(beta_grid))

for (i in seq_along(alpha_grid)) {
  for (j in seq_along(beta_grid)) {
    H <- hsgkw(c(alpha_grid[i], beta_grid[j], mle[3], mle[4], mle[5]), data)
    determinant_surface[i, j] <- det(H)
    trace_surface[i, j] <- sum(diag(H))
  }
}

# Plot

contour(alpha_grid, beta_grid, determinant_surface,
        xlab = expression(alpha), ylab = expression(beta),
        main = "Hessian Determinant", las = 1,
        col = "#2E4057", lwd = 1.5, nlevels = 15)
points(mle[1], mle[2], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[2], pch = 17, col = "#006400", cex = 1.5)
grid(col = "gray90")

contour(alpha_grid, beta_grid, trace_surface,
        xlab = expression(alpha), ylab = expression(beta),
        main = "Hessian Trace", las = 1,
        col = "#2E4057", lwd = 1.5, nlevels = 15)
points(mle[1], mle[2], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[2], pch = 17, col = "#006400", cex = 1.5)
grid(col = "gray90")


## Example 6: Confidence Ellipse (Alpha vs Beta)

# Extract 2x2 submatrix for alpha and beta
vcov_2d <- vcov_matrix[1:2, 1:2]

# Create confidence ellipse
theta <- seq(0, 2 * pi, length.out = round(n/4))
chi2_val <- qchisq(0.95, df = 2)

eig_decomp <- eigen(vcov_2d)
ellipse <- matrix(NA, nrow = round(n/4), ncol = 2)
for (i in 1:round(n/4)) {
  v <- c(cos(theta[i]), sin(theta[i]))
  ellipse[i, ] <- mle[1:2] + sqrt(chi2_val) *
    (eig_decomp$vectors %*% diag(sqrt(eig_decomp$values)) %*% v)
}

# Marginal confidence intervals
se_2d <- sqrt(diag(vcov_2d))
ci_alpha <- mle[1] + c(-1, 1) * 1.96 * se_2d[1]
ci_beta <- mle[2] + c(-1, 1) * 1.96 * se_2d[2]

# Plot
plot(ellipse[, 1], ellipse[, 2], type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(alpha), ylab = expression(beta),
     main = "95% Confidence Ellipse (Alpha vs Beta)", las = 1)

# Add marginal CIs
abline(v = ci_alpha, col = "#808080", lty = 3, lwd = 1.5)
abline(h = ci_beta, col = "#808080", lty = 3, lwd = 1.5)

points(mle[1], mle[2], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[2], pch = 17, col = "#006400", cex = 1.5)

legend("topright",
       legend = c("MLE", "True", "95% CR", "Marginal 95% CI"),
       col = c("#8B0000", "#006400", "#2E4057", "#808080"),
       pch = c(19, 17, NA, NA),
       lty = c(NA, NA, 1, 3),
       lwd = c(NA, NA, 2, 1.5),
       bty = "n")
grid(col = "gray90")


## Example 7: Confidence Ellipse (Gamma vs Delta)

# Extract 2x2 submatrix for gamma and delta
vcov_2d_gd <- vcov_matrix[3:4, 3:4]
```

```
# Create confidence ellipse
eig_decomp_gd <- eigen(vcov_2d_gd)
ellipse_gd <- matrix(NA, nrow = round(n/4), ncol = 2)
for (i in 1:round(n/4)) {
  v <- c(cos(theta[i]), sin(theta[i]))
  ellipse_gd[i, ] <- mle[3:4] + sqrt(chi2_val) *
    (eig_decomp_gd$vectors %*% diag(sqrt(eig_decomp_gd$values)) %*% v)
}

# Marginal confidence intervals
se_2d_gd <- sqrt(diag(vcov_2d_gd))
ci_gamma <- mle[3] + c(-1, 1) * 1.96 * se_2d_gd[1]
ci_delta <- mle[4] + c(-1, 1) * 1.96 * se_2d_gd[2]

# Plot
plot(ellipse_gd[, 1], ellipse_gd[, 2], type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(gamma), ylab = expression(delta),
     main = "95% Confidence Ellipse (Gamma vs Delta)", las = 1)

# Add marginal CIs
abline(v = ci_gamma, col = "#808080", lty = 3, lwd = 1.5)
abline(h = ci_delta, col = "#808080", lty = 3, lwd = 1.5)

points(mle[3], mle[4], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[3], true_params[4], pch = 17, col = "#006400", cex = 1.5)

legend("topright",
       legend = c("MLE", "True", "95% CR", "Marginal 95% CI"),
       col = c("#8B0000", "#006400", "#2E4057", "#808080"),
       pch = c(19, 17, NA, NA),
       lty = c(NA, NA, 1, 3),
       lwd = c(NA, NA, 2, 1.5),
       bty = "n")
grid(col = "gray90")


## Example 8: Confidence Ellipse (Delta vs Lambda)

# Extract 2x2 submatrix for delta and lambda
vcov_2d_dl <- vcov_matrix[4:5, 4:5]

# Create confidence ellipse
eig_decomp_dl <- eigen(vcov_2d_dl)
ellipse_dl <- matrix(NA, nrow = round(n/4), ncol = 2)
for (i in 1:round(n/4)) {
  v <- c(cos(theta[i]), sin(theta[i]))
  ellipse_dl[i, ] <- mle[4:5] + sqrt(chi2_val) *
    (eig_decomp_dl$vectors %*% diag(sqrt(eig_decomp_dl$values)) %*% v)
}

# Marginal confidence intervals
se_2d_dl <- sqrt(diag(vcov_2d_dl))
ci_delta_2 <- mle[4] + c(-1, 1) * 1.96 * se_2d_dl[1]
```

```
ci_lambda <- mle[5] + c(-1, 1) * 1.96 * se_2d_dl[2]

# Plot
plot(ellipse_dl[, 1], ellipse_dl[, 2], type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(delta), ylab = expression(lambda),
     main = "95% Confidence Ellipse (Delta vs Lambda)", las = 1)

# Add marginal CIs
abline(v = ci_delta_2, col = "#808080", lty = 3, lwd = 1.5)
abline(h = ci_lambda, col = "#808080", lty = 3, lwd = 1.5)

points(mle[4], mle[5], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[4], true_params[5], pch = 17, col = "#006400", cex = 1.5)

legend("topright",
       legend = c("MLE", "True", "95% CR", "Marginal 95% CI"),
       col = c("#8B0000", "#006400", "#2E4057", "#808080"),
       pch = c(19, 17, NA, NA),
       lty = c(NA, NA, 1, 3),
       lwd = c(NA, NA, 2, 1.5),
       bty = "n")
grid(col = "gray90")
```

hskkw — *Hessian Matrix of the Negative Log-Likelihood for the kkw Distribution*

Description

Computes the analytic 4x4 Hessian matrix (matrix of second partial derivatives) of the negative log-likelihood function for the Kumaraswamy-Kumaraswamy (kkw) distribution with parameters `alpha` (α), `beta` (β), `delta` (δ), and `lambda` (λ). This distribution is the special case of the Generalized Kumaraswamy (GKw) distribution where $\gamma = 1$. The Hessian is useful for estimating standard errors and in optimization algorithms.

Usage

```
hskkw(par, data)
```

Arguments

`par`	A numeric vector of length 4 containing the distribution parameters in the order: `alpha` ($\alpha > 0$), `beta` ($\beta > 0$), `delta` ($\delta \geq 0$), `lambda` ($\lambda > 0$).
`data`	A numeric vector of observations. All values must be strictly between 0 and 1 (exclusive).

Details

This function calculates the analytic second partial derivatives of the negative log-likelihood function based on the kkw log-likelihood ($\gamma = 1$ case of GKw, see `llkkw`):

$$\ell(\theta|\mathbf{x}) = n[\ln(\delta+1)+\ln(\lambda)+\ln(\alpha)+\ln(\beta)]+\sum_{i=1}^{n}[(\alpha-1)\ln(x_i)+(\beta-1)\ln(v_i)+(\lambda-1)\ln(w_i)+\delta\ln(z_i)]$$

where $\theta = (\alpha, \beta, \delta, \lambda)$ and intermediate terms are:

- $v_i = 1 - x_i^\alpha$
- $w_i = 1 - v_i^\beta = 1 - (1 - x_i^\alpha)^\beta$
- $z_i = 1 - w_i^\lambda = 1 - [1 - (1 - x_i^\alpha)^\beta]^\lambda$

The Hessian matrix returned contains the elements $-\frac{\partial^2 \ell(\theta|\mathbf{x})}{\partial\theta_i\partial\theta_j}$ for $\theta_i, \theta_j \in \{\alpha, \beta, \delta, \lambda\}$.

Key properties of the returned matrix:

- Dimensions: 4x4.
- Symmetry: The matrix is symmetric.
- Ordering: Rows and columns correspond to the parameters in the order $\alpha, \beta, \delta, \lambda$.
- Content: Analytic second derivatives of the *negative* log-likelihood.

This corresponds to the relevant submatrix of the 5x5 GKw Hessian (`hsgkw`) evaluated at $\gamma = 1$. The exact analytical formulas are implemented directly.

Value

Returns a 4x4 numeric matrix representing the Hessian matrix of the negative log-likelihood function, $-\partial^2\ell/(\partial\theta_i\partial\theta_j)$, where $\theta = (\alpha, \beta, \delta, \lambda)$. Returns a 4x4 matrix populated with `NaN` if any parameter values are invalid according to their constraints, or if any value in `data` is not in the interval (0, 1).

Author(s)

Lopes, J. E.

References

Cordeiro, G. M., & de Castro, M. (2011). A new family of generalized distributions. *Journal of Statistical Computation and Simulation*

Kumaraswamy, P. (1980). A generalized probability density function for double-bounded random processes. *Journal of Hydrology*, *46*(1-2), 79-88.

See Also

`hsgkw` (parent distribution Hessian), `llkkw` (negative log-likelihood for kkw), `grkkw` (gradient for kkw), `dkkw` (density for kkw), `optim`, `hessian` (for numerical Hessian comparison).

Examples

```
## Example 1: Basic Hessian Evaluation

# Generate sample data
set.seed(123)
n <- 1000
true_params <- c(alpha = 2.0, beta = 3.0, delta = 1.5, lambda = 2.0)
data <- rkkw(n, alpha = true_params[1], beta = true_params[2],
             delta = true_params[3], lambda = true_params[4])

# Evaluate Hessian at true parameters
hess_true <- hskkw(par = true_params, data = data)
cat("Hessian matrix at true parameters:\n")
print(hess_true, digits = 4)

# Check symmetry
cat("\nSymmetry check (max |H - H^T|):",
    max(abs(hess_true - t(hess_true))), "\n")


## Example 2: Hessian Properties at MLE

# Fit model
fit <- optim(
  par = c(1.5, 2.5, 1.0, 1.5),
  fn = llkkw,
  gr = grkkw,
  data = data,
  method = "BFGS",
  hessian = TRUE
)

mle <- fit$par
names(mle) <- c("alpha", "beta", "delta", "lambda")

# Hessian at MLE
hessian_at_mle <- hskkw(par = mle, data = data)
cat("\nHessian at MLE:\n")
print(hessian_at_mle, digits = 4)

# Compare with optim's numerical Hessian
cat("\nComparison with optim Hessian:\n")
cat("Max absolute difference:",
    max(abs(hessian_at_mle - fit$hessian)), "\n")

# Eigenvalue analysis
eigenvals <- eigen(hessian_at_mle, only.values = TRUE)$values
cat("\nEigenvalues:\n")
print(eigenvals)

cat("\nPositive definite:", all(eigenvals > 0), "\n")
cat("Condition number:", max(eigenvals) / min(eigenvals), "\n")
```

```
## Example 3: Standard Errors and Confidence Intervals

# Observed information matrix
obs_info <- hessian_at_mle

# Variance-covariance matrix
vcov_matrix <- solve(obs_info)
cat("\nVariance-Covariance Matrix:\n")
print(vcov_matrix, digits = 6)

# Standard errors
se <- sqrt(diag(vcov_matrix))
names(se) <- c("alpha", "beta", "delta", "lambda")

# Correlation matrix
corr_matrix <- cov2cor(vcov_matrix)
cat("\nCorrelation Matrix:\n")
print(corr_matrix, digits = 4)

# Confidence intervals
z_crit <- qnorm(0.975)
results <- data.frame(
  Parameter = c("alpha", "beta", "delta", "lambda"),
  True = true_params,
  MLE = mle,
  SE = se,
  CI_Lower = mle - z_crit * se,
  CI_Upper = mle + z_crit * se
)
print(results, digits = 4)


## Example 4: Determinant and Trace Analysis

# Compute at different points
test_params <- rbind(
  c(1.5, 2.5, 1.0, 1.5),
  c(2.0, 3.0, 1.5, 2.0),
  mle,
  c(2.5, 3.5, 2.0, 2.5)
)

hess_properties <- data.frame(
  Alpha = numeric(),
  Beta = numeric(),
  Delta = numeric(),
  Lambda = numeric(),
  Determinant = numeric(),
  Trace = numeric(),
  Min_Eigenval = numeric(),
  Max_Eigenval = numeric(),
  Cond_Number = numeric(),
  stringsAsFactors = FALSE
)

for (i in 1:nrow(test_params)) {
  H <- hskkw(par = test_params[i, ], data = data)
  eigs <- eigen(H, only.values = TRUE)$values

  hess_properties <- rbind(hess_properties, data.frame(
    Alpha = test_params[i, 1],
    Beta = test_params[i, 2],
    Delta = test_params[i, 3],
    Lambda = test_params[i, 4],
    Determinant = det(H),
    Trace = sum(diag(H)),
    Min_Eigenval = min(eigs),
    Max_Eigenval = max(eigs),
    Cond_Number = max(eigs) / min(eigs)
  ))
}

cat("\nHessian Properties at Different Points:\n")
print(hess_properties, digits = 4, row.names = FALSE)


## Example 5: Curvature Visualization (Alpha vs Beta)

# Create grid around MLE
alpha_grid <- seq(mle[1] - 1, mle[1] + 1, length.out = round(n/4))
beta_grid <- seq(mle[2] - 1, mle[2] + 1, length.out = round(n/4))
alpha_grid <- alpha_grid[alpha_grid > 0]
beta_grid <- beta_grid[beta_grid > 0]

# Compute curvature measures
determinant_surface <- matrix(NA, nrow = length(alpha_grid),
                               ncol = length(beta_grid))
trace_surface <- matrix(NA, nrow = length(alpha_grid),
                         ncol = length(beta_grid))

for (i in seq_along(alpha_grid)) {
  for (j in seq_along(beta_grid)) {
    H <- hskkw(c(alpha_grid[i], beta_grid[j], mle[3], mle[4]), data)
    determinant_surface[i, j] <- det(H)
    trace_surface[i, j] <- sum(diag(H))
  }
}

# Plot

contour(alpha_grid, beta_grid, determinant_surface,
        xlab = expression(alpha), ylab = expression(beta),
        main = "Hessian Determinant", las = 1,
        col = "#2E4057", lwd = 1.5, nlevels = 15)
```

```
points(mle[1], mle[2], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[2], pch = 17, col = "#006400", cex = 1.5)
grid(col = "gray90")

contour(alpha_grid, beta_grid, trace_surface,
        xlab = expression(alpha), ylab = expression(beta),
        main = "Hessian Trace", las = 1,
        col = "#2E4057", lwd = 1.5, nlevels = 15)
points(mle[1], mle[2], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[2], pch = 17, col = "#006400", cex = 1.5)
grid(col = "gray90")

## Example 6: Confidence Ellipse (Alpha vs Beta)

# Extract 2x2 submatrix for alpha and beta
vcov_2d <- vcov_matrix[1:2, 1:2]

# Create confidence ellipse
theta <- seq(0, 2 * pi, length.out = round(n/2))
chi2_val <- qchisq(0.95, df = 2)

eig_decomp <- eigen(vcov_2d)
ellipse <- matrix(NA, nrow = round(n/2), ncol = 2)
for (i in 1:round(n/2)) {
  v <- c(cos(theta[i]), sin(theta[i]))
  ellipse[i, ] <- mle[1:2] + sqrt(chi2_val) *
    (eig_decomp$vectors %*% diag(sqrt(eig_decomp$values)) %*% v)
}

# Marginal confidence intervals
se_2d <- sqrt(diag(vcov_2d))
ci_alpha <- mle[1] + c(-1, 1) * 1.96 * se_2d[1]
ci_beta <- mle[2] + c(-1, 1) * 1.96 * se_2d[2]

# Plot
plot(ellipse[, 1], ellipse[, 2], type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(alpha), ylab = expression(beta),
     main = "95% Confidence Ellipse (Alpha vs Beta)", las = 1)

# Add marginal CIs
abline(v = ci_alpha, col = "#808080", lty = 3, lwd = 1.5)
abline(h = ci_beta, col = "#808080", lty = 3, lwd = 1.5)

points(mle[1], mle[2], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[2], pch = 17, col = "#006400", cex = 1.5)

legend("topright",
       legend = c("MLE", "True", "95% CR", "Marginal 95% CI"),
       col = c("#8B0000", "#006400", "#2E4057", "#808080"),
       pch = c(19, 17, NA, NA),
       lty = c(NA, NA, 1, 3),
       lwd = c(NA, NA, 2, 1.5),
       bty = "n")
```

```
grid(col = "gray90")


## Example 7: Confidence Ellipse (Delta vs Lambda)

# Extract 2x2 submatrix for delta and lambda
vcov_2d_dl <- vcov_matrix[3:4, 3:4]

# Create confidence ellipse
eig_decomp_dl <- eigen(vcov_2d_dl)
ellipse_dl <- matrix(NA, nrow = round(n/2), ncol = 2)
for (i in 1:round(n/2)) {
  v <- c(cos(theta[i]), sin(theta[i]))
  ellipse_dl[i, ] <- mle[3:4] + sqrt(chi2_val) *
    (eig_decomp_dl$vectors %*% diag(sqrt(eig_decomp_dl$values)) %*% v)
}

# Marginal confidence intervals
se_2d_dl <- sqrt(diag(vcov_2d_dl))
ci_delta <- mle[3] + c(-1, 1) * 1.96 * se_2d_dl[1]
ci_lambda <- mle[4] + c(-1, 1) * 1.96 * se_2d_dl[2]

# Plot
plot(ellipse_dl[, 1], ellipse_dl[, 2], type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(delta), ylab = expression(lambda),
     main = "95% Confidence Ellipse (Delta vs Lambda)", las = 1)

# Add marginal CIs
abline(v = ci_delta, col = "#808080", lty = 3, lwd = 1.5)
abline(h = ci_lambda, col = "#808080", lty = 3, lwd = 1.5)

points(mle[3], mle[4], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[3], true_params[4], pch = 17, col = "#006400", cex = 1.5)

legend("topright",
       legend = c("MLE", "True", "95% CR", "Marginal 95% CI"),
       col = c("#8B0000", "#006400", "#2E4057", "#808080"),
       pch = c(19, 17, NA, NA),
       lty = c(NA, NA, 1, 3),
       lwd = c(NA, NA, 2, 1.5),
       bty = "n")
grid(col = "gray90")
```

hskw *Hessian Matrix of the Negative Log-Likelihood for the Kw Distribution*

Description

Computes the analytic 2x2 Hessian matrix (matrix of second partial derivatives) of the negative log-likelihood function for the two-parameter Kumaraswamy (Kw) distribution with parameters `alpha` (α) and `beta` (β). The Hessian is useful for estimating standard errors and in optimization algorithms.

Usage

```
hskw(par, data)
```

Arguments

`par`	A numeric vector of length 2 containing the distribution parameters in the order: `alpha` ($\alpha > 0$), `beta` ($\beta > 0$).
`data`	A numeric vector of observations. All values must be strictly between 0 and 1 (exclusive).

Details

This function calculates the analytic second partial derivatives of the negative log-likelihood function ($-\ell(\theta|\mathbf{x})$). The components are the negative of the second derivatives of the log-likelihood ℓ (derived from the PDF in `dkw`).

Let $v_i = 1 - x_i^\alpha$. The second derivatives of the positive log-likelihood (ℓ) are:

$$\frac{\partial^2 \ell}{\partial \alpha^2} = -\frac{n}{\alpha^2} - (\beta - 1) \sum_{i=1}^{n} \frac{x_i^\alpha (\ln(x_i))^2}{v_i^2}$$

$$\frac{\partial^2 \ell}{\partial \alpha \partial \beta} = -\sum_{i=1}^{n} \frac{x_i^\alpha \ln(x_i)}{v_i}$$

$$\frac{\partial^2 \ell}{\partial \beta^2} = -\frac{n}{\beta^2}$$

The function returns the Hessian matrix containing the negative of these values.

Key properties of the returned matrix:

- Dimensions: 2x2.
- Symmetry: The matrix is symmetric.
- Ordering: Rows and columns correspond to the parameters in the order α, β.
- Content: Analytic second derivatives of the *negative* log-likelihood.

This corresponds to the relevant 2x2 submatrix of the 5x5 GKw Hessian (`hsgkw`) evaluated at $\gamma = 1, \delta = 0, \lambda = 1$.

Value

Returns a 2x2 numeric matrix representing the Hessian matrix of the negative log-likelihood function, $-\partial^2 \ell / (\partial \theta_i \partial \theta_j)$, where $\theta = (\alpha, \beta)$. Returns a 2x2 matrix populated with `NaN` if any parameter values are invalid according to their constraints, or if any value in `data` is not in the interval (0, 1).

Author(s)

Lopes, J. E.

References

Kumaraswamy, P. (1980). A generalized probability density function for double-bounded random processes. *Journal of Hydrology*, *46*(1-2), 79-88.

Jones, M. C. (2009). Kumaraswamy's distribution: A beta-type distribution with some tractability advantages. *Statistical Methodology*, *6*(1), 70-81.

(Note: Specific Hessian formulas might be derived or sourced from additional references).

See Also

`hsgkw` (parent distribution Hessian), `llkw` (negative log-likelihood for Kw), `grkw` (gradient for Kw, if available), `dkw` (density for Kw), `optim`, `hessian` (for numerical Hessian comparison).

Examples

```
## Example 1: Basic Hessian Evaluation

# Generate sample data
set.seed(123)
n <- 1000
true_params <- c(alpha = 2.5, beta = 3.5)
data <- rkw(n, alpha = true_params[1], beta = true_params[2])

# Evaluate Hessian at true parameters
hess_true <- hskw(par = true_params, data = data)
cat("Hessian matrix at true parameters:\n")
print(hess_true, digits = 4)

# Check symmetry
cat("\nSymmetry check (max |H - H^T|):",
    max(abs(hess_true - t(hess_true))), "\n")

## Example 2: Hessian Properties at MLE

# Fit model
fit <- optim(
  par = c(2, 2),
  fn = llkw,
  gr = grkw,
  data = data,
  method = "BFGS",
  hessian = TRUE
)

mle <- fit$par
names(mle) <- c("alpha", "beta")

# Hessian at MLE
```

```
hessian_at_mle <- hskw(par = mle, data = data)
cat("\nHessian at MLE:\n")
print(hessian_at_mle, digits = 4)

# Compare with optim's numerical Hessian
cat("\nComparison with optim Hessian:\n")
cat("Max absolute difference:",
    max(abs(hessian_at_mle - fit$hessian)), "\n")

# Eigenvalue analysis
eigenvals <- eigen(hessian_at_mle, only.values = TRUE)$values
cat("\nEigenvalues:\n")
print(eigenvals)

cat("\nPositive definite:", all(eigenvals > 0), "\n")
cat("Condition number:", max(eigenvals) / min(eigenvals), "\n")


## Example 3: Standard Errors and Confidence Intervals

# Observed information matrix (negative Hessian for neg-loglik)
obs_info <- hessian_at_mle

# Variance-covariance matrix
vcov_matrix <- solve(obs_info)
cat("\nVariance-Covariance Matrix:\n")
print(vcov_matrix, digits = 6)

# Standard errors
se <- sqrt(diag(vcov_matrix))
names(se) <- c("alpha", "beta")

# Correlation matrix
corr_matrix <- cov2cor(vcov_matrix)
cat("\nCorrelation Matrix:\n")
print(corr_matrix, digits = 4)

# Confidence intervals
z_crit <- qnorm(0.975)
results <- data.frame(
  Parameter = c("alpha", "beta"),
  True = true_params,
  MLE = mle,
  SE = se,
  CI_Lower = mle - z_crit * se,
  CI_Upper = mle + z_crit * se
)
print(results, digits = 4)

## Example 4: Determinant and Trace Analysis

# Compute at different points
test_params <- rbind(
  c(1.5, 2.5),
  c(2.0, 3.0),
  mle,
  c(3.0, 4.0)
)

hess_properties <- data.frame(
  Alpha = numeric(),
  Beta = numeric(),
  Determinant = numeric(),
  Trace = numeric(),
  Min_Eigenval = numeric(),
  Max_Eigenval = numeric(),
  Cond_Number = numeric(),
  stringsAsFactors = FALSE
)

for (i in 1:nrow(test_params)) {
  H <- hskw(par = test_params[i, ], data = data)
  eigs <- eigen(H, only.values = TRUE)$values

  hess_properties <- rbind(hess_properties, data.frame(
    Alpha = test_params[i, 1],
    Beta = test_params[i, 2],
    Determinant = det(H),
    Trace = sum(diag(H)),
    Min_Eigenval = min(eigs),
    Max_Eigenval = max(eigs),
    Cond_Number = max(eigs) / min(eigs)
  ))
}

cat("\nHessian Properties at Different Points:\n")
print(hess_properties, digits = 4, row.names = FALSE)

## Example 5: Curvature Visualization

# Create grid around MLE
alpha_grid <- seq(mle[1] - 0.5, mle[1] + 0.5, length.out = 30)
beta_grid <- seq(mle[2] - 0.5, mle[2] + 0.5, length.out = 30)
alpha_grid <- alpha_grid[alpha_grid > 0]
beta_grid <- beta_grid[beta_grid > 0]

# Compute curvature measures
determinant_surface <- matrix(NA, nrow = length(alpha_grid),
                              ncol = length(beta_grid))
trace_surface <- matrix(NA, nrow = length(alpha_grid),
                        ncol = length(beta_grid))

for (i in seq_along(alpha_grid)) {
  for (j in seq_along(beta_grid)) {
    H <- hskw(c(alpha_grid[i], beta_grid[j]), data)
    determinant_surface[i, j] <- det(H)
    trace_surface[i, j] <- sum(diag(H))
  }
}

# Plot

contour(alpha_grid, beta_grid, determinant_surface,
        xlab = expression(alpha), ylab = expression(beta),
        main = "Hessian Determinant", las = 1,
        col = "#2E4057", lwd = 1.5, nlevels = 15)
points(mle[1], mle[2], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[2], pch = 17, col = "#006400", cex = 1.5)
grid(col = "gray90")

contour(alpha_grid, beta_grid, trace_surface,
        xlab = expression(alpha), ylab = expression(beta),
        main = "Hessian Trace", las = 1,
        col = "#2E4057", lwd = 1.5, nlevels = 15)
points(mle[1], mle[2], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[2], pch = 17, col = "#006400", cex = 1.5)
grid(col = "gray90")

## Example 6: Fisher Information and Asymptotic Efficiency

# Observed information (at MLE)
obs_fisher <- hessian_at_mle

# Asymptotic covariance matrix
asymp_cov <- solve(obs_fisher)

cat("\nAsymptotic Standard Errors:\n")
cat("SE(alpha):", sqrt(asymp_cov[1, 1]), "\n")
cat("SE(beta):", sqrt(asymp_cov[2, 2]), "\n")

# Cramér-Rao Lower Bound
cat("\nCramér-Rao Lower Bounds:\n")
cat("CRLB(alpha):", sqrt(asymp_cov[1, 1]), "\n")
cat("CRLB(beta):", sqrt(asymp_cov[2, 2]), "\n")

# Efficiency ellipse (95% confidence region)
theta <- seq(0, 2 * pi, length.out = 100)
chi2_val <- qchisq(0.95, df = 2)

# Eigendecomposition
eig_decomp <- eigen(asymp_cov)

# Ellipse points
ellipse <- matrix(NA, nrow = 100, ncol = 2)
for (i in 1:100) {
  v <- c(cos(theta[i]), sin(theta[i]))
  ellipse[i, ] <- mle + sqrt(chi2_val) *
    (eig_decomp$vectors %*% diag(sqrt(eig_decomp$values)) %*% v)
}
```

```
# Plot confidence ellipse

plot(ellipse[, 1], ellipse[, 2], type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(alpha), ylab = expression(beta),
     main = "95% Confidence Ellipse", las = 1)
points(mle[1], mle[2], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[2], pch = 17, col = "#006400", cex = 1.5)
legend("topright",
       legend = c("MLE", "True", "95% CR"),
       col = c("#8B0000", "#006400", "#2E4057"),
       pch = c(19, 17, NA), lty = c(NA, NA, 1),
       lwd = c(NA, NA, 2), bty = "n")
grid(col = "gray90")
```

hsmc — *Hessian Matrix of the Negative Log-Likelihood for the McDonald (Mc)/Beta Power Distribution*

Description

Computes the analytic 3x3 Hessian matrix (matrix of second partial derivatives) of the negative log-likelihood function for the McDonald (Mc) distribution (also known as Beta Power) with parameters `gamma` (γ), `delta` (δ), and `lambda` (λ). This distribution is the special case of the Generalized Kumaraswamy (GKw) distribution where $\alpha = 1$ and $\beta = 1$. The Hessian is useful for estimating standard errors and in optimization algorithms.

Usage

```
hsmc(par, data)
```

Arguments

`par`	A numeric vector of length 3 containing the distribution parameters in the order: `gamma` ($\gamma > 0$), `delta` ($\delta \geq 0$), `lambda` ($\lambda > 0$).
`data`	A numeric vector of observations. All values must be strictly between 0 and 1 (exclusive).

Details

This function calculates the analytic second partial derivatives of the negative log-likelihood function ($-\ell(\theta|\mathbf{x})$). The components are based on the second derivatives of the log-likelihood ℓ (derived from the PDF in `dmc`).

Note: The formulas below represent the second derivatives of the positive log-likelihood (ℓ). The function returns the **negative** of these values. Users should verify these formulas independently if using for critical applications.

$$\frac{\partial^2 \ell}{\partial \gamma^2} = -n[\psi'(\gamma) - \psi'(\gamma+\delta+1)]$$

$$\frac{\partial^2 \ell}{\partial \gamma \partial \delta} = -n\psi'(\gamma+\delta+1)$$

$$\frac{\partial^2 \ell}{\partial \gamma \partial \lambda} = \sum_{i=1}^{n} \ln(x_i)$$

$$\frac{\partial^2 \ell}{\partial \delta^2} = -n[\psi'(\delta+1) - \psi'(\gamma+\delta+1)]$$

$$\frac{\partial^2 \ell}{\partial \delta \partial \lambda} = -\sum_{i=1}^{n} \frac{x_i^\lambda \ln(x_i)}{1 - x_i^\lambda}$$

$$\frac{\partial^2 \ell}{\partial \lambda^2} = -\frac{n}{\lambda^2} - \delta \sum_{i=1}^{n} \frac{x_i^\lambda [\ln(x_i)]^2}{(1 - x_i^\lambda)^2}$$

where $\psi'(\cdot)$ is the trigamma function (`trigamma`). (*Note: The formula for $\partial^2\ell/\partial\lambda^2$ provided in the source comment was different and potentially related to the expected information matrix; the formula shown here is derived from the gradient provided earlier. Verification is recommended.*)

The returned matrix is symmetric, with rows/columns corresponding to γ, δ, λ.

Value

Returns a 3x3 numeric matrix representing the Hessian matrix of the negative log-likelihood function, $-\partial^2\ell/(\partial\theta_i\partial\theta_j)$, where $\theta = (\gamma, \delta, \lambda)$. Returns a 3x3 matrix populated with `NaN` if any parameter values are invalid according to their constraints, or if any value in `data` is not in the interval (0, 1).

Author(s)

Lopes, J. E.

References

McDonald, J. B. (1984). Some generalized functions for the size distribution of income. *Econometrica*, 52(3), 647-663.

Cordeiro, G. M., & de Castro, M. (2011). A new family of generalized distributions. *Journal of Statistical Computation and Simulation*,

(Note: Specific Hessian formulas might be derived or sourced from additional references).

See Also

`hsgkw` (parent distribution Hessian), `llmc` (negative log-likelihood for Mc), `grmc` (gradient for Mc, if available), `dmc` (density for Mc), `optim`, `hessian` (for numerical Hessian comparison), `trigamma`.

Examples

```
## Example 1: Basic Hessian Evaluation

# Generate sample data with more stable parameters
set.seed(123)
n <- 1000
true_params <- c(gamma = 2.0, delta = 2.5, lambda = 1.5)
data <- rmc(n, gamma = true_params[1], delta = true_params[2],
            lambda = true_params[3])

# Evaluate Hessian at true parameters
hess_true <- hsmc(par = true_params, data = data)
cat("Hessian matrix at true parameters:\n")
print(hess_true, digits = 4)

# Check symmetry
cat("\nSymmetry check (max |H - H^T|):",
    max(abs(hess_true - t(hess_true))), "\n")


## Example 2: Hessian Properties at MLE

# Fit model
fit <- optim(
  par = c(1.5, 2.0, 1.0),
  fn = llmc,
  gr = grmc,
  data = data,
  method = "BFGS",
  hessian = TRUE
)

mle <- fit$par
names(mle) <- c("gamma", "delta", "lambda")

# Hessian at MLE
hessian_at_mle <- hsmc(par = mle, data = data)
cat("\nHessian at MLE:\n")
print(hessian_at_mle, digits = 4)

# Compare with optim's numerical Hessian
cat("\nComparison with optim Hessian:\n")
cat("Max absolute difference:",
    max(abs(hessian_at_mle - fit$hessian)), "\n")

# Eigenvalue analysis
eigenvals <- eigen(hessian_at_mle, only.values = TRUE)$values
cat("\nEigenvalues:\n")
print(eigenvals)

cat("\nPositive definite:", all(eigenvals > 0), "\n")
cat("Condition number:", max(eigenvals) / min(eigenvals), "\n")
```

```
## Example 3: Standard Errors and Confidence Intervals

# Observed information matrix
obs_info <- hessian_at_mle

# Variance-covariance matrix
vcov_matrix <- solve(obs_info)
cat("\nVariance-Covariance Matrix:\n")
print(vcov_matrix, digits = 6)

# Standard errors
se <- sqrt(diag(vcov_matrix))
names(se) <- c("gamma", "delta", "lambda")

# Correlation matrix
corr_matrix <- cov2cor(vcov_matrix)
cat("\nCorrelation Matrix:\n")
print(corr_matrix, digits = 4)

# Confidence intervals
z_crit <- qnorm(0.975)
results <- data.frame(
  Parameter = c("gamma", "delta", "lambda"),
  True = true_params,
  MLE = mle,
  SE = se,
  CI_Lower = mle - z_crit * se,
  CI_Upper = mle + z_crit * se
)
print(results, digits = 4)


## Example 4: Determinant and Trace Analysis

# Compute at different points
test_params <- rbind(
  c(1.5, 2.0, 1.0),
  c(2.0, 2.5, 1.5),
  mle,
  c(2.5, 3.0, 2.0)
)

hess_properties <- data.frame(
  Gamma = numeric(),
  Delta = numeric(),
  Lambda = numeric(),
  Determinant = numeric(),
  Trace = numeric(),
  Min_Eigenval = numeric(),
  Max_Eigenval = numeric(),
  Cond_Number = numeric(),
  stringsAsFactors = FALSE
)

for (i in 1:nrow(test_params)) {
  H <- hsmc(par = test_params[i, ], data = data)
  eigs <- eigen(H, only.values = TRUE)$values

  hess_properties <- rbind(hess_properties, data.frame(
    Gamma = test_params[i, 1],
    Delta = test_params[i, 2],
    Lambda = test_params[i, 3],
    Determinant = det(H),
    Trace = sum(diag(H)),
    Min_Eigenval = min(eigs),
    Max_Eigenval = max(eigs),
    Cond_Number = max(eigs) / min(eigs)
  ))
}

cat("\nHessian Properties at Different Points:\n")
print(hess_properties, digits = 4, row.names = FALSE)


## Example 5: Curvature Visualization (All pairs side by side)

# Create grids around MLE with wider range (±1.5)
gamma_grid <- seq(mle[1] - 1.5, mle[1] + 1.5, length.out = 25)
delta_grid <- seq(mle[2] - 1.5, mle[2] + 1.5, length.out = 25)
lambda_grid <- seq(mle[3] - 1.5, mle[3] + 1.5, length.out = 25)

gamma_grid <- gamma_grid[gamma_grid > 0]
delta_grid <- delta_grid[delta_grid > 0]
lambda_grid <- lambda_grid[lambda_grid > 0]

# Compute curvature measures for all pairs
determinant_surface_gd <- matrix(NA, nrow = length(gamma_grid), ncol = length(delta_grid))
trace_surface_gd <- matrix(NA, nrow = length(gamma_grid), ncol = length(delta_grid))

determinant_surface_gl <- matrix(NA, nrow = length(gamma_grid), ncol = length(lambda_grid))
trace_surface_gl <- matrix(NA, nrow = length(gamma_grid), ncol = length(lambda_grid))

determinant_surface_dl <- matrix(NA, nrow = length(delta_grid), ncol = length(lambda_grid))
trace_surface_dl <- matrix(NA, nrow = length(delta_grid), ncol = length(lambda_grid))

# Gamma vs Delta
for (i in seq_along(gamma_grid)) {
  for (j in seq_along(delta_grid)) {
    H <- hsmc(c(gamma_grid[i], delta_grid[j], mle[3]), data)
    determinant_surface_gd[i, j] <- det(H)
    trace_surface_gd[i, j] <- sum(diag(H))
  }
}
```

```
# Gamma vs Lambda
for (i in seq_along(gamma_grid)) {
  for (j in seq_along(lambda_grid)) {
    H <- hsmc(c(gamma_grid[i], mle[2], lambda_grid[j]), data)
    determinant_surface_gl[i, j] <- det(H)
    trace_surface_gl[i, j] <- sum(diag(H))
  }
}

# Delta vs Lambda
for (i in seq_along(delta_grid)) {
  for (j in seq_along(lambda_grid)) {
    H <- hsmc(c(mle[1], delta_grid[i], lambda_grid[j]), data)
    determinant_surface_dl[i, j] <- det(H)
    trace_surface_dl[i, j] <- sum(diag(H))
  }
}

# Plot

# Determinant plots
contour(gamma_grid, delta_grid, determinant_surface_gd,
        xlab = expression(gamma), ylab = expression(delta),
        main = "Determinant: Gamma vs Delta", las = 1,
        col = "#2E4057", lwd = 1.5, nlevels = 15)
points(mle[1], mle[2], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[2], pch = 17, col = "#006400", cex = 1.5)
grid(col = "gray90")

contour(gamma_grid, lambda_grid, determinant_surface_gl,
        xlab = expression(gamma), ylab = expression(lambda),
        main = "Determinant: Gamma vs Lambda", las = 1,
        col = "#2E4057", lwd = 1.5, nlevels = 15)
points(mle[1], mle[3], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[3], pch = 17, col = "#006400", cex = 1.5)
grid(col = "gray90")

contour(delta_grid, lambda_grid, determinant_surface_dl,
        xlab = expression(delta), ylab = expression(lambda),
        main = "Determinant: Delta vs Lambda", las = 1,
        col = "#2E4057", lwd = 1.5, nlevels = 15)
points(mle[2], mle[3], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[2], true_params[3], pch = 17, col = "#006400", cex = 1.5)
grid(col = "gray90")

# Trace plots
contour(gamma_grid, delta_grid, trace_surface_gd,
        xlab = expression(gamma), ylab = expression(delta),
        main = "Trace: Gamma vs Delta", las = 1,
        col = "#2E4057", lwd = 1.5, nlevels = 15)
points(mle[1], mle[2], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[2], pch = 17, col = "#006400", cex = 1.5)
grid(col = "gray90")
```

```
contour(gamma_grid, lambda_grid, trace_surface_gl,
        xlab = expression(gamma), ylab = expression(lambda),
        main = "Trace: Gamma vs Lambda", las = 1,
        col = "#2E4057", lwd = 1.5, nlevels = 15)
points(mle[1], mle[3], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[3], pch = 17, col = "#006400", cex = 1.5)
grid(col = "gray90")

contour(delta_grid, lambda_grid, trace_surface_dl,
        xlab = expression(delta), ylab = expression(lambda),
        main = "Trace: Delta vs Lambda", las = 1,
        col = "#2E4057", lwd = 1.5, nlevels = 15)
points(mle[2], mle[3], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[2], true_params[3], pch = 17, col = "#006400", cex = 1.5)
grid(col = "gray90")

legend("topright",
       legend = c("MLE", "True"),
       col = c("#8B0000", "#006400"),
       pch = c(19, 17),
       bty = "n", cex = 0.8)

## Example 6: Confidence Ellipses (All pairs side by side)

# Extract all 2x2 submatrices
vcov_gd <- vcov_matrix[1:2, 1:2]
vcov_gl <- vcov_matrix[c(1, 3), c(1, 3)]
vcov_dl <- vcov_matrix[2:3, 2:3]

# Create confidence ellipses
theta <- seq(0, 2 * pi, length.out = 100)
chi2_val <- qchisq(0.95, df = 2)

# Gamma vs Delta ellipse
eig_decomp_gd <- eigen(vcov_gd)
ellipse_gd <- matrix(NA, nrow = 100, ncol = 2)
for (i in 1:100) {
  v <- c(cos(theta[i]), sin(theta[i]))
  ellipse_gd[i, ] <- mle[1:2] + sqrt(chi2_val) *
    (eig_decomp_gd$vectors %*% diag(sqrt(eig_decomp_gd$values)) %*% v)
}

# Gamma vs Lambda ellipse
eig_decomp_gl <- eigen(vcov_gl)
ellipse_gl <- matrix(NA, nrow = 100, ncol = 2)
for (i in 1:100) {
  v <- c(cos(theta[i]), sin(theta[i]))
  ellipse_gl[i, ] <- mle[c(1, 3)] + sqrt(chi2_val) *
    (eig_decomp_gl$vectors %*% diag(sqrt(eig_decomp_gl$values)) %*% v)
}

# Delta vs Lambda ellipse
```

```
eig_decomp_dl <- eigen(vcov_dl)
ellipse_dl <- matrix(NA, nrow = 100, ncol = 2)
for (i in 1:100) {
  v <- c(cos(theta[i]), sin(theta[i]))
  ellipse_dl[i, ] <- mle[2:3] + sqrt(chi2_val) *
    (eig_decomp_dl$vectors %*% diag(sqrt(eig_decomp_dl$values)) %*% v)
}

# Marginal confidence intervals
se_gd <- sqrt(diag(vcov_gd))
ci_gamma_gd <- mle[1] + c(-1, 1) * 1.96 * se_gd[1]
ci_delta_gd <- mle[2] + c(-1, 1) * 1.96 * se_gd[2]

se_gl <- sqrt(diag(vcov_gl))
ci_gamma_gl <- mle[1] + c(-1, 1) * 1.96 * se_gl[1]
ci_lambda_gl <- mle[3] + c(-1, 1) * 1.96 * se_gl[2]

se_dl <- sqrt(diag(vcov_dl))
ci_delta_dl <- mle[2] + c(-1, 1) * 1.96 * se_dl[1]
ci_lambda_dl <- mle[3] + c(-1, 1) * 1.96 * se_dl[2]

# Plot

# Gamma vs Delta
plot(ellipse_gd[, 1], ellipse_gd[, 2], type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(gamma), ylab = expression(delta),
     main = "Gamma vs Delta", las = 1, xlim = range(ellipse_gd[, 1], ci_gamma_gd),
     ylim = range(ellipse_gd[, 2], ci_delta_gd))
abline(v = ci_gamma_gd, col = "#808080", lty = 3, lwd = 1.5)
abline(h = ci_delta_gd, col = "#808080", lty = 3, lwd = 1.5)
points(mle[1], mle[2], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[2], pch = 17, col = "#006400", cex = 1.5)
grid(col = "gray90")

# Gamma vs Lambda
plot(ellipse_gl[, 1], ellipse_gl[, 2], type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(gamma), ylab = expression(lambda),
     main = "Gamma vs Lambda", las = 1, xlim = range(ellipse_gl[, 1], ci_gamma_gl),
     ylim = range(ellipse_gl[, 2], ci_lambda_gl))
abline(v = ci_gamma_gl, col = "#808080", lty = 3, lwd = 1.5)
abline(h = ci_lambda_gl, col = "#808080", lty = 3, lwd = 1.5)
points(mle[1], mle[3], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[3], pch = 17, col = "#006400", cex = 1.5)
grid(col = "gray90")

# Delta vs Lambda
plot(ellipse_dl[, 1], ellipse_dl[, 2], type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(delta), ylab = expression(lambda),
     main = "Delta vs Lambda", las = 1, xlim = range(ellipse_dl[, 1], ci_delta_dl),
     ylim = range(ellipse_dl[, 2], ci_lambda_dl))
abline(v = ci_delta_dl, col = "#808080", lty = 3, lwd = 1.5)
abline(h = ci_lambda_dl, col = "#808080", lty = 3, lwd = 1.5)
points(mle[2], mle[3], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[2], true_params[3], pch = 17, col = "#006400", cex = 1.5)
grid(col = "gray90")

legend("topright",
       legend = c("MLE", "True", "95% CR", "Marginal 95% CI"),
       col = c("#8B0000", "#006400", "#2E4057", "#808080"),
       pch = c(19, 17, NA, NA),
       lty = c(NA, NA, 1, 3),
       lwd = c(NA, NA, 2, 1.5),
       bty = "n", cex = 0.8)
```

llbeta — *Negative Log-Likelihood for the Beta Distribution (gamma, delta+1 Parameterization)*

Description

Computes the negative log-likelihood function for the standard Beta distribution, using a parameterization common in generalized distribution families. The distribution is parameterized by `gamma` (γ) and `delta` (δ), corresponding to the standard Beta distribution with shape parameters `shape1 = gamma` and `shape2 = delta + 1`. This function is suitable for maximum likelihood estimation.

Usage

```
llbeta(par, data)
```

Arguments

`par`	A numeric vector of length 2 containing the distribution parameters in the order: `gamma` ($\gamma > 0$), `delta` ($\delta \geq 0$).
`data`	A numeric vector of observations. All values must be strictly between 0 and 1 (exclusive).

Details

This function calculates the negative log-likelihood for a Beta distribution with parameters `shape1 = gamma` (γ) and `shape2 = delta + 1` ($\delta + 1$). The probability density function (PDF) is:

$$f(x|\gamma, \delta) = \frac{x^{\gamma-1}(1-x)^{\delta}}{B(\gamma, \delta+1)}$$

for $0 < x < 1$, where $B(a, b)$ is the Beta function (`beta`). The log-likelihood function $\ell(\theta|\mathbf{x})$ for a sample $\mathbf{x} = (x_1, \ldots, x_n)$ is $\sum_{i=1}^{n} \ln f(x_i|\theta)$:

$$\ell(\theta|\mathbf{x}) = \sum_{i=1}^{n} [(\gamma - 1)\ln(x_i) + \delta \ln(1 - x_i)] - n \ln B(\gamma, \delta + 1)$$

where $\theta = (\gamma, \delta)$. This function computes and returns the *negative* log-likelihood, $-\ell(\theta|\mathbf{x})$, suitable for minimization using optimization routines like optim. It is equivalent to the negative log-likelihood of the GKw distribution (llgkw) evaluated at $\alpha = 1, \beta = 1, \lambda = 1$, and also to the negative log-likelihood of the McDonald distribution (llmc) evaluated at $\lambda = 1$. The term $\ln B(\gamma, \delta+1)$ is typically computed using log-gamma functions (lgamma) for numerical stability.

Value

Returns a single double value representing the negative log-likelihood ($-\ell(\theta|\mathbf{x})$). Returns Inf if any parameter values in par are invalid according to their constraints, or if any value in data is not in the interval (0, 1).

Author(s)

Lopes, J. E.

References

Johnson, N. L., Kotz, S., & Balakrishnan, N. (1995). *Continuous Univariate Distributions, Volume 2* (2nd ed.). Wiley.

Cordeiro, G. M., & de Castro, M. (2011). A new family of generalized distributions. *Journal of Statistical Computation and Simulation*,

See Also

llgkw, llmc (related negative log-likelihoods), dbeta_, pbeta_, qbeta_, rbeta_, grbeta (gradient, if available), hsbeta (Hessian, if available), optim, lbeta.

Examples

```
## Example 1: Basic Log-Likelihood Evaluation

# Generate sample data
set.seed(123)
n <- 1000
true_params <- c(gamma = 2.0, delta = 3.0)
data <- rbeta_(n, gamma = true_params[1], delta = true_params[2])

# Evaluate negative log-likelihood at true parameters
nll_true <- llbeta(par = true_params, data = data)
cat("Negative log-likelihood at true parameters:", nll_true, "\n")

# Evaluate at different parameter values
test_params <- rbind(
  c(1.5, 2.5),
  c(2.0, 3.0),
  c(2.5, 3.5)
)

nll_values <- apply(test_params, 1, function(p) llbeta(p, data))
results <- data.frame(
  Gamma = test_params[, 1],
  Delta = test_params[, 2],
  NegLogLik = nll_values
)
print(results, digits = 4)


## Example 2: Maximum Likelihood Estimation

# Optimization using L-BFGS-B with bounds
fit <- optim(
  par = c(1.5, 2.5),
  fn = llbeta,
  gr = grbeta,
  data = data,
  method = "L-BFGS-B",
  lower = c(0.01, 0.01),
  upper = c(100, 100),
  hessian = TRUE
)

mle <- fit$par
names(mle) <- c("gamma", "delta")
se <- sqrt(diag(solve(fit$hessian)))

results <- data.frame(
  Parameter = c("gamma", "delta"),
  True = true_params,
  MLE = mle,
  SE = se,
  CI_Lower = mle - 1.96 * se,
  CI_Upper = mle + 1.96 * se
)
print(results, digits = 4)

cat(sprintf("\nMLE corresponds approx to Beta(%.2f, %.2f)\n",
    mle[1], mle[2] + 1))
cat("True corresponds to Beta(%.2f, %.2f)\n",
    true_params[1], true_params[2] + 1)

cat("\nNegative log-likelihood at MLE:", fit$value, "\n")
cat("AIC:", 2 * fit$value + 2 * length(mle), "\n")
cat("BIC:", 2 * fit$value + length(mle) * log(n), "\n")


## Example 3: Comparing Optimization Methods

methods <- c("BFGS", "L-BFGS-B", "Nelder-Mead", "CG")
start_params <- c(1.5, 2.5)

comparison <- data.frame(
  Method = character(),
  Gamma = numeric(),
  Delta = numeric(),
  NegLogLik = numeric(),
  Convergence = integer(),
  stringsAsFactors = FALSE
)

for (method in methods) {
  if (method %in% c("BFGS", "CG")) {
    fit_temp <- optim(
      par = start_params,
      fn = llbeta,
      gr = grbeta,
      data = data,
      method = method
    )
  } else if (method == "L-BFGS-B") {
    fit_temp <- optim(
      par = start_params,
      fn = llbeta,
      gr = grbeta,
      data = data,
      method = method,
      lower = c(0.01, 0.01),
      upper = c(100, 100)
    )
  } else {
    fit_temp <- optim(
      par = start_params,
      fn = llbeta,
      data = data,
      method = method
    )
  }

  comparison <- rbind(comparison, data.frame(
    Method = method,
    Gamma = fit_temp$par[1],
    Delta = fit_temp$par[2],
    NegLogLik = fit_temp$value,
    Convergence = fit_temp$convergence,
    stringsAsFactors = FALSE
  ))
}

print(comparison, digits = 4, row.names = FALSE)


## Example 4: Likelihood Ratio Test

# Test H0: delta = 3 vs H1: delta free
loglik_full <- -fit$value

restricted_ll <- function(params_restricted, data, delta_fixed) {
```

```
  llbeta(par = c(params_restricted[1], delta_fixed), data = data)
}

fit_restricted <- optim(
  par = mle[1],
  fn = restricted_ll,
  data = data,
  delta_fixed = 3,
  method = "BFGS"
)

loglik_restricted <- -fit_restricted$value
lr_stat <- 2 * (loglik_full - loglik_restricted)
p_value <- pchisq(lr_stat, df = 1, lower.tail = FALSE)

cat("LR Statistic:", round(lr_stat, 4), "\n")
cat("P-value:", format.pval(p_value, digits = 4), "\n")


## Example 5: Univariate Profile Likelihoods

# Profile for gamma
gamma_grid <- seq(mle[1] - 1.5, mle[1] + 1.5, length.out = 50)
gamma_grid <- gamma_grid[gamma_grid > 0]
profile_ll_gamma <- numeric(length(gamma_grid))

for (i in seq_along(gamma_grid)) {
  profile_fit <- optim(
    par = mle[2],
    fn = function(p) llbeta(c(gamma_grid[i], p), data),
    method = "BFGS"
  )
  profile_ll_gamma[i] <- -profile_fit$value
}

# Profile for delta
delta_grid <- seq(mle[2] - 1.5, mle[2] + 1.5, length.out = 50)
delta_grid <- delta_grid[delta_grid > 0]
profile_ll_delta <- numeric(length(delta_grid))

for (i in seq_along(delta_grid)) {
  profile_fit <- optim(
    par = mle[1],
    fn = function(p) llbeta(c(p, delta_grid[i]), data),
    method = "BFGS"
  )
  profile_ll_delta[i] <- -profile_fit$value
}

# 95% confidence threshold
chi_crit <- qchisq(0.95, df = 1)
threshold <- max(profile_ll_gamma) - chi_crit / 2
```

```
# Plot

plot(gamma_grid, profile_ll_gamma, type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(gamma), ylab = "Profile Log-Likelihood",
     main = expression(paste("Profile: ", gamma)), las = 1)
abline(v = mle[1], col = "#8B0000", lty = 2, lwd = 2)
abline(v = true_params[1], col = "#006400", lty = 2, lwd = 2)
abline(h = threshold, col = "#808080", lty = 3, lwd = 1.5)
legend("topright", legend = c("MLE", "True", "95% CI"),
       col = c("#8B0000", "#006400", "#808080"),
       lty = c(2, 2, 3), lwd = 2, bty = "n", cex = 0.8)
grid(col = "gray90")

plot(delta_grid, profile_ll_delta, type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(delta), ylab = "Profile Log-Likelihood",
     main = expression(paste("Profile: ", delta)), las = 1)
abline(v = mle[2], col = "#8B0000", lty = 2, lwd = 2)
abline(v = true_params[2], col = "#006400", lty = 2, lwd = 2)
abline(h = threshold, col = "#808080", lty = 3, lwd = 1.5)
legend("topright", legend = c("MLE", "True", "95% CI"),
       col = c("#8B0000", "#006400", "#808080"),
       lty = c(2, 2, 3), lwd = 2, bty = "n", cex = 0.8)
grid(col = "gray90")


## Example 6: 2D Log-Likelihood Surface (Gamma vs Delta)

# Create 2D grid with wider range (±1.5)
gamma_2d <- seq(mle[1] - 1.5, mle[1] + 1.5, length.out = round(n/25))
delta_2d <- seq(mle[2] - 1.5, mle[2] + 1.5, length.out = round(n/25))
gamma_2d <- gamma_2d[gamma_2d > 0]
delta_2d <- delta_2d[delta_2d > 0]

# Compute log-likelihood surface
ll_surface_gd <- matrix(NA, nrow = length(gamma_2d), ncol = length(delta_2d))

for (i in seq_along(gamma_2d)) {
  for (j in seq_along(delta_2d)) {
    ll_surface_gd[i, j] <- -llbeta(c(gamma_2d[i], delta_2d[j]), data)
  }
}

# Confidence region levels
max_ll_gd <- max(ll_surface_gd, na.rm = TRUE)
levels_90_gd <- max_ll_gd - qchisq(0.90, df = 2) / 2
levels_95_gd <- max_ll_gd - qchisq(0.95, df = 2) / 2
levels_99_gd <- max_ll_gd - qchisq(0.99, df = 2) / 2

# Plot contour

contour(gamma_2d, delta_2d, ll_surface_gd,
        xlab = expression(gamma), ylab = expression(delta),
        main = "2D Log-Likelihood: Gamma vs Delta",
        levels = seq(min(ll_surface_gd, na.rm = TRUE), max_ll_gd, length.out = 20),
        col = "#2E4057", las = 1, lwd = 1)

contour(gamma_2d, delta_2d, ll_surface_gd,
        levels = c(levels_90_gd, levels_95_gd, levels_99_gd),
        col = c("#FFA07A", "#FF6347", "#8B0000"),
        lwd = c(2, 2.5, 3), lty = c(3, 2, 1),
        add = TRUE, labcex = 0.8)

points(mle[1], mle[2], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[2], pch = 17, col = "#006400", cex = 1.5)

legend("topright",
       legend = c("MLE", "True", "90% CR", "95% CR", "99% CR"),
       col = c("#8B0000", "#006400", "#FFA07A", "#FF6347", "#8B0000"),
       pch = c(19, 17, NA, NA, NA),
       lty = c(NA, NA, 3, 2, 1),
       lwd = c(NA, NA, 2, 2.5, 3),
       bty = "n", cex = 0.8)
grid(col = "gray90")
```

llbkw — *Negative Log-Likelihood for Beta-Kumaraswamy (BKw) Distribution*

Description

Computes the negative log-likelihood function for the Beta-Kumaraswamy (BKw) distribution with parameters `alpha` (α), `beta` (β), `gamma` (γ), and `delta` (δ), given a vector of observations. This distribution is the special case of the Generalized Kumaraswamy (GKw) distribution where $\lambda = 1$. This function is typically used for maximum likelihood estimation via numerical optimization.

Usage

```
llbkw(par, data)
```

Arguments

`par`	A numeric vector of length 4 containing the distribution parameters in the order: `alpha` ($\alpha > 0$), `beta` ($\beta > 0$), `gamma` ($\gamma > 0$), `delta` ($\delta \geq 0$).
`data`	A numeric vector of observations. All values must be strictly between 0 and 1 (exclusive).

Details

The Beta-Kumaraswamy (BKw) distribution is the GKw distribution (dgkw) with $\lambda = 1$. Its probability density function (PDF) is:

$$f(x|\theta) = \frac{\alpha\beta}{B(\gamma, \delta+1)} x^{\alpha-1} \bigl(1 - x^\alpha\bigr)^{\beta(\delta+1)-1} \bigl[1 - \bigl(1 - x^\alpha\bigr)^\beta\bigr]^{\gamma-1}$$

for $0 < x < 1$, $\theta = (\alpha, \beta, \gamma, \delta)$, and $B(a, b)$ is the Beta function (beta). The log-likelihood function $\ell(\theta|\mathbf{x})$ for a sample $\mathbf{x} = (x_1, \ldots, x_n)$ is $\sum_{i=1}^n \ln f(x_i|\theta)$:

$$\ell(\theta|\mathbf{x}) = n[\ln(\alpha) + \ln(\beta) - \ln B(\gamma, \delta+1)] + \sum_{i=1}^{n} [(\alpha-1)\ln(x_i) + (\beta(\delta+1)-1)\ln(v_i) + (\gamma-1)\ln(w_i)]$$

where:

- $v_i = 1 - x_i^\alpha$
- $w_i = 1 - v_i^\beta = 1 - (1 - x_i^\alpha)^\beta$

This function computes and returns the *negative* log-likelihood, $-\ell(\theta|\mathbf{x})$, suitable for minimization using optimization routines like optim. Numerical stability is maintained similarly to llgkw.

Value

Returns a single double value representing the negative log-likelihood ($-\ell(\theta|\mathbf{x})$). Returns Inf if any parameter values in par are invalid according to their constraints, or if any value in data is not in the interval (0, 1).

Author(s)

Lopes, J. E.

References

Cordeiro, G. M., & de Castro, M. (2011). A new family of generalized distributions. *Journal of Statistical Computation and Simulation*

Kumaraswamy, P. (1980). A generalized probability density function for double-bounded random processes. *Journal of Hydrology*, *46*(1-2), 79-88.

See Also

llgkw (parent distribution negative log-likelihood), dbkw, pbkw, qbkw, rbkw, grbkw (gradient, if available), hsbkw (Hessian, if available), optim, lbeta

Examples

```
## Example 1: Basic Log-Likelihood Evaluation
# Generate sample data
set.seed(2203)
n <- 1000
true_params <- c(alpha = 2.0, beta = 1.5, gamma = 1.5, delta = 0.5)
```

```
data <- rbkw(n, alpha = true_params[1], beta = true_params[2],
             gamma = true_params[3], delta = true_params[4])

# Evaluate negative log-likelihood at true parameters
nll_true <- llbkw(par = true_params, data = data)
cat("Negative log-likelihood at true parameters:", nll_true, "\n")

# Evaluate at different parameter values
test_params <- rbind(
  c(1.5, 1.0, 1.0, 0.3),
  c(2.0, 1.5, 1.5, 0.5),
  c(2.5, 2.0, 2.0, 0.7)
)

nll_values <- apply(test_params, 1, function(p) llbkw(p, data))
results <- data.frame(
  Alpha = test_params[, 1],
  Beta = test_params[, 2],
  Gamma = test_params[, 3],
  Delta = test_params[, 4],
  NegLogLik = nll_values
)
print(results, digits = 4)


## Example 2: Maximum Likelihood Estimation

# Optimization using BFGS with no analytical gradient
fit <- optim(
  par = c(0.5, 1, 1.1, 0.3),
  fn = llbkw,
  # gr = grbkw,
  data = data,
  method = "BFGS",
  control = list(maxit = 2000),
  hessian = TRUE
)

mle <- fit$par
names(mle) <- c("alpha", "beta", "gamma", "delta")
se <- sqrt(diag(solve(fit$hessian)))

results <- data.frame(
  Parameter = c("alpha", "beta", "gamma", "delta"),
  True = true_params,
  MLE = mle,
  SE = se,
  CI_Lower = mle - 1.96 * se,
  CI_Upper = mle + 1.96 * se
)
print(results, digits = 4)

cat("\nNegative log-likelihood at MLE:", fit$value, "\n")
cat("AIC:", 2 * fit$value + 2 * length(mle), "\n")
cat("BIC:", 2 * fit$value + length(mle) * log(n), "\n")


## Example 3: Comparing Optimization Methods

methods <- c("BFGS", "L-BFGS-B", "Nelder-Mead", "CG")
start_params <- c(1.8, 1.2, 1.1, 0.3)

comparison <- data.frame(
  Method = character(),
  Alpha = numeric(),
  Beta = numeric(),
  Gamma = numeric(),
  Delta = numeric(),
  NegLogLik = numeric(),
  Convergence = integer(),
  stringsAsFactors = FALSE
)

for (method in methods) {
  if (method %in% c("BFGS", "CG")) {
    fit_temp <- optim(
      par = start_params,
      fn = llbkw,
      gr = grbkw,
      data = data,
      method = method
    )
  } else if (method == "L-BFGS-B") {
    fit_temp <- optim(
      par = start_params,
      fn = llbkw,
      gr = grbkw,
      data = data,
      method = method,
      lower = c(0.01, 0.01, 0.01, 0.01),
      upper = c(100, 100, 100, 100)
    )
  } else {
    fit_temp <- optim(
      par = start_params,
      fn = llbkw,
      data = data,
      method = method
    )
  }

  comparison <- rbind(comparison, data.frame(
    Method = method,
    Alpha = fit_temp$par[1],
    Beta = fit_temp$par[2],
    Gamma = fit_temp$par[3],
    Delta = fit_temp$par[4],
    NegLogLik = fit_temp$value,
    Convergence = fit_temp$convergence,
    stringsAsFactors = FALSE
  ))
}

print(comparison, digits = 4, row.names = FALSE)


## Example 4: Likelihood Ratio Test

# Test H0: delta = 0.5 vs H1: delta free
loglik_full <- -fit$value

restricted_ll <- function(params_restricted, data, delta_fixed) {
  llbkw(par = c(params_restricted[1], params_restricted[2],
                params_restricted[3], delta_fixed), data = data)
}

fit_restricted <- optim(
  par = mle[1:3],
  fn = restricted_ll,
  data = data,
  delta_fixed = 0.5,
  method = "Nelder-Mead"
)

loglik_restricted <- -fit_restricted$value
lr_stat <- 2 * (loglik_full - loglik_restricted)
p_value <- pchisq(lr_stat, df = 1, lower.tail = FALSE)

cat("LR Statistic:", round(lr_stat, 4), "\n")
cat("P-value:", format.pval(p_value, digits = 4), "\n")


## Example 5: Univariate Profile Likelihoods

# Profile for alpha
alpha_grid <- seq(mle[1] - 1.5, mle[1] + 1.5, length.out = 50)
alpha_grid <- alpha_grid[alpha_grid > 0]
profile_ll_alpha <- numeric(length(alpha_grid))

for (i in seq_along(alpha_grid)) {
  profile_fit <- optim(
    par = mle[-1],
    fn = function(p) llbkw(c(alpha_grid[i], p), data),
    method = "Nelder-Mead"
  )
  profile_ll_alpha[i] <- -profile_fit$value
}

# Profile for beta
```

```
beta_grid <- seq(mle[2] - 1.5, mle[2] + 1.5, length.out = 50)
beta_grid <- beta_grid[beta_grid > 0]
profile_ll_beta <- numeric(length(beta_grid))

for (i in seq_along(beta_grid)) {
  profile_fit <- optim(
    par = c(mle[1], mle[3], mle[4]),
    fn = function(p) llbkw(c(mle[1], beta_grid[i], p[1], p[2]), data),
    method = "Nelder-Mead"
  )
  profile_ll_beta[i] <- -profile_fit$value
}

# Profile for gamma
gamma_grid <- seq(mle[3] - 1.5, mle[3] + 1.5, length.out = 50)
gamma_grid <- gamma_grid[gamma_grid > 0]
profile_ll_gamma <- numeric(length(gamma_grid))

for (i in seq_along(gamma_grid)) {
  profile_fit <- optim(
    par = c(mle[1], mle[2], mle[4]),
    fn = function(p) llbkw(c(p[1], mle[2], gamma_grid[i], p[2]), data),
    method = "Nelder-Mead"
  )
  profile_ll_gamma[i] <- -profile_fit$value
}

# Profile for delta
delta_grid <- seq(mle[4] - 1.5, mle[4] + 1.5, length.out = 50)
delta_grid <- delta_grid[delta_grid > 0]
profile_ll_delta <- numeric(length(delta_grid))

for (i in seq_along(delta_grid)) {
  profile_fit <- optim(
    par = mle[-4],
    fn = function(p) llbkw(c(p[1], p[2], p[3], delta_grid[i]), data),
    method = "Nelder-Mead"
  )
  profile_ll_delta[i] <- -profile_fit$value
}

# 95% confidence threshold
chi_crit <- qchisq(0.95, df = 1)
threshold <- max(profile_ll_alpha) - chi_crit / 2

# Plot all profiles

plot(alpha_grid, profile_ll_alpha, type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(alpha), ylab = "Profile Log-Likelihood",
     main = expression(paste("Profile: ", alpha)), las = 1)
abline(v = mle[1], col = "#8B0000", lty = 2, lwd = 2)
abline(v = true_params[1], col = "#006400", lty = 2, lwd = 2)
abline(h = threshold, col = "#808080", lty = 3, lwd = 1.5)
```

```
legend("topright", legend = c("MLE", "True", "95% CI"),
       col = c("#8B0000", "#006400", "#808080"),
       lty = c(2, 2, 3), lwd = 2, bty = "n", cex = 0.8)
grid(col = "gray90")

plot(beta_grid, profile_ll_beta, type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(beta), ylab = "Profile Log-Likelihood",
     main = expression(paste("Profile: ", beta)), las = 1)
abline(v = mle[2], col = "#8B0000", lty = 2, lwd = 2)
abline(v = true_params[2], col = "#006400", lty = 2, lwd = 2)
abline(h = threshold, col = "#808080", lty = 3, lwd = 1.5)
legend("topright", legend = c("MLE", "True", "95% CI"),
       col = c("#8B0000", "#006400", "#808080"),
       lty = c(2, 2, 3), lwd = 2, bty = "n", cex = 0.8)
grid(col = "gray90")

plot(gamma_grid, profile_ll_gamma, type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(gamma), ylab = "Profile Log-Likelihood",
     main = expression(paste("Profile: ", gamma)), las = 1)
abline(v = mle[3], col = "#8B0000", lty = 2, lwd = 2)
abline(v = true_params[3], col = "#006400", lty = 2, lwd = 2)
abline(h = threshold, col = "#808080", lty = 3, lwd = 1.5)
legend("topright", legend = c("MLE", "True", "95% CI"),
       col = c("#8B0000", "#006400", "#808080"),
       lty = c(2, 2, 3), lwd = 2, bty = "n", cex = 0.8)
grid(col = "gray90")

plot(delta_grid, profile_ll_delta, type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(delta), ylab = "Profile Log-Likelihood",
     main = expression(paste("Profile: ", delta)), las = 1)
abline(v = mle[4], col = "#8B0000", lty = 2, lwd = 2)
abline(v = true_params[4], col = "#006400", lty = 2, lwd = 2)
abline(h = threshold, col = "#808080", lty = 3, lwd = 1.5)
legend("topright", legend = c("MLE", "True", "95% CI"),
       col = c("#8B0000", "#006400", "#808080"),
       lty = c(2, 2, 3), lwd = 2, bty = "n", cex = 0.8)
grid(col = "gray90")


## Example 6: 2D Log-Likelihood Surfaces (Selected pairs)

# Create 2D grids with wider range (±1.5)
alpha_2d <- seq(mle[1] - 1.5, mle[1] + 1.5, length.out = round(n/25))
beta_2d <- seq(mle[2] - 1.5, mle[2] + 1.5, length.out = round(n/25))
gamma_2d <- seq(mle[3] - 1.5, mle[3] + 1.5, length.out = round(n/25))
delta_2d <- seq(mle[4] - 1.5, mle[4] + 1.5, length.out = round(n/25))

alpha_2d <- alpha_2d[alpha_2d > 0]
beta_2d <- beta_2d[beta_2d > 0]
gamma_2d <- gamma_2d[gamma_2d > 0]
delta_2d <- delta_2d[delta_2d > 0]

# Compute selected log-likelihood surfaces
```

```
ll_surface_ab <- matrix(NA, nrow = length(alpha_2d), ncol = length(beta_2d))
ll_surface_ag <- matrix(NA, nrow = length(alpha_2d), ncol = length(gamma_2d))
ll_surface_bd <- matrix(NA, nrow = length(beta_2d), ncol = length(delta_2d))

# Alpha vs Beta
for (i in seq_along(alpha_2d)) {
  for (j in seq_along(beta_2d)) {
    ll_surface_ab[i, j] <- -llbkw(c(alpha_2d[i], beta_2d[j], mle[3], mle[4]), data)
  }
}

# Alpha vs Gamma
for (i in seq_along(alpha_2d)) {
  for (j in seq_along(gamma_2d)) {
    ll_surface_ag[i, j] <- -llbkw(c(alpha_2d[i], mle[2], gamma_2d[j], mle[4]), data)
  }
}

# Beta vs Delta
for (i in seq_along(beta_2d)) {
  for (j in seq_along(delta_2d)) {
    ll_surface_bd[i, j] <- -llbkw(c(mle[1], beta_2d[i], mle[3], delta_2d[j]), data)
  }
}

# Confidence region levels
max_ll_ab <- max(ll_surface_ab, na.rm = TRUE)
max_ll_ag <- max(ll_surface_ag, na.rm = TRUE)
max_ll_bd <- max(ll_surface_bd, na.rm = TRUE)

levels_95_ab <- max_ll_ab - qchisq(0.95, df = 2) / 2
levels_95_ag <- max_ll_ag - qchisq(0.95, df = 2) / 2
levels_95_bd <- max_ll_bd - qchisq(0.95, df = 2) / 2

# Plot selected surfaces

# Alpha vs Beta
contour(alpha_2d, beta_2d, ll_surface_ab,
        xlab = expression(alpha), ylab = expression(beta),
        main = "Alpha vs Beta", las = 1,
        levels = seq(min(ll_surface_ab, na.rm = TRUE), max_ll_ab, length.out = 20),
        col = "#2E4057", lwd = 1)
contour(alpha_2d, beta_2d, ll_surface_ab,
        levels = levels_95_ab, col = "#FF6347", lwd = 2.5, lty = 1, add = TRUE)
points(mle[1], mle[2], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[2], pch = 17, col = "#006400", cex = 1.5)
grid(col = "gray90")

# Alpha vs Gamma
contour(alpha_2d, gamma_2d, ll_surface_ag,
        xlab = expression(alpha), ylab = expression(gamma),
        main = "Alpha vs Gamma", las = 1,
        levels = seq(min(ll_surface_ag, na.rm = TRUE), max_ll_ag, length.out = 20),
        col = "#2E4057", lwd = 1)
contour(alpha_2d, gamma_2d, ll_surface_ag,
        levels = levels_95_ag, col = "#FF6347", lwd = 2.5, lty = 1, add = TRUE)
points(mle[1], mle[3], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[3], pch = 17, col = "#006400", cex = 1.5)
grid(col = "gray90")

# Beta vs Delta
contour(beta_2d, delta_2d, ll_surface_bd,
        xlab = expression(beta), ylab = expression(delta),
        main = "Beta vs Delta", las = 1,
        levels = seq(min(ll_surface_bd, na.rm = TRUE), max_ll_bd, length.out = 20),
        col = "#2E4057", lwd = 1)
contour(beta_2d, delta_2d, ll_surface_bd,
        levels = levels_95_bd, col = "#FF6347", lwd = 2.5, lty = 1, add = TRUE)
points(mle[2], mle[4], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[2], true_params[4], pch = 17, col = "#006400", cex = 1.5)
grid(col = "gray90")

legend("topright",
       legend = c("MLE", "True", "95% CR"),
       col = c("#8B0000", "#006400", "#FF6347"),
       pch = c(19, 17, NA),
       lty = c(NA, NA, 1),
       lwd = c(NA, NA, 2.5),
       bty = "n", cex = 0.8)
```

---

llekw — *Negative Log-Likelihood for the Exponentiated Kumaraswamy (EKw) Distribution*

Description

Computes the negative log-likelihood function for the Exponentiated Kumaraswamy (EKw) distribution with parameters `alpha` (α), `beta` (β), and `lambda` (λ), given a vector of observations. This distribution is the special case of the Generalized Kumaraswamy (GKw) distribution where $\gamma = 1$ and $\delta = 0$. This function is suitable for maximum likelihood estimation.

Usage

```
llekw(par, data)
```

Arguments

`par`	A numeric vector of length 3 containing the distribution parameters in the order: `alpha` ($\alpha > 0$), `beta` ($\beta > 0$), `lambda` ($\lambda > 0$).
`data`	A numeric vector of observations. All values must be strictly between 0 and 1 (exclusive).

Details

The Exponentiated Kumaraswamy (EKw) distribution is the GKw distribution (dekw) with $\gamma = 1$ and $\delta = 0$. Its probability density function (PDF) is:

$$f(x|\theta) = \lambda\alpha\beta x^{\alpha-1}(1-x^\alpha)^{\beta-1}\bigl[1-(1-x^\alpha)^\beta\bigr]^{\lambda-1}$$

for $0 < x < 1$ and $\theta = (\alpha, \beta, \lambda)$. The log-likelihood function $\ell(\theta|\mathbf{x})$ for a sample $\mathbf{x} = (x_1, \dots, x_n)$ is $\sum_{i=1}^n \ln f(x_i|\theta)$:

$$\ell(\theta|\mathbf{x}) = n[\ln(\lambda) + \ln(\alpha) + \ln(\beta)] + \sum_{i=1}^{n}[(\alpha-1)\ln(x_i) + (\beta-1)\ln(v_i) + (\lambda-1)\ln(w_i)]$$

where:

- $v_i = 1 - x_i^\alpha$
- $w_i = 1 - v_i^\beta = 1 - (1 - x_i^\alpha)^\beta$

This function computes and returns the *negative* log-likelihood, $-\ell(\theta|\mathbf{x})$, suitable for minimization using optimization routines like optim. Numerical stability is maintained similarly to llgkw.

Value

Returns a single double value representing the negative log-likelihood ($-\ell(\theta|\mathbf{x})$). Returns Inf if any parameter values in par are invalid according to their constraints, or if any value in data is not in the interval (0, 1).

Author(s)

Lopes, J. E.

References

Nadarajah, S., Cordeiro, G. M., & Ortega, E. M. (2012). The exponentiated Kumaraswamy distribution. *Journal of the Franklin Institute*, *349*(3),

Cordeiro, G. M., & de Castro, M. (2011). A new family of generalized distributions. *Journal of Statistical Computation and Simulation*,

Kumaraswamy, P. (1980). A generalized probability density function for double-bounded random processes. *Journal of Hydrology*, *46*(1-2), 79-88.

See Also

llgkw (parent distribution negative log-likelihood), dekw, pekw, qekw, rekw, grekw (gradient, if available), hsekw (Hessian, if available), optim

Examples

```
## Example 1: Basic Log-Likelihood Evaluation

# Generate sample data
set.seed(123)
n <- 1000
true_params <- c(alpha = 2.5, beta = 3.5, lambda = 2.0)
data <- rekw(n, alpha = true_params[1], beta = true_params[2],
             lambda = true_params[3])

# Evaluate negative log-likelihood at true parameters
nll_true <- llekw(par = true_params, data = data)
cat("Negative log-likelihood at true parameters:", nll_true, "\n")

# Evaluate at different parameter values
test_params <- rbind(
  c(2.0, 3.0, 1.5),
  c(2.5, 3.5, 2.0),
  c(3.0, 4.0, 2.5)
)

nll_values <- apply(test_params, 1, function(p) llekw(p, data))
results <- data.frame(
  Alpha = test_params[, 1],
  Beta = test_params[, 2],
  Lambda = test_params[, 3],
  NegLogLik = nll_values
)
print(results, digits = 4)


## Example 2: Maximum Likelihood Estimation

# Optimization using BFGS with analytical gradient
fit <- optim(
  par = c(2, 3, 1.5),
  fn = llekw,
  gr = grekw,
  data = data,
  method = "BFGS",
  hessian = TRUE
)

mle <- fit$par
names(mle) <- c("alpha", "beta", "lambda")
se <- sqrt(diag(solve(fit$hessian)))

results <- data.frame(
  Parameter = c("alpha", "beta", "lambda"),
  True = true_params,
  MLE = mle,
  SE = se,
  CI_Lower = mle - 1.96 * se,
  CI_Upper = mle + 1.96 * se
)
print(results, digits = 4)

cat("\nNegative log-likelihood at MLE:", fit$value, "\n")
cat("AIC:", 2 * fit$value + 2 * length(mle), "\n")
cat("BIC:", 2 * fit$value + length(mle) * log(n), "\n")


## Example 3: Comparing Optimization Methods

methods <- c("BFGS", "L-BFGS-B", "Nelder-Mead", "CG")
start_params <- c(2, 3, 1.5)

comparison <- data.frame(
  Method = character(),
  Alpha = numeric(),
  Beta = numeric(),
  Lambda = numeric(),
  NegLogLik = numeric(),
  Convergence = integer(),
  stringsAsFactors = FALSE
)

for (method in methods) {
  if (method %in% c("BFGS", "CG")) {
    fit_temp <- optim(
      par = start_params,
      fn = llekw,
      gr = grekw,
      data = data,
      method = method
    )
  } else if (method == "L-BFGS-B") {
    fit_temp <- optim(
      par = start_params,
      fn = llekw,
      gr = grekw,
      data = data,
      method = method,
      lower = c(0.01, 0.01, 0.01),
      upper = c(100, 100, 100)
    )
  } else {
    fit_temp <- optim(
      par = start_params,
      fn = llekw,
      data = data,
      method = method
    )
  }
```

```
    comparison <- rbind(comparison, data.frame(
      Method = method,
      Alpha = fit_temp$par[1],
      Beta = fit_temp$par[2],
      Lambda = fit_temp$par[3],
      NegLogLik = fit_temp$value,
      Convergence = fit_temp$convergence,
      stringsAsFactors = FALSE
    ))
  }

  print(comparison, digits = 4, row.names = FALSE)


  ## Example 4: Likelihood Ratio Test

  # Test H0: lambda = 2 vs H1: lambda free
  loglik_full <- -fit$value

  restricted_ll <- function(params_restricted, data, lambda_fixed) {
    llekw(par = c(params_restricted[1], params_restricted[2],
                  lambda_fixed), data = data)
  }

  fit_restricted <- optim(
    par = c(mle[1], mle[2]),
    fn = restricted_ll,
    data = data,
    lambda_fixed = 2,
    method = "BFGS"
  )

  loglik_restricted <- -fit_restricted$value
  lr_stat <- 2 * (loglik_full - loglik_restricted)
  p_value <- pchisq(lr_stat, df = 1, lower.tail = FALSE)

  cat("LR Statistic:", round(lr_stat, 4), "\n")
  cat("P-value:", format.pval(p_value, digits = 4), "\n")


  ## Example 5: Univariate Profile Likelihoods

  # Profile for alpha
  alpha_grid <- seq(mle[1] - 1, mle[1] + 1, length.out = 50)
  alpha_grid <- alpha_grid[alpha_grid > 0]
  profile_ll_alpha <- numeric(length(alpha_grid))

  for (i in seq_along(alpha_grid)) {
    profile_fit <- optim(
      par = mle[-1],
      fn = function(p) llekw(c(alpha_grid[i], p), data),
      method = "BFGS"
    )
    profile_ll_alpha[i] <- -profile_fit$value
  }

  # Profile for beta
  beta_grid <- seq(mle[2] - 1, mle[2] + 1, length.out = 50)
  beta_grid <- beta_grid[beta_grid > 0]
  profile_ll_beta <- numeric(length(beta_grid))

  for (i in seq_along(beta_grid)) {
    profile_fit <- optim(
      par = mle[-2],
      fn = function(p) llekw(c(p[1], beta_grid[i], p[2]), data),
      method = "BFGS"
    )
    profile_ll_beta[i] <- -profile_fit$value
  }

  # Profile for lambda
  lambda_grid <- seq(mle[3] - 1, mle[3] + 1, length.out = 50)
  lambda_grid <- lambda_grid[lambda_grid > 0]
  profile_ll_lambda <- numeric(length(lambda_grid))

  for (i in seq_along(lambda_grid)) {
    profile_fit <- optim(
      par = mle[-3],
      fn = function(p) llekw(c(p[1], p[2], lambda_grid[i]), data),
      method = "BFGS"
    )
    profile_ll_lambda[i] <- -profile_fit$value
  }

  # 95% confidence threshold
  chi_crit <- qchisq(0.95, df = 1)
  threshold <- max(profile_ll_alpha) - chi_crit / 2

  # Plot all profiles

  plot(alpha_grid, profile_ll_alpha, type = "l", lwd = 2, col = "#2E4057",
       xlab = expression(alpha), ylab = "Profile Log-Likelihood",
       main = expression(paste("Profile: ", alpha)), las = 1)
  abline(v = mle[1], col = "#8B0000", lty = 2, lwd = 2)
  abline(v = true_params[1], col = "#006400", lty = 2, lwd = 2)
  abline(h = threshold, col = "#808080", lty = 3, lwd = 1.5)
  legend("topright", legend = c("MLE", "True", "95% CI"),
         col = c("#8B0000", "#006400", "#808080"),
         lty = c(2, 2, 3), lwd = 2, bty = "n", cex = 0.8)
  grid(col = "gray90")

  plot(beta_grid, profile_ll_beta, type = "l", lwd = 2, col = "#2E4057",
       xlab = expression(beta), ylab = "Profile Log-Likelihood",
       main = expression(paste("Profile: ", beta)), las = 1)
  abline(v = mle[2], col = "#8B0000", lty = 2, lwd = 2)
  abline(v = true_params[2], col = "#006400", lty = 2, lwd = 2)
  abline(h = threshold, col = "#808080", lty = 3, lwd = 1.5)
  legend("topright", legend = c("MLE", "True", "95% CI"),
         col = c("#8B0000", "#006400", "#808080"),
         lty = c(2, 2, 3), lwd = 2, bty = "n", cex = 0.8)
  grid(col = "gray90")

  plot(lambda_grid, profile_ll_lambda, type = "l", lwd = 2, col = "#2E4057",
       xlab = expression(lambda), ylab = "Profile Log-Likelihood",
       main = expression(paste("Profile: ", lambda)), las = 1)
  abline(v = mle[3], col = "#8B0000", lty = 2, lwd = 2)
  abline(v = true_params[3], col = "#006400", lty = 2, lwd = 2)
  abline(h = threshold, col = "#808080", lty = 3, lwd = 1.5)
  legend("topright", legend = c("MLE", "True", "95% CI"),
         col = c("#8B0000", "#006400", "#808080"),
         lty = c(2, 2, 3), lwd = 2, bty = "n", cex = 0.8)
  grid(col = "gray90")


  ## Example 6: 2D Log-Likelihood Surface (Alpha vs Beta)

  # Create 2D grid
  alpha_2d <- seq(mle[1] - 0.8, mle[1] + 0.8, length.out = round(n/25))
  beta_2d <- seq(mle[2] - 0.8, mle[2] + 0.8, length.out = round(n/25))
  alpha_2d <- alpha_2d[alpha_2d > 0]
  beta_2d <- beta_2d[beta_2d > 0]

  # Compute log-likelihood surface
  ll_surface_ab <- matrix(NA, nrow = length(alpha_2d), ncol = length(beta_2d))

  for (i in seq_along(alpha_2d)) {
    for (j in seq_along(beta_2d)) {
      ll_surface_ab[i, j] <- -llekw(c(alpha_2d[i], beta_2d[j], mle[3]), data)
    }
  }

  # Confidence region levels
  max_ll_ab <- max(ll_surface_ab, na.rm = TRUE)
  levels_90_ab <- max_ll_ab - qchisq(0.90, df = 2) / 2
  levels_95_ab <- max_ll_ab - qchisq(0.95, df = 2) / 2
  levels_99_ab <- max_ll_ab - qchisq(0.99, df = 2) / 2

  # Plot contour
  contour(alpha_2d, beta_2d, ll_surface_ab,
          xlab = expression(alpha), ylab = expression(beta),
          main = "2D Log-Likelihood: Alpha vs Beta",
          levels = seq(min(ll_surface_ab, na.rm = TRUE), max_ll_ab, length.out = 20),
          col = "#2E4057", las = 1, lwd = 1)

  contour(alpha_2d, beta_2d, ll_surface_ab,
          levels = c(levels_90_ab, levels_95_ab, levels_99_ab),
          col = c("#FFA07A", "#FF6347", "#8B0000"),
          lwd = c(2, 2.5, 3), lty = c(3, 2, 1),
          add = TRUE, labcex = 0.8)
```

```
points(mle[1], mle[2], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[2], pch = 17, col = "#006400", cex = 1.5)

legend("topright",
       legend = c("MLE", "True", "90% CR", "95% CR", "99% CR"),
       col = c("#8B0000", "#006400", "#FFA07A", "#FF6347", "#8B0000"),
       pch = c(19, 17, NA, NA, NA),
       lty = c(NA, NA, 3, 2, 1),
       lwd = c(NA, NA, 2, 2.5, 3),
       bty = "n", cex = 0.8)
grid(col = "gray90")


## Example 7: 2D Log-Likelihood Surface (Alpha vs Lambda)

# Create 2D grid
alpha_2d_2 <- seq(mle[1] - 0.8, mle[1] + 0.8, length.out = round(n/25))
lambda_2d <- seq(mle[3] - 0.8, mle[3] + 0.8, length.out = round(n/25))
alpha_2d_2 <- alpha_2d_2[alpha_2d_2 > 0]
lambda_2d <- lambda_2d[lambda_2d > 0]

# Compute log-likelihood surface
ll_surface_al <- matrix(NA, nrow = length(alpha_2d_2), ncol = length(lambda_2d))

for (i in seq_along(alpha_2d_2)) {
  for (j in seq_along(lambda_2d)) {
    ll_surface_al[i, j] <- -llekw(c(alpha_2d_2[i], mle[2], lambda_2d[j]), data)
  }
}

# Confidence region levels
max_ll_al <- max(ll_surface_al, na.rm = TRUE)
levels_90_al <- max_ll_al - qchisq(0.90, df = 2) / 2
levels_95_al <- max_ll_al - qchisq(0.95, df = 2) / 2
levels_99_al <- max_ll_al - qchisq(0.99, df = 2) / 2

# Plot contour
contour(alpha_2d_2, lambda_2d, ll_surface_al,
        xlab = expression(alpha), ylab = expression(lambda),
        main = "2D Log-Likelihood: Alpha vs Lambda",
        levels = seq(min(ll_surface_al, na.rm = TRUE), max_ll_al, length.out = 20),
        col = "#2E4057", las = 1, lwd = 1)

contour(alpha_2d_2, lambda_2d, ll_surface_al,
        levels = c(levels_90_al, levels_95_al, levels_99_al),
        col = c("#FFA07A", "#FF6347", "#8B0000"),
        lwd = c(2, 2.5, 3), lty = c(3, 2, 1),
        add = TRUE, labcex = 0.8)

points(mle[1], mle[3], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[3], pch = 17, col = "#006400", cex = 1.5)
```

```
legend("topright",
       legend = c("MLE", "True", "90% CR", "95% CR", "99% CR"),
       col = c("#8B0000", "#006400", "#FFA07A", "#FF6347", "#8B0000"),
       pch = c(19, 17, NA, NA, NA),
       lty = c(NA, NA, 3, 2, 1),
       lwd = c(NA, NA, 2, 2.5, 3),
       bty = "n", cex = 0.8)
grid(col = "gray90")


## Example 8: 2D Log-Likelihood Surface (Beta vs Lambda)

# Create 2D grid
beta_2d_2 <- seq(mle[2] - 0.8, mle[2] + 0.8, length.out = round(n/25))
lambda_2d_2 <- seq(mle[3] - 0.8, mle[3] + 0.8, length.out = round(n/25))
beta_2d_2 <- beta_2d_2[beta_2d_2 > 0]
lambda_2d_2 <- lambda_2d_2[lambda_2d_2 > 0]

# Compute log-likelihood surface
ll_surface_bl <- matrix(NA, nrow = length(beta_2d_2), ncol = length(lambda_2d_2))

for (i in seq_along(beta_2d_2)) {
  for (j in seq_along(lambda_2d_2)) {
    ll_surface_bl[i, j] <- -llekw(c(mle[1], beta_2d_2[i], lambda_2d_2[j]), data)
  }
}

# Confidence region levels
max_ll_bl <- max(ll_surface_bl, na.rm = TRUE)
levels_90_bl <- max_ll_bl - qchisq(0.90, df = 2) / 2
levels_95_bl <- max_ll_bl - qchisq(0.95, df = 2) / 2
levels_99_bl <- max_ll_bl - qchisq(0.99, df = 2) / 2

# Plot contour
contour(beta_2d_2, lambda_2d_2, ll_surface_bl,
        xlab = expression(beta), ylab = expression(lambda),
        main = "2D Log-Likelihood: Beta vs Lambda",
        levels = seq(min(ll_surface_bl, na.rm = TRUE), max_ll_bl, length.out = 20),
        col = "#2E4057", las = 1, lwd = 1)

contour(beta_2d_2, lambda_2d_2, ll_surface_bl,
        levels = c(levels_90_bl, levels_95_bl, levels_99_bl),
        col = c("#FFA07A", "#FF6347", "#8B0000"),
        lwd = c(2, 2.5, 3), lty = c(3, 2, 1),
        add = TRUE, labcex = 0.8)

points(mle[2], mle[3], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[2], true_params[3], pch = 17, col = "#006400", cex = 1.5)

legend("topright",
       legend = c("MLE", "True", "90% CR", "95% CR", "99% CR"),
       col = c("#8B0000", "#006400", "#FFA07A", "#FF6347", "#8B0000"),
       pch = c(19, 17, NA, NA, NA),
       lty = c(NA, NA, 3, 2, 1),
       lwd = c(NA, NA, 2, 2.5, 3),
       bty = "n", cex = 0.8)
grid(col = "gray90")
```

`llgkw` *Negative Log-Likelihood for the Generalized Kumaraswamy Distribution*

Description

Computes the negative log-likelihood function for the five-parameter Generalized Kumaraswamy (GKw) distribution given a vector of observations. This function is designed for use in optimization routines (e.g., maximum likelihood estimation).

Usage

```
llgkw(par, data)
```

Arguments

`par`	A numeric vector of length 5 containing the distribution parameters in the order: `alpha` ($\alpha > 0$), `beta` ($\beta > 0$), `gamma` ($\gamma > 0$), `delta` ($\delta \geq 0$), `lambda` ($\lambda > 0$).
`data`	A numeric vector of observations. All values must be strictly between 0 and 1 (exclusive).

Details

The probability density function (PDF) of the GKw distribution is given in dgkw. The log-likelihood function $\ell(\theta)$ for a sample $\mathbf{x} = (x_1, \dots, x_n)$ is:

$$\ell(\theta|\mathbf{x}) = n\ln(\lambda\alpha\beta) - n\ln B(\gamma, \delta+1) + \sum_{i=1}^{n} [(\alpha-1)\ln(x_i) + (\beta-1)\ln(v_i) + (\gamma\lambda-1)\ln(w_i) + \delta\ln(z_i)]$$

where $\theta = (\alpha, \beta, \gamma, \delta, \lambda)$, $B(a, b)$ is the Beta function (beta), and:

- $v_i = 1 - x_i^\alpha$
- $w_i = 1 - v_i^\beta = 1 - (1 - x_i^\alpha)^\beta$
- $z_i = 1 - w_i^\lambda = 1 - [1 - (1 - x_i^\alpha)^\beta]^\lambda$

This function computes $-\ell(\theta|\mathbf{x})$.

Numerical stability is prioritized using:

- lbeta function for the log-Beta term.
- Log-transformations of intermediate terms (v_i, w_i, z_i) and use of log1p where appropriate to handle values close to 0 or 1 accurately.
- Checks for invalid parameters and data.

Value

Returns a single double value representing the negative log-likelihood ($-\ell(\theta|\mathbf{x})$). Returns a large positive value (e.g., Inf) if any parameter values in par are invalid according to their constraints, or if any value in data is not in the interval (0, 1).

Author(s)

Lopes, J. E.

References

Cordeiro, G. M., & de Castro, M. (2011). A new family of generalized distributions. *Journal of Statistical Computation and Simulation*

Kumaraswamy, P. (1980). A generalized probability density function for double-bounded random processes. *Journal of Hydrology*, *46*(1-2), 79-88.

See Also

dgkw, pgkw, qgkw, rgkw, grgkw, hsgkw (gradient and Hessian functions, if available), optim, lbeta, log1p

Examples

```
## Example 1: Basic Log-Likelihood Evaluation

# Generate sample data
set.seed(123)
n <- 1000
true_params <- c(alpha = 2.0, beta = 3.0, gamma = 1.5, delta = 2.0, lambda = 1.8)
data <- rgkw(n, alpha = true_params[1], beta = true_params[2],
             gamma = true_params[3], delta = true_params[4],
             lambda = true_params[5])

# Evaluate negative log-likelihood at true parameters
nll_true <- llgkw(par = true_params, data = data)
cat("Negative log-likelihood at true parameters:", nll_true, "\n")

# Evaluate at different parameter values
test_params <- rbind(
  c(1.5, 2.5, 1.2, 1.5, 1.5),
  c(2.0, 3.0, 1.5, 2.0, 1.8),
  c(2.5, 3.5, 1.8, 2.5, 2.0)
)

nll_values <- apply(test_params, 1, function(p) llgkw(p, data))
results <- data.frame(
  Alpha = test_params[, 1],
  Beta = test_params[, 2],
  Gamma = test_params[, 3],
  Delta = test_params[, 4],
  Lambda = test_params[, 5],
  NegLogLik = nll_values
)
print(results, digits = 4)


## Example 2: Maximum Likelihood Estimation

# Optimization using BFGS with analytical gradient
fit <- optim(
  par = c(1.5, 2.5, 1.2, 1.5, 1.5),
  fn = llgkw,
  gr = grgkw,
  data = data,
  method = "BFGS",
  hessian = TRUE,
  control = list(maxit = 1000)
)

mle <- fit$par
names(mle) <- c("alpha", "beta", "gamma", "delta", "lambda")
se <- sqrt(diag(solve(fit$hessian)))

results <- data.frame(
  Parameter = c("alpha", "beta", "gamma", "delta", "lambda"),
  True = true_params,
  MLE = mle,
  SE = se,
  CI_Lower = mle - 1.96 * se,
  CI_Upper = mle + 1.96 * se
)
print(results, digits = 4)

cat("\nNegative log-likelihood at MLE:", fit$value, "\n")
cat("AIC:", 2 * fit$value + 2 * length(mle), "\n")
cat("BIC:", 2 * fit$value + length(mle) * log(n), "\n")


## Example 3: Comparing Optimization Methods

methods <- c("BFGS", "Nelder-Mead")
start_params <- c(1.5, 2.5, 1.2, 1.5, 1.5)

comparison <- data.frame(
  Method = character(),
  Alpha = numeric(),
  Beta = numeric(),
  Gamma = numeric(),
  Delta = numeric(),
  Lambda = numeric(),
  NegLogLik = numeric(),
  Convergence = integer(),
  stringsAsFactors = FALSE
)
```

```
for (method in methods) {
  if (method == "BFGS") {
    fit_temp <- optim(
      par = start_params,
      fn = llgkw,
      gr = grgkw,
      data = data,
      method = method,
      control = list(maxit = 1000)
    )
  } else if (method == "L-BFGS-B") {
    fit_temp <- optim(
      par = start_params,
      fn = llgkw,
      gr = grgkw,
      data = data,
      method = method,
      lower = rep(0.001, 5),
      upper = rep(20, 5),
      control = list(maxit = 1000)
    )
  } else {
    fit_temp <- optim(
      par = start_params,
      fn = llgkw,
      data = data,
      method = method,
      control = list(maxit = 1000)
    )
  }

  comparison <- rbind(comparison, data.frame(
    Method = method,
    Alpha = fit_temp$par[1],
    Beta = fit_temp$par[2],
    Gamma = fit_temp$par[3],
    Delta = fit_temp$par[4],
    Lambda = fit_temp$par[5],
    NegLogLik = fit_temp$value,
    Convergence = fit_temp$convergence,
    stringsAsFactors = FALSE
  ))
}

print(comparison, digits = 4, row.names = FALSE)


## Example 4: Likelihood Ratio Test

# Test H0: gamma = 1.5 vs H1: gamma free
loglik_full <- -fit$value
```

```
restricted_ll <- function(params_restricted, data, gamma_fixed) {
  llgkw(par = c(params_restricted[1], params_restricted[2],
                gamma_fixed, params_restricted[3], params_restricted[4]),
        data = data)
}

fit_restricted <- optim(
  par = c(mle[1], mle[2], mle[4], mle[5]),
  fn = restricted_ll,
  data = data,
  gamma_fixed = 1.5,
  method = "Nelder-Mead",
  control = list(maxit = 1000)
)

loglik_restricted <- -fit_restricted$value
lr_stat <- 2 * (loglik_full - loglik_restricted)
p_value <- pchisq(lr_stat, df = 1, lower.tail = FALSE)

cat("LR Statistic:", round(lr_stat, 4), "\n")
cat("P-value:", format.pval(p_value, digits = 4), "\n")


## Example 5: Univariate Profile Likelihoods

# Profile for alpha
xd <- 1
alpha_grid <- seq(mle[1] - xd, mle[1] + xd, length.out = 35)
alpha_grid <- alpha_grid[alpha_grid > 0]
profile_ll_alpha <- numeric(length(alpha_grid))

for (i in seq_along(alpha_grid)) {
  profile_fit <- optim(
    par = mle[-1],
    fn = function(p) llgkw(c(alpha_grid[i], p), data),
    method = "Nelder-Mead",
    control = list(maxit = 500)
  )
  profile_ll_alpha[i] <- -profile_fit$value
}

# Profile for beta
beta_grid <- seq(mle[2] - xd, mle[2] + xd, length.out = 35)
beta_grid <- beta_grid[beta_grid > 0]
profile_ll_beta <- numeric(length(beta_grid))

for (i in seq_along(beta_grid)) {
  profile_fit <- optim(
    par = mle[-2],
    fn = function(p) llgkw(c(p[1], beta_grid[i], p[2], p[3], p[4]), data),
    method = "Nelder-Mead",
    control = list(maxit = 500)
  )
```

```
    profile_ll_beta[i] <- -profile_fit$value
  }

  # Profile for gamma
  gamma_grid <- seq(mle[3] - xd, mle[3] + xd, length.out = 35)
  gamma_grid <- gamma_grid[gamma_grid > 0]
  profile_ll_gamma <- numeric(length(gamma_grid))

  for (i in seq_along(gamma_grid)) {
    profile_fit <- optim(
      par = mle[-3],
      fn = function(p) llgkw(c(p[1], p[2], gamma_grid[i], p[3], p[4]), data),
      method = "Nelder-Mead",
      control = list(maxit = 500)
    )
    profile_ll_gamma[i] <- -profile_fit$value
  }

  # Profile for delta
  delta_grid <- seq(mle[4] - xd, mle[4] + xd, length.out = 35)
  delta_grid <- delta_grid[delta_grid > 0]
  profile_ll_delta <- numeric(length(delta_grid))

  for (i in seq_along(delta_grid)) {
    profile_fit <- optim(
      par = mle[-4],
      fn = function(p) llgkw(c(p[1], p[2], p[3], delta_grid[i], p[4]), data),
      method = "Nelder-Mead",
      control = list(maxit = 500)
    )
    profile_ll_delta[i] <- -profile_fit$value
  }

  # Profile for lambda
  lambda_grid <- seq(mle[5] - xd, mle[5] + xd, length.out = 35)
  lambda_grid <- lambda_grid[lambda_grid > 0]
  profile_ll_lambda <- numeric(length(lambda_grid))

  for (i in seq_along(lambda_grid)) {
    profile_fit <- optim(
      par = mle[-5],
      fn = function(p) llgkw(c(p[1], p[2], p[3], p[4], lambda_grid[i]), data),
      method = "Nelder-Mead",
      control = list(maxit = 500)
    )
    profile_ll_lambda[i] <- -profile_fit$value
  }

  # 95% confidence threshold
  chi_crit <- qchisq(0.95, df = 1)
  threshold <- max(profile_ll_alpha) - chi_crit / 2

  # Plot all profiles
```

```
plot(alpha_grid, profile_ll_alpha, type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(alpha), ylab = "Profile Log-Likelihood",
     main = expression(paste("Profile: ", alpha)), las = 1)
abline(v = mle[1], col = "#8B0000", lty = 2, lwd = 2)
abline(v = true_params[1], col = "#006400", lty = 2, lwd = 2)
abline(h = threshold, col = "#808080", lty = 3, lwd = 1.5)
legend("topright", legend = c("MLE", "True", "95% CI"),
       col = c("#8B0000", "#006400", "#808080"),
       lty = c(2, 2, 3), lwd = 2, bty = "n", cex = 0.6)
grid(col = "gray90")

plot(beta_grid, profile_ll_beta, type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(beta), ylab = "Profile Log-Likelihood",
     main = expression(paste("Profile: ", beta)), las = 1)
abline(v = mle[2], col = "#8B0000", lty = 2, lwd = 2)
abline(v = true_params[2], col = "#006400", lty = 2, lwd = 2)
abline(h = threshold, col = "#808080", lty = 3, lwd = 1.5)
legend("topright", legend = c("MLE", "True", "95% CI"),
       col = c("#8B0000", "#006400", "#808080"),
       lty = c(2, 2, 3), lwd = 2, bty = "n", cex = 0.6)
grid(col = "gray90")

plot(gamma_grid, profile_ll_gamma, type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(gamma), ylab = "Profile Log-Likelihood",
     main = expression(paste("Profile: ", gamma)), las = 1)
abline(v = mle[3], col = "#8B0000", lty = 2, lwd = 2)
abline(v = true_params[3], col = "#006400", lty = 2, lwd = 2)
abline(h = threshold, col = "#808080", lty = 3, lwd = 1.5)
legend("topright", legend = c("MLE", "True", "95% CI"),
       col = c("#8B0000", "#006400", "#808080"),
       lty = c(2, 2, 3), lwd = 2, bty = "n", cex = 0.6)
grid(col = "gray90")

plot(delta_grid, profile_ll_delta, type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(delta), ylab = "Profile Log-Likelihood",
     main = expression(paste("Profile: ", delta)), las = 1)
abline(v = mle[4], col = "#8B0000", lty = 2, lwd = 2)
abline(v = true_params[4], col = "#006400", lty = 2, lwd = 2)
abline(h = threshold, col = "#808080", lty = 3, lwd = 1.5)
legend("topright", legend = c("MLE", "True", "95% CI"),
       col = c("#8B0000", "#006400", "#808080"),
       lty = c(2, 2, 3), lwd = 2, bty = "n", cex = 0.6)
grid(col = "gray90")

plot(lambda_grid, profile_ll_lambda, type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(lambda), ylab = "Profile Log-Likelihood",
     main = expression(paste("Profile: ", lambda)), las = 1)
abline(v = mle[5], col = "#8B0000", lty = 2, lwd = 2)
abline(v = true_params[5], col = "#006400", lty = 2, lwd = 2)
abline(h = threshold, col = "#808080", lty = 3, lwd = 1.5)
legend("topright", legend = c("MLE", "True", "95% CI"),
       col = c("#8B0000", "#006400", "#808080"),
       lty = c(2, 2, 3), lwd = 2, bty = "n", cex = 0.6)
grid(col = "gray90")


## Example 6: 2D Log-Likelihood Surface (Alpha vs Beta)
# Plot all profiles

# Create 2D grid
alpha_2d <- seq(mle[1] - xd, mle[1] + xd, length.out = round(n/4))
beta_2d <- seq(mle[2] - xd, mle[2] + xd, length.out = round(n/4))
alpha_2d <- alpha_2d[alpha_2d > 0]
beta_2d <- beta_2d[beta_2d > 0]

# Compute log-likelihood surface
ll_surface_ab <- matrix(NA, nrow = length(alpha_2d), ncol = length(beta_2d))

for (i in seq_along(alpha_2d)) {
  for (j in seq_along(beta_2d)) {
    ll_surface_ab[i, j] <- llgkw(c(alpha_2d[i], beta_2d[j],
                                   mle[3], mle[4], mle[5]), data)
  }
}

# Confidence region levels
max_ll_ab <- max(ll_surface_ab, na.rm = TRUE)
levels_90_ab <- max_ll_ab - qchisq(0.90, df = 2) / 2
levels_95_ab <- max_ll_ab - qchisq(0.95, df = 2) / 2
levels_99_ab <- max_ll_ab - qchisq(0.99, df = 2) / 2

# Plot contour
contour(alpha_2d, beta_2d, ll_surface_ab,
        xlab = expression(alpha), ylab = expression(beta),
        main = "2D Log-Likelihood: Alpha vs Beta",
        levels = seq(min(ll_surface_ab, na.rm = TRUE), max_ll_ab, length.out = 20),
        col = "#2E4057", las = 1, lwd = 1)

contour(alpha_2d, beta_2d, ll_surface_ab,
        levels = c(levels_90_ab, levels_95_ab, levels_99_ab),
        col = c("#FFA07A", "#FF6347", "#8B0000"),
        lwd = c(2, 2.5, 3), lty = c(3, 2, 1),
        add = TRUE, labcex = 0.8)

points(mle[1], mle[2], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[2], pch = 17, col = "#006400", cex = 1.5)

legend("topright",
       legend = c("MLE", "True", "90% CR", "95% CR", "99% CR"),
       col = c("#8B0000", "#006400", "#FFA07A", "#FF6347", "#8B0000"),
       pch = c(19, 17, NA, NA, NA),
       lty = c(NA, NA, 3, 2, 1),
       lwd = c(NA, NA, 2, 2.5, 3),
       bty = "n", cex = 0.8)
grid(col = "gray90")
```

```
## Example 7: 2D Log-Likelihood Surface (Gamma vs Delta)

# Create 2D grid
gamma_2d <- seq(mle[3] - xd, mle[3] + xd, length.out = round(n/4))
delta_2d <- seq(mle[4] - xd, mle[4] + xd, length.out = round(n/4))
gamma_2d <- gamma_2d[gamma_2d > 0]
delta_2d <- delta_2d[delta_2d > 0]

# Compute log-likelihood surface
ll_surface_gd <- matrix(NA, nrow = length(gamma_2d), ncol = length(delta_2d))

for (i in seq_along(gamma_2d)) {
  for (j in seq_along(delta_2d)) {
    ll_surface_gd[i, j] <- -llgkw(c(mle[1], mle[2], gamma_2d[i],
                                     delta_2d[j], mle[5]), data)
  }
}

# Confidence region levels
max_ll_gd <- max(ll_surface_gd, na.rm = TRUE)
levels_90_gd <- max_ll_gd - qchisq(0.90, df = 2) / 2
levels_95_gd <- max_ll_gd - qchisq(0.95, df = 2) / 2
levels_99_gd <- max_ll_gd - qchisq(0.99, df = 2) / 2

# Plot contour
contour(gamma_2d, delta_2d, ll_surface_gd,
        xlab = expression(gamma), ylab = expression(delta),
        main = "2D Log-Likelihood: Gamma vs Delta",
        levels = seq(min(ll_surface_gd, na.rm = TRUE), max_ll_gd, length.out = 20),
        col = "#2E4057", las = 1, lwd = 1)

contour(gamma_2d, delta_2d, ll_surface_gd,
        levels = c(levels_90_gd, levels_95_gd, levels_99_gd),
        col = c("#FFA07A", "#FF6347", "#8B0000"),
        lwd = c(2, 2.5, 3), lty = c(3, 2, 1),
        add = TRUE, labcex = 0.8)

points(mle[3], mle[4], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[3], true_params[4], pch = 17, col = "#006400", cex = 1.5)

legend("topright",
       legend = c("MLE", "True", "90% CR", "95% CR", "99% CR"),
       col = c("#8B0000", "#006400", "#FFA07A", "#FF6347", "#8B0000"),
       pch = c(19, 17, NA, NA, NA),
       lty = c(NA, NA, 3, 2, 1),
       lwd = c(NA, NA, 2, 2.5, 3),
       bty = "n", cex = 0.8)
grid(col = "gray90")


## Example 8: 2D Log-Likelihood Surface (Delta vs Lambda)
```

```
# Create 2D grid
delta_2d_2 <- seq(mle[4] - xd, mle[4] + xd, length.out = round(n/30))
lambda_2d <- seq(mle[5] - xd, mle[5] + xd, length.out = round(n/30))
delta_2d_2 <- delta_2d_2[delta_2d_2 > 0]
lambda_2d <- lambda_2d[lambda_2d > 0]

# Compute log-likelihood surface
ll_surface_dl <- matrix(NA, nrow = length(delta_2d_2), ncol = length(lambda_2d))

for (i in seq_along(delta_2d_2)) {
  for (j in seq_along(lambda_2d)) {
    ll_surface_dl[i, j] <- -llgkw(c(mle[1], mle[2], mle[3],
                                    delta_2d_2[i], lambda_2d[j]), data)
  }
}

# Confidence region levels
max_ll_dl <- max(ll_surface_dl, na.rm = TRUE)
levels_90_dl <- max_ll_dl - qchisq(0.90, df = 2) / 2
levels_95_dl <- max_ll_dl - qchisq(0.95, df = 2) / 2
levels_99_dl <- max_ll_dl - qchisq(0.99, df = 2) / 2

# Plot contour
contour(delta_2d_2, lambda_2d, ll_surface_dl,
        xlab = expression(delta), ylab = expression(lambda),
        main = "2D Log-Likelihood: Delta vs Lambda",
        levels = seq(min(ll_surface_dl, na.rm = TRUE), max_ll_dl, length.out = 20),
        col = "#2E4057", las = 1, lwd = 1)

contour(delta_2d_2, lambda_2d, ll_surface_dl,
        levels = c(levels_90_dl, levels_95_dl, levels_99_dl),
        col = c("#FFA07A", "#FF6347", "#8B0000"),
        lwd = c(2, 2.5, 3), lty = c(3, 2, 1),
        add = TRUE, labcex = 0.8)

points(mle[4], mle[5], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[4], true_params[5], pch = 17, col = "#006400", cex = 1.5)

legend("topright",
       legend = c("MLE", "True", "90% CR", "95% CR", "99% CR"),
       col = c("#8B0000", "#006400", "#FFA07A", "#FF6347", "#8B0000"),
       pch = c(19, 17, NA, NA, NA),
       lty = c(NA, NA, 3, 2, 1),
       lwd = c(NA, NA, 2, 2.5, 3),
       bty = "n", cex = 0.8)
grid(col = "gray90")
```

llkkw — *Negative Log-Likelihood for the kkw Distribution*

Description

Computes the negative log-likelihood function for the Kumaraswamy-Kumaraswamy (kkw) distribution with parameters `alpha` (α), `beta` (β), `delta` (δ), and `lambda` (λ), given a vector of observations. This distribution is a special case of the Generalized Kumaraswamy (GKw) distribution where $\gamma = 1$.

Usage

```
llkkw(par, data)
```

Arguments

`par`	A numeric vector of length 4 containing the distribution parameters in the order: `alpha` ($\alpha > 0$), `beta` ($\beta > 0$), `delta` ($\delta \geq 0$), `lambda` ($\lambda > 0$).
`data`	A numeric vector of observations. All values must be strictly between 0 and 1 (exclusive).

Details

The kkw distribution is the GKw distribution (`dgkw`) with $\gamma = 1$. Its probability density function (PDF) is:

$$f(x|\theta) = (\delta+1)\lambda\alpha\beta x^{\alpha-1}(1-x^\alpha)^{\beta-1}\bigl[1-(1-x^\alpha)^\beta\bigr]^{\lambda-1}\bigl\{1-\bigl[1-(1-x^\alpha)^\beta\bigr]^\lambda\bigr\}^\delta$$

for $0 < x < 1$ and $\theta = (\alpha, \beta, \delta, \lambda)$. The log-likelihood function $\ell(\theta|\mathbf{x})$ for a sample $\mathbf{x} = (x_1, \dots, x_n)$ is $\sum_{i=1}^n \ln f(x_i|\theta)$:

$$\ell(\theta|\mathbf{x}) = n[\ln(\delta+1)+\ln(\lambda)+\ln(\alpha)+\ln(\beta)]+\sum_{i=1}^{n}[(\alpha-1)\ln(x_i)+(\beta-1)\ln(v_i)+(\lambda-1)\ln(w_i)+\delta\ln(z_i)]$$

where:

- $v_i = 1 - x_i^\alpha$
- $w_i = 1 - v_i^\beta = 1 - (1 - x_i^\alpha)^\beta$
- $z_i = 1 - w_i^\lambda = 1 - [1 - (1 - x_i^\alpha)^\beta]^\lambda$

This function computes and returns the *negative* log-likelihood, $-\ell(\theta|\mathbf{x})$, suitable for minimization using optimization routines like `optim`. Numerical stability is maintained similarly to `llgkw`.

Value

Returns a single `double` value representing the negative log-likelihood ($-\ell(\theta|\mathbf{x})$). Returns `Inf` if any parameter values in `par` are invalid according to their constraints, or if any value in `data` is not in the interval (0, 1).

Author(s)

Lopes, J. E.

References

Cordeiro, G. M., & de Castro, M. (2011). A new family of generalized distributions. *Journal of Statistical Computation and Simulation*,

Kumaraswamy, P. (1980). A generalized probability density function for double-bounded random processes. *Journal of Hydrology*, *46*(1-2), 79-88.

See Also

`llgkw` (parent distribution negative log-likelihood), `dkkw`, `pkkw`, `qkkw`, `rkkw`, `grkkw` (gradient, if available), `hskkw` (Hessian, if available), `optim`

Examples

```
## Example 1: Basic Log-Likelihood Evaluation

# Generate sample data
set.seed(123)
n <- 1000
true_params <- c(alpha = 2.0, beta = 3.0, delta = 1.5, lambda = 2.0)
data <- rkkw(n, alpha = true_params[1], beta = true_params[2],
             delta = true_params[3], lambda = true_params[4])

# Evaluate negative log-likelihood at true parameters
nll_true <- llkkw(par = true_params, data = data)
cat("Negative log-likelihood at true parameters:", nll_true, "\n")

# Evaluate at different parameter values
test_params <- rbind(
  c(1.5, 2.5, 1.0, 1.5),
  c(2.0, 3.0, 1.5, 2.0),
  c(2.5, 3.5, 2.0, 2.5)
)

nll_values <- apply(test_params, 1, function(p) llkkw(p, data))
results <- data.frame(
  Alpha = test_params[, 1],
  Beta = test_params[, 2],
  Delta = test_params[, 3],
  Lambda = test_params[, 4],
  NegLogLik = nll_values
)
print(results, digits = 4)


## Example 2: Maximum Likelihood Estimation

# Optimization using BFGS with analytical gradient
```

```
fit <- optim(
  par = c(1.5, 2.5, 1.0, 1.5),
  fn = llkkw,
  gr = grkkw,
  data = data,
  method = "BFGS",
  hessian = TRUE
)

mle <- fit$par
names(mle) <- c("alpha", "beta", "delta", "lambda")
se <- sqrt(diag(solve(fit$hessian)))

results <- data.frame(
  Parameter = c("alpha", "beta", "delta", "lambda"),
  True = true_params,
  MLE = mle,
  SE = se,
  CI_Lower = mle - 1.96 * se,
  CI_Upper = mle + 1.96 * se
)
print(results, digits = 4)

cat("\nNegative log-likelihood at MLE:", fit$value, "\n")
cat("AIC:", 2 * fit$value + 2 * length(mle), "\n")
cat("BIC:", 2 * fit$value + length(mle) * log(n), "\n")


## Example 3: Comparing Optimization Methods

methods <- c("BFGS", "L-BFGS-B", "Nelder-Mead", "CG")
start_params <- c(1.5, 2.5, 1.0, 1.5)

comparison <- data.frame(
  Method = character(),
  Alpha = numeric(),
  Beta = numeric(),
  Delta = numeric(),
  Lambda = numeric(),
  NegLogLik = numeric(),
  Convergence = integer(),
  stringsAsFactors = FALSE
)

for (method in methods) {
  if (method %in% c("BFGS", "CG")) {
    fit_temp <- optim(
      par = start_params,
      fn = llkkw,
      gr = grkkw,
      data = data,
      method = method
    )
  } else if (method == "L-BFGS-B") {
    fit_temp <- optim(
      par = start_params,
      fn = llkkw,
      gr = grkkw,
      data = data,
      method = method,
      lower = c(0.01, 0.01, 0.01, 0.01),
      upper = c(100, 100, 100, 100)
    )
  } else {
    fit_temp <- optim(
      par = start_params,
      fn = llkkw,
      data = data,
      method = method
    )
  }

  comparison <- rbind(comparison, data.frame(
    Method = method,
    Alpha = fit_temp$par[1],
    Beta = fit_temp$par[2],
    Delta = fit_temp$par[3],
    Lambda = fit_temp$par[4],
    NegLogLik = fit_temp$value,
    Convergence = fit_temp$convergence,
    stringsAsFactors = FALSE
  ))
}

print(comparison, digits = 4, row.names = FALSE)


## Example 4: Likelihood Ratio Test

# Test H0: delta = 1.5 vs H1: delta free
loglik_full <- -fit$value

restricted_ll <- function(params_restricted, data, delta_fixed) {
  llkkw(par = c(params_restricted[1], params_restricted[2],
                delta_fixed, params_restricted[3]), data = data)
}

fit_restricted <- optim(
  par = c(mle[1], mle[2], mle[4]),
  fn = restricted_ll,
  data = data,
  delta_fixed = 1.5,
  method = "BFGS"
)

loglik_restricted <- -fit_restricted$value
```

```
lr_stat <- 2 * (loglik_full - loglik_restricted)
p_value <- pchisq(lr_stat, df = 1, lower.tail = FALSE)

cat("LR Statistic:", round(lr_stat, 4), "\n")
cat("P-value:", format.pval(p_value, digits = 4), "\n")


## Example 5: Univariate Profile Likelihoods

# Profile for alpha
alpha_grid <- seq(mle[1] - 1, mle[1] + 1, length.out = 40)
alpha_grid <- alpha_grid[alpha_grid > 0]
profile_ll_alpha <- numeric(length(alpha_grid))

for (i in seq_along(alpha_grid)) {
  profile_fit <- optim(
    par = mle[-1],
    fn = function(p) llkkw(c(alpha_grid[i], p), data),
    method = "Nelder-Mead"
  )
  profile_ll_alpha[i] <- -profile_fit$value
}

# Profile for beta
beta_grid <- seq(mle[2] - 1, mle[2] + 1, length.out = 40)
beta_grid <- beta_grid[beta_grid > 0]
profile_ll_beta <- numeric(length(beta_grid))

for (i in seq_along(beta_grid)) {
  profile_fit <- optim(
    par = mle[-2],
    fn = function(p) llkkw(c(p[1], beta_grid[i], p[2], p[3]), data),
    method = "Nelder-Mead"
  )
  profile_ll_beta[i] <- -profile_fit$value
}

# Profile for delta
delta_grid <- seq(mle[3] - 0.8, mle[3] + 0.8, length.out = 40)
delta_grid <- delta_grid[delta_grid > 0]
profile_ll_delta <- numeric(length(delta_grid))

for (i in seq_along(delta_grid)) {
  profile_fit <- optim(
    par = mle[-3],
    fn = function(p) llkkw(c(p[1], p[2], delta_grid[i], p[3]), data),
    method = "Nelder-Mead"
  )
  profile_ll_delta[i] <- -profile_fit$value
}

# Profile for lambda
lambda_grid <- seq(mle[4] - 1, mle[4] + 1, length.out = 40)
lambda_grid <- lambda_grid[lambda_grid > 0]
profile_ll_lambda <- numeric(length(lambda_grid))

for (i in seq_along(lambda_grid)) {
  profile_fit <- optim(
    par = mle[-4],
    fn = function(p) llkkw(c(p[1], p[2], p[3], lambda_grid[i]), data),
    method = "Nelder-Mead"
  )
  profile_ll_lambda[i] <- -profile_fit$value
}

# 95% confidence threshold
chi_crit <- qchisq(0.95, df = 1)
threshold <- max(profile_ll_alpha) - chi_crit / 2

# Plot all profiles

plot(alpha_grid, profile_ll_alpha, type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(alpha), ylab = "Profile Log-Likelihood",
     main = expression(paste("Profile: ", alpha)), las = 1)
abline(v = mle[1], col = "#8B0000", lty = 2, lwd = 2)
abline(v = true_params[1], col = "#006400", lty = 2, lwd = 2)
abline(h = threshold, col = "#808080", lty = 3, lwd = 1.5)
legend("topright", legend = c("MLE", "True", "95% CI"),
       col = c("#8B0000", "#006400", "#808080"),
       lty = c(2, 2, 3), lwd = 2, bty = "n", cex = 0.7)
grid(col = "gray90")

plot(beta_grid, profile_ll_beta, type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(beta), ylab = "Profile Log-Likelihood",
     main = expression(paste("Profile: ", beta)), las = 1)
abline(v = mle[2], col = "#8B0000", lty = 2, lwd = 2)
abline(v = true_params[2], col = "#006400", lty = 2, lwd = 2)
abline(h = threshold, col = "#808080", lty = 3, lwd = 1.5)
legend("topright", legend = c("MLE", "True", "95% CI"),
       col = c("#8B0000", "#006400", "#808080"),
       lty = c(2, 2, 3), lwd = 2, bty = "n", cex = 0.7)
grid(col = "gray90")

plot(delta_grid, profile_ll_delta, type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(delta), ylab = "Profile Log-Likelihood",
     main = expression(paste("Profile: ", delta)), las = 1)
abline(v = mle[3], col = "#8B0000", lty = 2, lwd = 2)
abline(v = true_params[3], col = "#006400", lty = 2, lwd = 2)
abline(h = threshold, col = "#808080", lty = 3, lwd = 1.5)
legend("topright", legend = c("MLE", "True", "95% CI"),
       col = c("#8B0000", "#006400", "#808080"),
       lty = c(2, 2, 3), lwd = 2, bty = "n", cex = 0.7)
grid(col = "gray90")

plot(lambda_grid, profile_ll_lambda, type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(lambda), ylab = "Profile Log-Likelihood",
     main = expression(paste("Profile: ", lambda)), las = 1)
abline(v = mle[4], col = "#8B0000", lty = 2, lwd = 2)
abline(v = true_params[4], col = "#006400", lty = 2, lwd = 2)
abline(h = threshold, col = "#808080", lty = 3, lwd = 1.5)
legend("topright", legend = c("MLE", "True", "95% CI"),
       col = c("#8B0000", "#006400", "#808080"),
       lty = c(2, 2, 3), lwd = 2, bty = "n", cex = 0.7)
grid(col = "gray90")

## Example 6: 2D Log-Likelihood Surface (Alpha vs Beta)

# Create 2D grid
alpha_2d <- seq(mle[1] - 0.8, mle[1] + 0.8, length.out = round(n/25))
beta_2d <- seq(mle[2] - 0.8, mle[2] + 0.8, length.out = round(n/25))
alpha_2d <- alpha_2d[alpha_2d > 0]
beta_2d <- beta_2d[beta_2d > 0]

# Compute log-likelihood surface
ll_surface <- matrix(NA, nrow = length(alpha_2d), ncol = length(beta_2d))

for (i in seq_along(alpha_2d)) {
  for (j in seq_along(beta_2d)) {
    ll_surface[i, j] <- -llkkw(c(alpha_2d[i], beta_2d[j], mle[3], mle[4]), data)
  }
}

# Confidence region levels
max_ll <- max(ll_surface, na.rm = TRUE)
levels_90 <- max_ll - qchisq(0.90, df = 2) / 2
levels_95 <- max_ll - qchisq(0.95, df = 2) / 2
levels_99 <- max_ll - qchisq(0.99, df = 2) / 2

# Plot contour
contour(alpha_2d, beta_2d, ll_surface,
        xlab = expression(alpha), ylab = expression(beta),
        main = "2D Log-Likelihood: Alpha vs Beta",
        levels = seq(min(ll_surface, na.rm = TRUE), max_ll, length.out = 20),
        col = "#2E4057", las = 1, lwd = 1)

contour(alpha_2d, beta_2d, ll_surface,
        levels = c(levels_90, levels_95, levels_99),
        col = c("#FFA07A", "#FF6347", "#8B0000"),
        lwd = c(2, 2.5, 3), lty = c(3, 2, 1),
        add = TRUE, labcex = 0.8)

points(mle[1], mle[2], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[2], pch = 17, col = "#006400", cex = 1.5)

legend("topright",
       legend = c("MLE", "True", "90% CR", "95% CR", "99% CR"),
       col = c("#8B0000", "#006400", "#FFA07A", "#FF6347", "#8B0000"),
       pch = c(19, 17, NA, NA, NA),
       lty = c(NA, NA, 3, 2, 1),
       lwd = c(NA, NA, 2, 2.5, 3),
       bty = "n", cex = 0.8)
grid(col = "gray90")


## Example 7: 2D Log-Likelihood Surface (Delta vs Lambda)

# Create 2D grid
delta_2d <- seq(mle[3] - 0.6, mle[3] + 0.6, length.out = round(n/25))
lambda_2d <- seq(mle[4] - 0.8, mle[4] + 0.8, length.out = round(n/25))
delta_2d <- delta_2d[delta_2d > 0]
lambda_2d <- lambda_2d[lambda_2d > 0]

# Compute log-likelihood surface
ll_surface2 <- matrix(NA, nrow = length(delta_2d), ncol = length(lambda_2d))

for (i in seq_along(delta_2d)) {
  for (j in seq_along(lambda_2d)) {
    ll_surface2[i, j] <- -llkkw(c(mle[1], mle[2], delta_2d[i], lambda_2d[j]), data)
  }
}

# Confidence region levels
max_ll2 <- max(ll_surface2, na.rm = TRUE)
levels2_90 <- max_ll2 - qchisq(0.90, df = 2) / 2
levels2_95 <- max_ll2 - qchisq(0.95, df = 2) / 2
levels2_99 <- max_ll2 - qchisq(0.99, df = 2) / 2

# Plot contour
contour(delta_2d, lambda_2d, ll_surface2,
        xlab = expression(delta), ylab = expression(lambda),
        main = "2D Log-Likelihood: Delta vs Lambda",
        levels = seq(min(ll_surface2, na.rm = TRUE), max_ll2, length.out = 20),
        col = "#2E4057", las = 1, lwd = 1)

contour(delta_2d, lambda_2d, ll_surface2,
        levels = c(levels2_90, levels2_95, levels2_99),
        col = c("#FFA07A", "#FF6347", "#8B0000"),
        lwd = c(2, 2.5, 3), lty = c(3, 2, 1),
        add = TRUE, labcex = 0.8)

points(mle[3], mle[4], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[3], true_params[4], pch = 17, col = "#006400", cex = 1.5)

legend("topright",
       legend = c("MLE", "True", "90% CR", "95% CR", "99% CR"),
       col = c("#8B0000", "#006400", "#FFA07A", "#FF6347", "#8B0000"),
       pch = c(19, 17, NA, NA, NA),
       lty = c(NA, NA, 3, 2, 1),
       lwd = c(NA, NA, 2, 2.5, 3),
       bty = "n", cex = 0.8)
grid(col = "gray90")
```

`llkw` *Negative Log-Likelihood of the Kumaraswamy (Kw) Distribution*

Description

Computes the negative log-likelihood function for the two-parameter Kumaraswamy (Kw) distribution with parameters `alpha` (α) and `beta` (β), given a vector of observations. This function is suitable for maximum likelihood estimation.

Usage

```
llkw(par, data)
```

Arguments

`par`	A numeric vector of length 2 containing the distribution parameters in the order: `alpha` ($\alpha > 0$), `beta` ($\beta > 0$).
`data`	A numeric vector of observations. All values must be strictly between 0 and 1 (exclusive).

Details

The Kumaraswamy (Kw) distribution's probability density function (PDF) is (see `dkw`):

$$f(x|\theta) = \alpha\beta x^{\alpha-1}(1 - x^{\alpha})^{\beta-1}$$

for $0 < x < 1$ and $\theta = (\alpha, \beta)$. The log-likelihood function $\ell(\theta|\mathbf{x})$ for a sample $\mathbf{x} = (x_1, \dots, x_n)$ is $\sum_{i=1}^{n} \ln f(x_i|\theta)$:

$$\ell(\theta|\mathbf{x}) = n[\ln(\alpha) + \ln(\beta)] + \sum_{i=1}^{n}[(\alpha - 1)\ln(x_i) + (\beta - 1)\ln(v_i)]$$

where $v_i = 1 - x_i^{\alpha}$. This function computes and returns the *negative* log-likelihood, $-\ell(\theta|\mathbf{x})$, suitable for minimization using optimization routines like `optim`. It is equivalent to the negative log-likelihood of the GKw distribution (`llgkw`) evaluated at $\gamma = 1, \delta = 0, \lambda = 1$.

Value

Returns a single `double` value representing the negative log-likelihood ($-\ell(\theta|\mathbf{x})$). Returns `Inf` if any parameter values in `par` are invalid according to their constraints, or if any value in `data` is not in the interval (0, 1).

Author(s)

Lopes, J. E.

References

Kumaraswamy, P. (1980). A generalized probability density function for double-bounded random processes. *Journal of Hydrology*, *46*(1-2), 79-88.

Jones, M. C. (2009). Kumaraswamy's distribution: A beta-type distribution with some tractability advantages. *Statistical Methodology*, *6*(1), 70-81.

See Also

`llgkw` (parent distribution negative log-likelihood), `dkw`, `pkw`, `qkw`, `rkw`, `grkw` (gradient, if available), `hskw` (Hessian, if available), `optim`

Examples

```
## Example 1: Maximum Likelihood Estimation with Analytical Gradient

# Generate sample data
set.seed(123)
n <- 1000
true_params <- c(alpha = 2.5, beta = 3.5)
data <- rkw(n, alpha = true_params[1], beta = true_params[2])

# Optimization using BFGS with analytical gradient
fit <- optim(
  par = c(2, 2),
  fn = llkw,
  gr = grkw,
  data = data,
  method = "BFGS",
  hessian = TRUE
)

# Extract results
mle <- fit$par
names(mle) <- c("alpha", "beta")
se <- sqrt(diag(solve(fit$hessian)))
ci_lower <- mle - 1.96 * se
ci_upper <- mle + 1.96 * se

# Summary table
results <- data.frame(
  Parameter = c("alpha", "beta"),
  True = true_params,
  MLE = mle,
  SE = se,
  CI_Lower = ci_lower,
  CI_Upper = ci_upper
)
print(results, digits = 4)

## Example 2: Verifying Gradient at MLE
```

```
# At MLE, gradient should be approximately zero
gradient_at_mle <- grkw(par = mle, data = data)
print(gradient_at_mle)
cat("Max absolute score:", max(abs(gradient_at_mle)), "\n")

## Example 3: Checking Hessian Properties

# Hessian at MLE
hessian_at_mle <- hskw(par = mle, data = data)
print(hessian_at_mle, digits = 4)

# Check positive definiteness via eigenvalues
eigenvals <- eigen(hessian_at_mle, only.values = TRUE)$values
print(eigenvals)
all(eigenvals > 0)

# Condition number
cond_number <- max(eigenvals) / min(eigenvals)
cat("Condition number:", format(cond_number, scientific = TRUE), "\n")

## Example 4: Comparing Optimization Methods

methods <- c("BFGS", "L-BFGS-B", "Nelder-Mead", "CG")
start_params <- c(2, 2)

comparison <- data.frame(
  Method = character(),
  Alpha_Est = numeric(),
  Beta_Est = numeric(),
  NegLogLik = numeric(),
  Convergence = integer(),
  stringsAsFactors = FALSE
)

for (method in methods) {
  if (method %in% c("BFGS", "CG")) {
    fit_temp <- optim(
      par = start_params,
      fn = llkw,
      gr = grkw,
      data = data,
      method = method
    )
  } else if (method == "L-BFGS-B") {
    fit_temp <- optim(
      par = start_params,
      fn = llkw,
      gr = grkw,
      data = data,
      method = method,
      lower = c(0.01, 0.01),
      upper = c(100, 100)
    )
  } else {
    fit_temp <- optim(
      par = start_params,
      fn = llkw,
      data = data,
      method = method
    )
  }

  comparison <- rbind(comparison, data.frame(
    Method = method,
    Alpha_Est = fit_temp$par[1],
    Beta_Est = fit_temp$par[2],
    NegLogLik = fit_temp$value,
    Convergence = fit_temp$convergence,
    stringsAsFactors = FALSE
  ))
}

print(comparison, digits = 4, row.names = FALSE)

## Example 5: Likelihood Ratio Test

# Test H0: beta = 3 vs H1: beta free
loglik_full <- -fit$value

# Restricted model: fix beta = 3
restricted_ll <- function(alpha, data, beta_fixed) {
  llkw(par = c(alpha, beta_fixed), data = data)
}

fit_restricted <- optimize(
  f = restricted_ll,
  interval = c(0.1, 10),
  data = data,
  beta_fixed = 3,
  maximum = FALSE
)

loglik_restricted <- -fit_restricted$objective
lr_stat <- 2 * (loglik_full - loglik_restricted)
p_value <- pchisq(lr_stat, df = 1, lower.tail = FALSE)

cat("LR Statistic:", round(lr_stat, 4), "\n")
cat("P-value:", format.pval(p_value, digits = 4), "\n")

## Example 6: Univariate Profile Likelihoods

# Grid for alpha
alpha_grid <- seq(mle[1] - 1.5, mle[1] + 1.5, length.out = 50)
alpha_grid <- alpha_grid[alpha_grid > 0]
profile_ll_alpha <- numeric(length(alpha_grid))
```

```
for (i in seq_along(alpha_grid)) {
  profile_fit <- optimize(
    f = function(beta) llkw(c(alpha_grid[i], beta), data),
    interval = c(0.1, 10),
    maximum = FALSE
  )
  profile_ll_alpha[i] <- -profile_fit$objective
}

# Grid for beta
beta_grid <- seq(mle[2] - 1.5, mle[2] + 1.5, length.out = 50)
beta_grid <- beta_grid[beta_grid > 0]
profile_ll_beta <- numeric(length(beta_grid))

for (i in seq_along(beta_grid)) {
  profile_fit <- optimize(
    f = function(alpha) llkw(c(alpha, beta_grid[i]), data),
    interval = c(0.1, 10),
    maximum = FALSE
  )
  profile_ll_beta[i] <- -profile_fit$objective
}

# 95% confidence threshold
chi_crit <- qchisq(0.95, df = 1)
threshold <- max(profile_ll_alpha) - chi_crit / 2

# Plot

# Profile for alpha
plot(alpha_grid, profile_ll_alpha, type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(alpha), ylab = "Profile Log-Likelihood",
     main = expression(paste("Profile Likelihood: ", alpha)), las = 1)
abline(v = mle[1], col = "#8B0000", lty = 2, lwd = 2)
abline(v = true_params[1], col = "#006400", lty = 2, lwd = 2)
abline(h = threshold, col = "#808080", lty = 3, lwd = 1.5)
legend("topright",
       legend = c("MLE", "True", "95% CI"),
       col = c("#8B0000", "#006400", "#808080"),
       lty = c(2, 2, 3), lwd = 2, bty = "n", cex = 0.8)
grid(col = "gray90")

# Profile for beta
plot(beta_grid, profile_ll_beta, type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(beta), ylab = "Profile Log-Likelihood",
     main = expression(paste("Profile Likelihood: ", beta)), las = 1)
abline(v = mle[2], col = "#8B0000", lty = 2, lwd = 2)
abline(v = true_params[2], col = "#006400", lty = 2, lwd = 2)
abline(h = threshold, col = "#808080", lty = 3, lwd = 1.5)
legend("topright",
       legend = c("MLE", "True", "95% CI"),
       col = c("#8B0000", "#006400", "#808080"),
       lty = c(2, 2, 3), lwd = 2, bty = "n", cex = 0.8)
```

```
grid(col = "gray90")

## Example 7: 2D Profile Likelihood Surface

# Create 2D grid
alpha_2d <- seq(mle[1] - 1, mle[1] + 1, length.out = round(n/4))
beta_2d <- seq(mle[2] - 1, mle[2] + 1, length.out = round(n/4))
alpha_2d <- alpha_2d[alpha_2d > 0]
beta_2d <- beta_2d[beta_2d > 0]

# Compute log-likelihood surface
ll_surface <- matrix(NA, nrow = length(alpha_2d), ncol = length(beta_2d))

for (i in seq_along(alpha_2d)) {
  for (j in seq_along(beta_2d)) {
    ll_surface[i, j] <- -llkw(c(alpha_2d[i], beta_2d[j]), data)
  }
}

# Confidence region levels
max_ll <- max(ll_surface, na.rm = TRUE)
levels_90 <- max_ll - qchisq(0.90, df = 2) / 2
levels_95 <- max_ll - qchisq(0.95, df = 2) / 2
levels_99 <- max_ll - qchisq(0.99, df = 2) / 2

# Plot contour
contour(alpha_2d, beta_2d, ll_surface,
        xlab = expression(alpha), ylab = expression(beta),
        main = "2D Profile Log-Likelihood",
        levels = seq(min(ll_surface, na.rm = TRUE), max_ll, length.out = round(n/4)),
        col = "#2E4057", las = 1, lwd = 1)

# Add confidence region contours
contour(alpha_2d, beta_2d, ll_surface,
        levels = c(levels_90, levels_95, levels_99),
        col = c("#FFA07A", "#FF6347", "#8B0000"),
        lwd = c(2, 2.5, 3), lty = c(3, 2, 1),
        add = TRUE, labcex = 0.8)

# Mark points
points(mle[1], mle[2], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[2], pch = 17, col = "#006400", cex = 1.5)

legend("topright",
       legend = c("MLE", "True", "90% CR", "95% CR", "99% CR"),
       col = c("#8B0000", "#006400", "#FFA07A", "#FF6347", "#8B0000"),
       pch = c(19, 17, NA, NA, NA),
       lty = c(NA, NA, 3, 2, 1),
       lwd = c(NA, NA, 2, 2.5, 3),
       bty = "n", cex = 0.8)
grid(col = "gray90")

## Example 8: Combined View - Profiles with 2D Surface
```

```
# Top left: Profile for alpha
plot(alpha_grid, profile_ll_alpha, type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(alpha), ylab = "Profile Log-Likelihood",
     main = expression(paste("Profile: ", alpha)), las = 1)
abline(v = mle[1], col = "#8B0000", lty = 2, lwd = 2)
abline(v = true_params[1], col = "#006400", lty = 2, lwd = 2)
abline(h = threshold, col = "#808080", lty = 3)
grid(col = "gray90")

# Top right: Profile for beta
plot(beta_grid, profile_ll_beta, type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(beta), ylab = "Profile Log-Likelihood",
     main = expression(paste("Profile: ", beta)), las = 1)
abline(v = mle[2], col = "#8B0000", lty = 2, lwd = 2)
abline(v = true_params[2], col = "#006400", lty = 2, lwd = 2)
abline(h = threshold, col = "#808080", lty = 3)
grid(col = "gray90")

# Bottom left: 2D contour
contour(alpha_2d, beta_2d, ll_surface,
        xlab = expression(alpha), ylab = expression(beta),
        main = "2D Log-Likelihood Surface",
        levels = seq(min(ll_surface, na.rm = TRUE), max_ll, length.out = 15),
        col = "#2E4057", las = 1, lwd = 1)
contour(alpha_2d, beta_2d, ll_surface,
        levels = c(levels_95),
        col = "#8B0000", lwd = 2.5, add = TRUE)
points(mle[1], mle[2], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[2], pch = 17, col = "#006400", cex = 1.5)
grid(col = "gray90")

## Example 9: Numerical Gradient Verification

# Manual finite difference gradient
numerical_gradient <- function(f, x, data, h = 1e-7) {
  grad <- numeric(length(x))
  for (i in seq_along(x)) {
    x_plus <- x_minus <- x
    x_plus[i] <- x[i] + h
    x_minus[i] <- x[i] - h
    grad[i] <- (f(x_plus, data) - f(x_minus, data)) / (2 * h)
  }
  return(grad)
}

# Compare
grad_analytical <- grkw(par = mle, data = data)
grad_numerical <- numerical_gradient(llkw, mle, data)

comparison_grad <- data.frame(
  Parameter = c("alpha", "beta"),
  Analytical = grad_analytical,
  Numerical = grad_numerical,
  Difference = abs(grad_analytical - grad_numerical)
)
print(comparison_grad, digits = 8)

## Example 10: Bootstrap Confidence Intervals

n_boot <- round(n/4)
boot_estimates <- matrix(NA, nrow = n_boot, ncol = 2)

set.seed(456)
for (b in 1:n_boot) {
  boot_data <- rkw(n, alpha = mle[1], beta = mle[2])
  boot_fit <- optim(
    par = mle,
    fn = llkw,
    gr = grkw,
    data = boot_data,
    method = "BFGS",
    control = list(maxit = 500)
  )
  if (boot_fit$convergence == 0) {
    boot_estimates[b, ] <- boot_fit$par
  }
}

boot_estimates <- boot_estimates[complete.cases(boot_estimates), ]
boot_ci <- apply(boot_estimates, 2, quantile, probs = c(0.025, 0.975))
colnames(boot_ci) <- c("alpha", "beta")

print(t(boot_ci), digits = 4)

# Plot bootstrap distributions

hist(boot_estimates[, 1], breaks = 20, col = "#87CEEB", border = "white",
     main = expression(paste("Bootstrap: ", hat(alpha))),
     xlab = expression(hat(alpha)), las = 1)
abline(v = mle[1], col = "#8B0000", lwd = 2)
abline(v = true_params[1], col = "#006400", lwd = 2, lty = 2)
abline(v = boot_ci[, 1], col = "#2E4057", lwd = 2, lty = 3)
legend("topright", legend = c("MLE", "True", "95% CI"),
       col = c("#8B0000", "#006400", "#2E4057"),
       lwd = 2, lty = c(1, 2, 3), bty = "n")

hist(boot_estimates[, 2], breaks = 20, col = "#FFA07A", border = "white",
     main = expression(paste("Bootstrap: ", hat(beta))),
     xlab = expression(hat(beta)), las = 1)
abline(v = mle[2], col = "#8B0000", lwd = 2)
abline(v = true_params[2], col = "#006400", lwd = 2, lty = 2)
abline(v = boot_ci[, 2], col = "#2E4057", lwd = 2, lty = 3)
legend("topright", legend = c("MLE", "True", "95% CI"),
       col = c("#8B0000", "#006400", "#2E4057"),
       lwd = 2, lty = c(1, 2, 3), bty = "n")
```

llmc *Negative Log-Likelihood for the McDonald (Mc)/Beta Power Distribution*

Description

Computes the negative log-likelihood function for the McDonald (Mc) distribution (also known as Beta Power) with parameters `gamma` (γ), `delta` (δ), and `lambda` (λ), given a vector of observations. This distribution is the special case of the Generalized Kumaraswamy (GKw) distribution where $\alpha = 1$ and $\beta = 1$. This function is suitable for maximum likelihood estimation.

Usage

```
llmc(par, data)
```

Arguments

`par`	A numeric vector of length 3 containing the distribution parameters in the order: `gamma` ($\gamma > 0$), `delta` ($\delta \geq 0$), `lambda` ($\lambda > 0$).
`data`	A numeric vector of observations. All values must be strictly between 0 and 1 (exclusive).

Details

The McDonald (Mc) distribution is the GKw distribution (`dmc`) with $\alpha = 1$ and $\beta = 1$. Its probability density function (PDF) is:

$$f(x|\theta) = \frac{\lambda}{B(\gamma, \delta+1)} x^{\gamma\lambda - 1} (1 - x^\lambda)^\delta$$

for $0 < x < 1$, $\theta = (\gamma, \delta, \lambda)$, and $B(a, b)$ is the Beta function (`beta`). The log-likelihood function $\ell(\theta|\mathbf{x})$ for a sample $\mathbf{x} = (x_1, \ldots, x_n)$ is $\sum_{i=1}^n \ln f(x_i|\theta)$:

$$\ell(\theta|\mathbf{x}) = n[\ln(\lambda) - \ln B(\gamma, \delta+1)] + \sum_{i=1}^{n} [(\gamma\lambda - 1)\ln(x_i) + \delta \ln(1 - x_i^\lambda)]$$

This function computes and returns the *negative* log-likelihood, $-\ell(\theta|\mathbf{x})$, suitable for minimization using optimization routines like `optim`. Numerical stability is maintained, including using the log-gamma function (`lgamma`) for the Beta function term.

Value

Returns a single `double` value representing the negative log-likelihood ($-\ell(\theta|\mathbf{x})$). Returns `Inf` if any parameter values in `par` are invalid according to their constraints, or if any value in `data` is not in the interval (0, 1).

Author(s)

Lopes, J. E.

References

McDonald, J. B. (1984). Some generalized functions for the size distribution of income. *Econometrica*, 52(3), 647-663.

Cordeiro, G. M., & de Castro, M. (2011). A new family of generalized distributions. *Journal of Statistical Computation and Simulation*,

Kumaraswamy, P. (1980). A generalized probability density function for double-bounded random processes. *Journal of Hydrology*, *46*(1-2), 79-88.

See Also

`llgkw` (parent distribution negative log-likelihood), `dmc`, `pmc`, `qmc`, `rmc`, `grmc` (gradient, if available), `hsmc` (Hessian, if available), `optim`, `lbeta`

Examples

```
## Example 1: Basic Log-Likelihood Evaluation

# Generate sample data with more stable parameters
set.seed(123)
n <- 1000
true_params <- c(gamma = 2.0, delta = 2.5, lambda = 1.5)
data <- rmc(n, gamma = true_params[1], delta = true_params[2],
            lambda = true_params[3])

# Evaluate negative log-likelihood at true parameters
nll_true <- llmc(par = true_params, data = data)
cat("Negative log-likelihood at true parameters:", nll_true, "\n")

# Evaluate at different parameter values
test_params <- rbind(
  c(1.5, 2.0, 1.0),
  c(2.0, 2.5, 1.5),
  c(2.5, 3.0, 2.0)
)

nll_values <- apply(test_params, 1, function(p) llmc(p, data))
results <- data.frame(
  Gamma = test_params[, 1],
  Delta = test_params[, 2],
  Lambda = test_params[, 3],
  NegLogLik = nll_values
)
print(results, digits = 4)


## Example 2: Maximum Likelihood Estimation
```

```
# Optimization using BFGS with analytical gradient
fit <- optim(
  par = c(1.5, 2.0, 1.0),
  fn = llmc,
  gr = grmc,
  data = data,
  method = "BFGS",
  hessian = TRUE
)

mle <- fit$par
names(mle) <- c("gamma", "delta", "lambda")
se <- sqrt(diag(solve(fit$hessian)))

results <- data.frame(
  Parameter = c("gamma", "delta", "lambda"),
  True = true_params,
  MLE = mle,
  SE = se,
  CI_Lower = mle - 1.96 * se,
  CI_Upper = mle + 1.96 * se
)
print(results, digits = 4)

cat("\nNegative log-likelihood at MLE:", fit$value, "\n")
cat("AIC:", 2 * fit$value + 2 * length(mle), "\n")
cat("BIC:", 2 * fit$value + length(mle) * log(n), "\n")


## Example 3: Comparing Optimization Methods

methods <- c("BFGS", "L-BFGS-B", "Nelder-Mead", "CG")
start_params <- c(1.5, 2.0, 1.0)

comparison <- data.frame(
  Method = character(),
  Gamma = numeric(),
  Delta = numeric(),
  Lambda = numeric(),
  NegLogLik = numeric(),
  Convergence = integer(),
  stringsAsFactors = FALSE
)

for (method in methods) {
  if (method %in% c("BFGS", "CG")) {
    fit_temp <- optim(
      par = start_params,
      fn = llmc,
      gr = grmc,
      data = data,
      method = method
    )
  } else if (method == "L-BFGS-B") {
    fit_temp <- optim(
      par = start_params,
      fn = llmc,
      gr = grmc,
      data = data,
      method = method,
      lower = c(0.01, 0.01, 0.01),
      upper = c(100, 100, 100)
    )
  } else {
    fit_temp <- optim(
      par = start_params,
      fn = llmc,
      data = data,
      method = method
    )
  }

  comparison <- rbind(comparison, data.frame(
    Method = method,
    Gamma = fit_temp$par[1],
    Delta = fit_temp$par[2],
    Lambda = fit_temp$par[3],
    NegLogLik = fit_temp$value,
    Convergence = fit_temp$convergence,
    stringsAsFactors = FALSE
  ))
}

print(comparison, digits = 4, row.names = FALSE)


## Example 4: Likelihood Ratio Test

# Test H0: lambda = 1.5 vs H1: lambda free
loglik_full <- -fit$value

restricted_ll <- function(params_restricted, data, lambda_fixed) {
  llmc(par = c(params_restricted[1], params_restricted[2],
               lambda_fixed), data = data)
}

fit_restricted <- optim(
  par = c(mle[1], mle[2]),
  fn = restricted_ll,
  data = data,
  lambda_fixed = 1.5,
  method = "BFGS"
)

loglik_restricted <- -fit_restricted$value
lr_stat <- 2 * (loglik_full - loglik_restricted)
```

```
p_value <- pchisq(lr_stat, df = 1, lower.tail = FALSE)

cat("LR Statistic:", round(lr_stat, 4), "\n")
cat("P-value:", format.pval(p_value, digits = 4), "\n")


## Example 5: Univariate Profile Likelihoods

# Profile for gamma
gamma_grid <- seq(mle[1] - 1.5, mle[1] + 1.5, length.out = 50)
gamma_grid <- gamma_grid[gamma_grid > 0]
profile_ll_gamma <- numeric(length(gamma_grid))

for (i in seq_along(gamma_grid)) {
  profile_fit <- optim(
    par = mle[-1],
    fn = function(p) llmc(c(gamma_grid[i], p), data),
    method = "BFGS"
  )
  profile_ll_gamma[i] <- -profile_fit$value
}

# Profile for delta
delta_grid <- seq(mle[2] - 1.5, mle[2] + 1.5, length.out = 50)
delta_grid <- delta_grid[delta_grid > 0]
profile_ll_delta <- numeric(length(delta_grid))

for (i in seq_along(delta_grid)) {
  profile_fit <- optim(
    par = mle[-2],
    fn = function(p) llmc(c(p[1], delta_grid[i], p[2]), data),
    method = "BFGS"
  )
  profile_ll_delta[i] <- -profile_fit$value
}

# Profile for lambda
lambda_grid <- seq(mle[3] - 1.5, mle[3] + 1.5, length.out = 50)
lambda_grid <- lambda_grid[lambda_grid > 0]
profile_ll_lambda <- numeric(length(lambda_grid))

for (i in seq_along(lambda_grid)) {
  profile_fit <- optim(
    par = mle[-3],
    fn = function(p) llmc(c(p[1], p[2], lambda_grid[i]), data),
    method = "BFGS"
  )
  profile_ll_lambda[i] <- -profile_fit$value
}

# 95% confidence threshold
chi_crit <- qchisq(0.95, df = 1)
threshold <- max(profile_ll_gamma) - chi_crit / 2
```

```
# Plot all profiles

plot(gamma_grid, profile_ll_gamma, type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(gamma), ylab = "Profile Log-Likelihood",
     main = expression(paste("Profile: ", gamma)), las = 1)
abline(v = mle[1], col = "#8B0000", lty = 2, lwd = 2)
abline(v = true_params[1], col = "#006400", lty = 2, lwd = 2)
abline(h = threshold, col = "#808080", lty = 3, lwd = 1.5)
legend("topright", legend = c("MLE", "True", "95% CI"),
       col = c("#8B0000", "#006400", "#808080"),
       lty = c(2, 2, 3), lwd = 2, bty = "n", cex = 0.8)
grid(col = "gray90")

plot(delta_grid, profile_ll_delta, type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(delta), ylab = "Profile Log-Likelihood",
     main = expression(paste("Profile: ", delta)), las = 1)
abline(v = mle[2], col = "#8B0000", lty = 2, lwd = 2)
abline(v = true_params[2], col = "#006400", lty = 2, lwd = 2)
abline(h = threshold, col = "#808080", lty = 3, lwd = 1.5)
legend("topright", legend = c("MLE", "True", "95% CI"),
       col = c("#8B0000", "#006400", "#808080"),
       lty = c(2, 2, 3), lwd = 2, bty = "n", cex = 0.8)
grid(col = "gray90")

plot(lambda_grid, profile_ll_lambda, type = "l", lwd = 2, col = "#2E4057",
     xlab = expression(lambda), ylab = "Profile Log-Likelihood",
     main = expression(paste("Profile: ", lambda)), las = 1)
abline(v = mle[3], col = "#8B0000", lty = 2, lwd = 2)
abline(v = true_params[3], col = "#006400", lty = 2, lwd = 2)
abline(h = threshold, col = "#808080", lty = 3, lwd = 1.5)
legend("topright", legend = c("MLE", "True", "95% CI"),
       col = c("#8B0000", "#006400", "#808080"),
       lty = c(2, 2, 3), lwd = 2, bty = "n", cex = 0.8)
grid(col = "gray90")

## Example 6: 2D Log-Likelihood Surfaces (All pairs side by side)

# Create 2D grids with wider range (±1.5)
gamma_2d <- seq(mle[1] - 1.5, mle[1] + 1.5, length.out = round(n/25))
delta_2d <- seq(mle[2] - 1.5, mle[2] + 1.5, length.out = round(n/25))
lambda_2d <- seq(mle[3] - 1.5, mle[3] + 1.5, length.out = round(n/25))

gamma_2d <- gamma_2d[gamma_2d > 0]
delta_2d <- delta_2d[delta_2d > 0]
lambda_2d <- lambda_2d[lambda_2d > 0]

# Compute all log-likelihood surfaces
ll_surface_gd <- matrix(NA, nrow = length(gamma_2d), ncol = length(delta_2d))
ll_surface_gl <- matrix(NA, nrow = length(gamma_2d), ncol = length(lambda_2d))
ll_surface_dl <- matrix(NA, nrow = length(delta_2d), ncol = length(lambda_2d))

# Gamma vs Delta
```

```
for (i in seq_along(gamma_2d)) {
  for (j in seq_along(delta_2d)) {
    ll_surface_gd[i, j] <- -llmc(c(gamma_2d[i], delta_2d[j], mle[3]), data)
  }
}

# Gamma vs Lambda
for (i in seq_along(gamma_2d)) {
  for (j in seq_along(lambda_2d)) {
    ll_surface_gl[i, j] <- -llmc(c(gamma_2d[i], mle[2], lambda_2d[j]), data)
  }
}

# Delta vs Lambda
for (i in seq_along(delta_2d)) {
  for (j in seq_along(lambda_2d)) {
    ll_surface_dl[i, j] <- -llmc(c(mle[1], delta_2d[i], lambda_2d[j]), data)
  }
}

# Confidence region levels
max_ll_gd <- max(ll_surface_gd, na.rm = TRUE)
max_ll_gl <- max(ll_surface_gl, na.rm = TRUE)
max_ll_dl <- max(ll_surface_dl, na.rm = TRUE)

levels_95_gd <- max_ll_gd - qchisq(0.95, df = 2) / 2
levels_95_gl <- max_ll_gl - qchisq(0.95, df = 2) / 2
levels_95_dl <- max_ll_dl - qchisq(0.95, df = 2) / 2

# Plot

# Gamma vs Delta
contour(gamma_2d, delta_2d, ll_surface_gd,
        xlab = expression(gamma), ylab = expression(delta),
        main = "Gamma vs Delta", las = 1,
        levels = seq(min(ll_surface_gd, na.rm = TRUE), max_ll_gd, length.out = 20),
        col = "#2E4057", lwd = 1)
contour(gamma_2d, delta_2d, ll_surface_gd,
        levels = levels_95_gd, col = "#FF6347", lwd = 2.5, lty = 1, add = TRUE)
points(mle[1], mle[2], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[2], pch = 17, col = "#006400", cex = 1.5)
grid(col = "gray90")

# Gamma vs Lambda
contour(gamma_2d, lambda_2d, ll_surface_gl,
        xlab = expression(gamma), ylab = expression(lambda),
        main = "Gamma vs Lambda", las = 1,
        levels = seq(min(ll_surface_gl, na.rm = TRUE), max_ll_gl, length.out = 20),
        col = "#2E4057", lwd = 1)
contour(gamma_2d, lambda_2d, ll_surface_gl,
        levels = levels_95_gl, col = "#FF6347", lwd = 2.5, lty = 1, add = TRUE)
points(mle[1], mle[3], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[1], true_params[3], pch = 17, col = "#006400", cex = 1.5)
```

```
grid(col = "gray90")

# Delta vs Lambda
contour(delta_2d, lambda_2d, ll_surface_dl,
        xlab = expression(delta), ylab = expression(lambda),
        main = "Delta vs Lambda", las = 1,
        levels = seq(min(ll_surface_dl, na.rm = TRUE), max_ll_dl, length.out = 20),
        col = "#2E4057", lwd = 1)
contour(delta_2d, lambda_2d, ll_surface_dl,
        levels = levels_95_dl, col = "#FF6347", lwd = 2.5, lty = 1, add = TRUE)
points(mle[2], mle[3], pch = 19, col = "#8B0000", cex = 1.5)
points(true_params[2], true_params[3], pch = 17, col = "#006400", cex = 1.5)
grid(col = "gray90")

legend("topright",
       legend = c("MLE", "True", "95% CR"),
       col = c("#8B0000", "#006400", "#FF6347"),
       pch = c(19, 17, NA),
       lty = c(NA, NA, 1),
       lwd = c(NA, NA, 2.5),
       bty = "n", cex = 0.8)
```

pbeta_ *CDF of the Beta Distribution (gamma, delta+1 Parameterization)*

Description

Computes the cumulative distribution function (CDF), $F(q) = P(X \leq q)$, for the standard Beta distribution, using a parameterization common in generalized distribution families. The distribution is parameterized by `gamma` (γ) and `delta` (δ), corresponding to the standard Beta distribution with shape parameters `shape1 = gamma` and `shape2 = delta + 1`.

Usage

```
pbeta_(q, gamma, delta, lower_tail = TRUE, log_p = FALSE)
```

Arguments

`q`	Vector of quantiles (values generally between 0 and 1).
`gamma`	First shape parameter (`shape1`), $\gamma > 0$. Can be a scalar or a vector. Default: 1.0.
`delta`	Second shape parameter is `delta + 1` (`shape2`), requires $\delta \geq 0$ so that `shape2 >= 1`. Can be a scalar or a vector. Default: 0.0 (leading to `shape2 = 1`).
`lower_tail`	Logical; if `TRUE` (default), probabilities are $P(X \leq q)$, otherwise, $P(X > q)$.
`log_p`	Logical; if `TRUE`, probabilities p are given as $\log(p)$. Default: `FALSE`.

Details

This function computes the CDF of a Beta distribution with parameters `shape1 = gamma` and `shape2 = delta + 1`. It is equivalent to calling `stats::pbeta(q, shape1 = gamma, shape2 = delta + 1,lower.tail = lower_tail, log.p = log_p)`.

This distribution arises as a special case of the five-parameter Generalized Kumaraswamy (GKw) distribution (`pgkw`) obtained by setting $\alpha = 1$, $\beta = 1$, and $\lambda = 1$. It is therefore also equivalent to the McDonald (Mc)/Beta Power distribution (`pmc`) with $\lambda = 1$.

The function likely calls R's underlying `pbeta` function but ensures consistent parameter recycling and handling within the C++ environment, matching the style of other functions in the related families.

Value

A vector of probabilities, $F(q)$, or their logarithms/complements depending on `lower_tail` and `log_p`. The length of the result is determined by the recycling rule applied to the arguments (`q`, `gamma`, `delta`). Returns `0` (or `-Inf` if `log_p = TRUE`) for `q <= 0` and `1` (or `0` if `log_p = TRUE`) for `q >= 1`. Returns `NaN` for invalid parameters.

Author(s)

Lopes, J. E.

References

Johnson, N. L., Kotz, S., & Balakrishnan, N. (1995). *Continuous Univariate Distributions, Volume 2* (2nd ed.). Wiley.

Cordeiro, G. M., & de Castro, M. (2011). A new family of generalized distributions. *Journal of Statistical Computation and Simulation*,

See Also

`pbeta` (standard R implementation), `pgkw` (parent distribution CDF), `pmc` (McDonald/Beta Power CDF), `dbeta_`, `qbeta_`, `rbeta_` (other functions for this parameterization, if they exist).

Examples

```
# Example values
q_vals <- c(0.2, 0.5, 0.8)
gamma_par <- 2.0 # Corresponds to shape1
delta_par <- 3.0 # Corresponds to shape2 - 1
shape1 <- gamma_par
shape2 <- delta_par + 1

# Calculate CDF using pbeta_
probs <- pbeta_(q_vals, gamma_par, delta_par)
print(probs)

# Compare with stats::pbeta
probs_stats <- stats::pbeta(q_vals, shape1 = shape1, shape2 = shape2)
```

```
print(paste("Max difference vs stats::pbeta:", max(abs(probs - probs_stats))))

# Compare with pgkw setting alpha=1, beta=1, lambda=1
probs_gkw <- pgkw(q_vals, alpha = 1.0, beta = 1.0, gamma = gamma_par,
                  delta = delta_par, lambda = 1.0)
print(paste("Max difference vs pgkw:", max(abs(probs - probs_gkw))))

# Compare with pmc setting lambda=1
probs_mc <- pmc(q_vals, gamma = gamma_par, delta = delta_par, lambda = 1.0)
print(paste("Max difference vs pmc:", max(abs(probs - probs_mc))))

# Calculate upper tail P(X > q)
probs_upper <- pbeta_(q_vals, gamma_par, delta_par, lower_tail = FALSE)
print(probs_upper)
print(stats::pbeta(q_vals, shape1, shape2, lower.tail = FALSE))

# Calculate log CDF
log_probs <- pbeta_(q_vals, gamma_par, delta_par, log_p = TRUE)
print(log_probs)
print(stats::pbeta(q_vals, shape1, shape2, log.p = TRUE))

# Plot the CDF
curve_q <- seq(0.001, 0.999, length.out = 200)
curve_p <- pbeta_(curve_q, gamma = 2, delta = 3) # Beta(2, 4)
plot(curve_q, curve_p, type = "l", main = "Beta(2, 4) CDF via pbeta_",
     xlab = "q", ylab = "F(q)", col = "blue")
curve(stats::pbeta(x, 2, 4), add=TRUE, col="red", lty=2)
legend("bottomright", legend=c("pbeta_(gamma=2, delta=3)", "stats::pbeta(shape1=2, shape2=4)"),
       col=c("blue", "red"), lty=c(1,2), bty="n")
```

pbkw *Cumulative Distribution Function (CDF) of the Beta-Kumaraswamy (BKw) Distribution*

Description

Computes the cumulative distribution function (CDF), $P(X \leq q)$, for the Beta-Kumaraswamy (BKw) distribution with parameters `alpha` (α), `beta` (β), `gamma` (γ), and `delta` (δ). This distribution is defined on the interval (0, 1) and is a special case of the Generalized Kumaraswamy (GKw) distribution where $\lambda = 1$.

Usage

```
pbkw(q, alpha, beta, gamma, delta, lower_tail = TRUE, log_p = FALSE)
```

Arguments

`q`	Vector of quantiles (values generally between 0 and 1).
`alpha`	Shape parameter `alpha` > 0. Can be a scalar or a vector. Default: 1.0.
`beta`	Shape parameter `beta` > 0. Can be a scalar or a vector. Default: 1.0.
`gamma`	Shape parameter `gamma` > 0. Can be a scalar or a vector. Default: 1.0.
`delta`	Shape parameter `delta` >= 0. Can be a scalar or a vector. Default: 0.0.
`lower_tail`	Logical; if `TRUE` (default), probabilities are $P(X \leq q)$, otherwise, $P(X > q)$.
`log_p`	Logical; if `TRUE`, probabilities p are given as $\log(p)$. Default: `FALSE`.

Details

The Beta-Kumaraswamy (BKw) distribution is a special case of the five-parameter Generalized Kumaraswamy distribution (`pgkw`) obtained by setting the shape parameter $\lambda = 1$.

The CDF of the GKw distribution is $F_{GKw}(q) = I_{y(q)}(\gamma, \delta + 1)$, where $y(q) = [1 - (1 - q^\alpha)^\beta]^\lambda$ and $I_x(a, b)$ is the regularized incomplete beta function (`pbeta`). Setting $\lambda = 1$ simplifies $y(q)$ to $1 - (1 - q^\alpha)^\beta$, yielding the BKw CDF:

$$F(q; \alpha, \beta, \gamma, \delta) = I_{1-(1-q^\alpha)^\beta}(\gamma, \delta + 1)$$

This is evaluated using the `pbeta` function.

Value

A vector of probabilities, $F(q)$, or their logarithms/complements depending on `lower_tail` and `log_p`. The length of the result is determined by the recycling rule applied to the arguments (`q`, `alpha`, `beta`, `gamma`, `delta`). Returns `0` (or `-Inf` if `log_p = TRUE`) for `q <= 0` and `1` (or `0` if `log_p = TRUE`) for `q >= 1`. Returns `NaN` for invalid parameters.

Author(s)

Lopes, J. E.

References

Cordeiro, G. M., & de Castro, M. (2011). A new family of generalized distributions. *Journal of Statistical Computation and Simulation*

Kumaraswamy, P. (1980). A generalized probability density function for double-bounded random processes. *Journal of Hydrology*, *46*(1-2), 79-88.

See Also

`pgkw` (parent distribution CDF), `dbkw`, `qbkw`, `rbkw` (other BKw functions), `pbeta`

Examples

```
# Example values
q_vals <- c(0.2, 0.5, 0.8)
alpha_par <- 2.0
beta_par <- 1.5
gamma_par <- 1.0
delta_par <- 0.5

# Calculate CDF P(X <= q)
probs <- pbkw(q_vals, alpha_par, beta_par, gamma_par, delta_par)
print(probs)

# Calculate upper tail P(X > q)
probs_upper <- pbkw(q_vals, alpha_par, beta_par, gamma_par, delta_par,
                    lower_tail = FALSE)
print(probs_upper)
# Check: probs + probs_upper should be 1
print(probs + probs_upper)

# Calculate log CDF
log_probs <- pbkw(q_vals, alpha_par, beta_par, gamma_par, delta_par,
                  log_p = TRUE)
print(log_probs)
# Check: should match log(probs)
print(log(probs))

# Compare with pgkw setting lambda = 1
probs_gkw <- pgkw(q_vals, alpha_par, beta_par, gamma = gamma_par,
                 delta = delta_par, lambda = 1.0)
print(paste("Max difference:", max(abs(probs - probs_gkw)))) # Should be near zero

# Plot the CDF
curve_q <- seq(0.01, 0.99, length.out = 200)
curve_p <- pbkw(curve_q, alpha = 2, beta = 3, gamma = 0.5, delta = 1)
plot(curve_q, curve_p, type = "l", main = "BKw CDF Example",
     xlab = "q", ylab = "F(q)", col = "blue", ylim = c(0, 1))
```

---

pekw *Cumulative Distribution Function (CDF) of the EKw Distribution*

Description

Computes the cumulative distribution function (CDF), $P(X \leq q)$, for the Exponentiated Kumaraswamy (EKw) distribution with parameters alpha (α), beta (β), and lambda (λ). This distribution is defined on the interval (0, 1) and is a special case of the Generalized Kumaraswamy (GKw) distribution where $\gamma = 1$ and $\delta = 0$.

Usage

```
pekw(q, alpha, beta, lambda, lower_tail = TRUE, log_p = FALSE)
```

Arguments

q	Vector of quantiles (values generally between 0 and 1).
alpha	Shape parameter `alpha` > 0. Can be a scalar or a vector. Default: 1.0.
beta	Shape parameter `beta` > 0. Can be a scalar or a vector. Default: 1.0.
lambda	Shape parameter `lambda` > 0 (exponent parameter). Can be a scalar or a vector. Default: 1.0.
lower_tail	Logical; if `TRUE` (default), probabilities are $P(X \leq q)$, otherwise, $P(X > q)$.
log_p	Logical; if `TRUE`, probabilities p are given as $\log(p)$. Default: `FALSE`.

Details

The Exponentiated Kumaraswamy (EKw) distribution is a special case of the five-parameter Generalized Kumaraswamy distribution (`pgkw`) obtained by setting parameters $\gamma = 1$ and $\delta = 0$.

The CDF of the GKw distribution is $F_{GKw}(q) = I_{y(q)}(\gamma, \delta + 1)$, where $y(q) = [1 - (1 - q^{\alpha})^{\beta}]^{\lambda}$ and $I_x(a, b)$ is the regularized incomplete beta function (`pbeta`). Setting $\gamma = 1$ and $\delta = 0$ gives $I_{y(q)}(1, 1)$. Since $I_x(1, 1) = x$, the CDF simplifies to $y(q)$:

$$F(q; \alpha, \beta, \lambda) = \left[1 - (1 - q^{\alpha})^{\beta}\right]^{\lambda}$$

for $0 < q < 1$. The implementation uses this closed-form expression for efficiency and handles `lower_tail` and `log_p` arguments appropriately.

Value

A vector of probabilities, $F(q)$, or their logarithms/complements depending on `lower_tail` and `log_p`. The length of the result is determined by the recycling rule applied to the arguments (`q`, `alpha`, `beta`, `lambda`). Returns `0` (or `-Inf` if `log_p = TRUE`) for `q <= 0` and `1` (or `0` if `log_p = TRUE`) for `q >= 1`. Returns `NaN` for invalid parameters.

Author(s)

Lopes, J. E.

References

Nadarajah, S., Cordeiro, G. M., & Ortega, E. M. (2012). The exponentiated Kumaraswamy distribution. *Journal of the Franklin Institute*, *349*(3),

Cordeiro, G. M., & de Castro, M. (2011). A new family of generalized distributions. *Journal of Statistical Computation and Simulation*,

Kumaraswamy, P. (1980). A generalized probability density function for double-bounded random processes. *Journal of Hydrology*, *46*(1-2), 79-88.

See Also

pgkw (parent distribution CDF), dekw, qekw, rekw (other EKw functions),

Examples

```
# Example values
q_vals <- c(0.2, 0.5, 0.8)
alpha_par <- 2.0
beta_par <- 3.0
lambda_par <- 1.5

# Calculate CDF P(X <= q)
probs <- pekw(q_vals, alpha_par, beta_par, lambda_par)
print(probs)

# Calculate upper tail P(X > q)
probs_upper <- pekw(q_vals, alpha_par, beta_par, lambda_par,
                    lower_tail = FALSE)
print(probs_upper)
# Check: probs + probs_upper should be 1
print(probs + probs_upper)

# Calculate log CDF
log_probs <- pekw(q_vals, alpha_par, beta_par, lambda_par, log_p = TRUE)
print(log_probs)
# Check: should match log(probs)
print(log(probs))

# Compare with pgkw setting gamma = 1, delta = 0
probs_gkw <- pgkw(q_vals, alpha_par, beta_par, gamma = 1.0, delta = 0.0,
                 lambda = lambda_par)
print(paste("Max difference:", max(abs(probs - probs_gkw)))) # Should be near zero

# Plot the CDF for different lambda values
curve_q <- seq(0.01, 0.99, length.out = 200)
curve_p1 <- pekw(curve_q, alpha = 2, beta = 3, lambda = 0.5)
curve_p2 <- pekw(curve_q, alpha = 2, beta = 3, lambda = 1.0) # standard Kw
curve_p3 <- pekw(curve_q, alpha = 2, beta = 3, lambda = 2.0)

plot(curve_q, curve_p2, type = "l", main = "EKw CDF Examples (alpha=2, beta=3)",
     xlab = "q", ylab = "F(q)", col = "red", ylim = c(0, 1))
lines(curve_q, curve_p1, col = "blue")
lines(curve_q, curve_p3, col = "green")
legend("bottomright", legend = c("lambda=0.5", "lambda=1.0 (Kw)", "lambda=2.0"),
       col = c("blue", "red", "green"), lty = 1, bty = "n")
```

---

pgkw *Generalized Kumaraswamy Distribution CDF*

Description

Computes the cumulative distribution function (CDF) for the five-parameter Generalized Kumaraswamy (GKw) distribution, defined on the interval (0, 1). Calculates $P(X \le q)$.

Usage

```
pgkw(q, alpha, beta, gamma, delta, lambda, lower_tail = TRUE, log_p = FALSE)
```

Arguments

`q`	Vector of quantiles (values generally between 0 and 1).
`alpha`	Shape parameter `alpha` > 0. Can be a scalar or a vector. Default: 1.0.
`beta`	Shape parameter `beta` > 0. Can be a scalar or a vector. Default: 1.0.
`gamma`	Shape parameter `gamma` > 0. Can be a scalar or a vector. Default: 1.0.
`delta`	Shape parameter `delta` >= 0. Can be a scalar or a vector. Default: 0.0.
`lambda`	Shape parameter `lambda` > 0. Can be a scalar or a vector. Default: 1.0.
`lower_tail`	Logical; if `TRUE` (default), probabilities are $P(X \le q)$, otherwise, $P(X > q)$.
`log_p`	Logical; if `TRUE`, probabilities p are given as $\log(p)$. Default: `FALSE`.

Details

The cumulative distribution function (CDF) of the Generalized Kumaraswamy (GKw) distribution with parameters `alpha` (α), `beta` (β), `gamma` (γ), `delta` (δ), and `lambda` (λ) is given by:

$$F(q; \alpha, \beta, \gamma, \delta, \lambda) = I_{x(q)}(\gamma, \delta + 1)$$

where $x(q) = [1 - (1 - q^\alpha)^\beta]^\lambda$ and $I_x(a, b)$ is the regularized incomplete beta function, defined as:

$$I_x(a, b) = \frac{B_x(a, b)}{B(a, b)} = \frac{\int_0^x t^{a-1}(1-t)^{b-1} dt}{\int_0^1 t^{a-1}(1-t)^{b-1} dt}$$

This corresponds to the `pbeta` function in R, such that $F(q; \alpha, \beta, \gamma, \delta, \lambda) = \texttt{pbeta}(x(q), \texttt{shape1} = \gamma, \texttt{shape2} = \delta + 1)$.

The GKw distribution includes several special cases, such as the Kumaraswamy, Beta, and Exponentiated Kumaraswamy distributions (see `dgkw` for details). The function utilizes numerical algorithms for computing the regularized incomplete beta function accurately, especially near the boundaries.

Value

A vector of probabilities, $F(q)$, or their logarithms if `log_p = TRUE`. The length of the result is determined by the recycling rule applied to the arguments (`q`, `alpha`, `beta`, `gamma`, `delta`, `lambda`). Returns `0` (or `-Inf` if `log_p = TRUE`) for `q <= 0` and `1` (or `0` if `log_p = TRUE`) for `q >= 1`. Returns `NaN` for invalid parameters.

Author(s)

Lopes, J. E.

References

Cordeiro, G. M., & de Castro, M. (2011). A new family of generalized distributions. *Journal of Statistical Computation and Simulation*

Kumaraswamy, P. (1980). A generalized probability density function for double-bounded random processes. *Journal of Hydrology*, *46*(1-2), 79-88.

See Also

dgkw, qgkw, rgkw, pbeta

Examples

```
# Simple CDF evaluation
prob <- pgkw(0.5, alpha = 2, beta = 3, gamma = 1, delta = 0, lambda = 1) # Kw case
print(prob)

# Upper tail probability P(X > q)
prob_upper <- pgkw(0.5, alpha = 2, beta = 3, gamma = 1, delta = 0, lambda = 1,
                 lower_tail = FALSE)
print(prob_upper)
# Check: prob + prob_upper should be 1
print(prob + prob_upper)

# Log probability
log_prob <- pgkw(0.5, alpha = 2, beta = 3, gamma = 1, delta = 0, lambda = 1,
                 log_p = TRUE)
print(log_prob)
# Check: exp(log_prob) should be prob
print(exp(log_prob))

# Use of vectorized parameters
q_vals <- c(0.2, 0.5, 0.8)
alphas_vec <- c(0.5, 1.0, 2.0)
betas_vec <- c(1.0, 2.0, 3.0)
# Vectorizes over q, alpha, beta
pgkw(q_vals, alpha = alphas_vec, beta = betas_vec, gamma = 1, delta = 0.5, lambda = 0.5)

# Plotting the CDF for special cases
x_seq <- seq(0.01, 0.99, by = 0.01)
# Standard Kumaraswamy CDF
cdf_kw <- pgkw(x_seq, alpha = 2, beta = 3, gamma = 1, delta = 0, lambda = 1)
# Beta distribution CDF equivalent (Beta(gamma, delta+1))
cdf_beta_equiv <- pgkw(x_seq, alpha = 1, beta = 1, gamma = 2, delta = 3, lambda = 1)
# Compare with stats::pbeta
cdf_beta_check <- stats::pbeta(x_seq, shape1 = 2, shape2 = 3 + 1)
# max(abs(cdf_beta_equiv - cdf_beta_check)) # Should be close to zero

plot(x_seq, cdf_kw, type = "l", ylim = c(0, 1),
     main = "GKw CDF Examples", ylab = "F(x)", xlab = "x", col = "blue")
lines(x_seq, cdf_beta_equiv, col = "red", lty = 2)
legend("bottomright", legend = c("Kw(2,3)", "Beta(2,4) equivalent"),
        col = c("blue", "red"), lty = c(1, 2), bty = "n")
```

---

pkkw *Cumulative Distribution Function (CDF) of the kkw Distribution*

Description

Computes the cumulative distribution function (CDF), $P(X \leq q)$, for the Kumaraswamy-Kumaraswamy (kkw) distribution with parameters `alpha` (α), `beta` (β), `delta` (δ), and `lambda` (λ). This distribution is defined on the interval (0, 1).

Usage

```
pkkw(q, alpha, beta, delta, lambda, lower_tail = TRUE, log_p = FALSE)
```

Arguments

`q`	Vector of quantiles (values generally between 0 and 1).
`alpha`	Shape parameter `alpha` > 0. Can be a scalar or a vector. Default: 1.0.
`beta`	Shape parameter `beta` > 0. Can be a scalar or a vector. Default: 1.0.
`delta`	Shape parameter `delta` >= 0. Can be a scalar or a vector. Default: 0.0.
`lambda`	Shape parameter `lambda` > 0. Can be a scalar or a vector. Default: 1.0.
`lower_tail`	Logical; if `TRUE` (default), probabilities are $P(X \leq q)$, otherwise, $P(X > q)$.
`log_p`	Logical; if `TRUE`, probabilities p are given as $\log(p)$. Default: `FALSE`.

Details

The Kumaraswamy-Kumaraswamy (kkw) distribution is a special case of the five-parameter Generalized Kumaraswamy distribution (`pgkw`) obtained by setting the shape parameter $\gamma = 1$.

The CDF of the GKw distribution is $F_{GKw}(q) = I_{y(q)}(\gamma, \delta + 1)$, where $y(q) = [1 - (1 - q^\alpha)^\beta]^\lambda$ and $I_x(a, b)$ is the regularized incomplete beta function (`pbeta`). Setting $\gamma = 1$ utilizes the property $I_x(1, b) = 1 - (1 - x)^b$, yielding the kkw CDF:

$$F(q; \alpha, \beta, \delta, \lambda) = 1 - \left\{1 - \left[1 - (1 - q^\alpha)^\beta\right]^\lambda\right\}^{\delta+1}$$

for $0 < q < 1$.

The implementation uses this closed-form expression for efficiency and handles `lower_tail` and `log_p` arguments appropriately.

Value

A vector of probabilities, $F(q)$, or their logarithms/complements depending on `lower_tail` and `log_p`. The length of the result is determined by the recycling rule applied to the arguments (`q`, `alpha`, `beta`, `delta`, `lambda`). Returns `0` (or `-Inf` if `log_p = TRUE`) for `q <= 0` and `1` (or `0` if `log_p = TRUE`) for `q >= 1`. Returns `NaN` for invalid parameters.

Author(s)

Lopes, J. E.

References

Cordeiro, G. M., & de Castro, M. (2011). A new family of generalized distributions. *Journal of Statistical Computation and Simulation*

Kumaraswamy, P. (1980). A generalized probability density function for double-bounded random processes. *Journal of Hydrology*, *46*(1-2), 79-88.

See Also

`pgkw` (parent distribution CDF), `dkkw`, `qkkw`, `rkkw`, `pbeta`

Examples

```
# Example values
q_vals <- c(0.2, 0.5, 0.8)
alpha_par <- 2.0
beta_par <- 3.0
delta_par <- 0.5
lambda_par <- 1.5

# Calculate CDF P(X <= q)
probs <- pkkw(q_vals, alpha_par, beta_par, delta_par, lambda_par)
print(probs)

# Calculate upper tail P(X > q)
probs_upper <- pkkw(q_vals, alpha_par, beta_par, delta_par, lambda_par,
                     lower_tail = FALSE)
print(probs_upper)
# Check: probs + probs_upper should be 1
print(probs + probs_upper)

# Calculate log CDF
log_probs <- pkkw(q_vals, alpha_par, beta_par, delta_par, lambda_par,
                   log_p = TRUE)
print(log_probs)
# Check: should match log(probs)
print(log(probs))

# Compare with pgkw setting gamma = 1
probs_gkw <- pgkw(q_vals, alpha_par, beta_par, gamma = 1.0,
                  delta_par, lambda_par)
print(paste("Max difference:", max(abs(probs - probs_gkw)))) # Should be near zero

# Plot the CDF
curve_q <- seq(0.01, 0.99, length.out = 200)
curve_p <- pkkw(curve_q, alpha_par, beta_par, delta_par, lambda_par)
plot(curve_q, curve_p, type = "l", main = "kkw CDF Example",
     xlab = "q", ylab = "F(q)", col = "blue", ylim = c(0, 1))
```

pkw *Cumulative Distribution Function (CDF) of the Kumaraswamy (Kw) Distribution*

Description

Computes the cumulative distribution function (CDF), $P(X \leq q)$, for the two-parameter Kumaraswamy (Kw) distribution with shape parameters `alpha` (α) and `beta` (β). This distribution is defined on the interval (0, 1).

Usage

```
pkw(q, alpha, beta, lower_tail = TRUE, log_p = FALSE)
```

Arguments

`q`	Vector of quantiles (values generally between 0 and 1).
`alpha`	Shape parameter `alpha` > 0. Can be a scalar or a vector. Default: 1.0.
`beta`	Shape parameter `beta` > 0. Can be a scalar or a vector. Default: 1.0.
`lower_tail`	Logical; if `TRUE` (default), probabilities are $P(X \leq q)$, otherwise, $P(X > q)$.
`log_p`	Logical; if `TRUE`, probabilities p are given as $\log(p)$. Default: `FALSE`.

Details

The cumulative distribution function (CDF) of the Kumaraswamy (Kw) distribution is given by:

$$F(x; \alpha, \beta) = 1 - (1 - x^\alpha)^\beta$$

for $0 < x < 1$, $\alpha > 0$, and $\beta > 0$.

The Kw distribution is a special case of several generalized distributions:

- Generalized Kumaraswamy (`pgkw`) with $\gamma = 1, \delta = 0, \lambda = 1$.
- Exponentiated Kumaraswamy (`pekw`) with $\lambda = 1$.
- Kumaraswamy-Kumaraswamy (`pkkw`) with $\delta = 0, \lambda = 1$.

The implementation uses the closed-form expression for efficiency.

Value

A vector of probabilities, $F(q)$, or their logarithms/complements depending on `lower_tail` and `log_p`. The length of the result is determined by the recycling rule applied to the arguments (`q`, `alpha`, `beta`). Returns `0` (or `-Inf` if `log_p = TRUE`) for `q <= 0` and `1` (or `0` if `log_p = TRUE`) for `q >= 1`. Returns `NaN` for invalid parameters.

Author(s)

Lopes, J. E.

References

Kumaraswamy, P. (1980). A generalized probability density function for double-bounded random processes. *Journal of Hydrology*, *46*(1-2), 79-88.

Jones, M. C. (2009). Kumaraswamy's distribution: A beta-type distribution with some tractability advantages. *Statistical Methodology*, *6*(1), 70-81.

See Also

`pgkw`, `pekw`, `pkkw` (related generalized CDFs), `dkw`, `qkw`, `rkw` (other Kw functions), `pbeta`

Examples

```
# Example values
q_vals <- c(0.2, 0.5, 0.8)
alpha_par <- 2.0
beta_par <- 3.0

# Calculate CDF P(X <= q) using pkw
probs <- pkw(q_vals, alpha_par, beta_par)
print(probs)

# Calculate upper tail P(X > q)
probs_upper <- pkw(q_vals, alpha_par, beta_par, lower_tail = FALSE)
print(probs_upper)
# Check: probs + probs_upper should be 1
print(probs + probs_upper)

# Calculate log CDF
log_probs <- pkw(q_vals, alpha_par, beta_par, log_p = TRUE)
print(log_probs)
# Check: should match log(probs)
print(log(probs))

# Compare with pgkw setting gamma = 1, delta = 0, lambda = 1
probs_gkw <- pgkw(q_vals, alpha_par, beta_par, gamma = 1.0, delta = 0.0,
                  lambda = 1.0)
print(paste("Max difference:", max(abs(probs - probs_gkw)))) # Should be near zero

# Plot the CDF for different shape parameter combinations
curve_q <- seq(0.001, 0.999, length.out = 200)
plot(curve_q, pkw(curve_q, alpha = 2, beta = 3), type = "l",
     main = "Kumaraswamy CDF Examples", xlab = "q", ylab = "F(q)",
     col = "blue", ylim = c(0, 1))
lines(curve_q, pkw(curve_q, alpha = 3, beta = 2), col = "red")
lines(curve_q, pkw(curve_q, alpha = 0.5, beta = 0.5), col = "green")
lines(curve_q, pkw(curve_q, alpha = 5, beta = 1), col = "purple")
lines(curve_q, pkw(curve_q, alpha = 1, beta = 3), col = "orange")
```

```
legend("bottomright", legend = c("a=2, b=3", "a=3, b=2", "a=0.5, b=0.5", "a=5, b=1", "a=1, b=3"),
       col = c("blue", "red", "green", "purple", "orange"), lty = 1, bty = "n", ncol = 2)
```

pmc *CDF of the McDonald (Mc)/Beta Power Distribution*

Description

Computes the cumulative distribution function (CDF), $F(q) = P(X \leq q)$, for the McDonald (Mc) distribution (also known as Beta Power) with parameters `gamma` (γ), `delta` (δ), and `lambda` (λ). This distribution is defined on the interval (0, 1) and is a special case of the Generalized Kumaraswamy (GKw) distribution where $\alpha = 1$ and $\beta = 1$.

Usage

```
pmc(q, gamma, delta, lambda, lower_tail = TRUE, log_p = FALSE)
```

Arguments

`q`	Vector of quantiles (values generally between 0 and 1).
`gamma`	Shape parameter `gamma` > 0. Can be a scalar or a vector. Default: 1.0.
`delta`	Shape parameter `delta` >= 0. Can be a scalar or a vector. Default: 0.0.
`lambda`	Shape parameter `lambda` > 0. Can be a scalar or a vector. Default: 1.0.
`lower_tail`	Logical; if `TRUE` (default), probabilities are $P(X \leq q)$, otherwise, $P(X > q)$.
`log_p`	Logical; if `TRUE`, probabilities p are given as $\log(p)$. Default: `FALSE`.

Details

The McDonald (Mc) distribution is a special case of the five-parameter Generalized Kumaraswamy (GKw) distribution (`pgkw`) obtained by setting parameters $\alpha = 1$ and $\beta = 1$.

The CDF of the GKw distribution is $F_{GKw}(q) = I_{y(q)}(\gamma, \delta+1)$, where $y(q) = [1-(1-q^{\alpha})^{\beta}]^{\lambda}$ and $I_x(a, b)$ is the regularized incomplete beta function (`pbeta`). Setting $\alpha = 1$ and $\beta = 1$ simplifies $y(q)$ to q^{λ}, yielding the Mc CDF:

$$F(q; \gamma, \delta, \lambda) = I_{q^{\lambda}}(\gamma, \delta + 1)$$

This is evaluated using the `pbeta` function as `pbeta(q^lambda, shape1 = gamma, shape2 = delta + 1)`.

Value

A vector of probabilities, $F(q)$, or their logarithms/complements depending on `lower_tail` and `log_p`. The length of the result is determined by the recycling rule applied to the arguments (`q`, `gamma`, `delta`, `lambda`). Returns `0` (or `-Inf` if `log_p = TRUE`) for `q <= 0` and `1` (or `0` if `log_p = TRUE`) for `q >= 1`. Returns `NaN` for invalid parameters.

Author(s)

Lopes, J. E.

References

McDonald, J. B. (1984). Some generalized functions for the size distribution of income. *Econometrica*, 52(3), 647-663.

Cordeiro, G. M., & de Castro, M. (2011). A new family of generalized distributions. *Journal of Statistical Computation and Simulation*,

Kumaraswamy, P. (1980). A generalized probability density function for double-bounded random processes. *Journal of Hydrology*, *46*(1-2), 79-88.

See Also

pgkw (parent distribution CDF), dmc, qmc, rmc (other Mc functions), pbeta

Examples

```
# Example values
q_vals <- c(0.2, 0.5, 0.8)
gamma_par <- 2.0
delta_par <- 1.5
lambda_par <- 1.0 # Equivalent to Beta(gamma, delta+1)

# Calculate CDF P(X <= q) using pmc
probs <- pmc(q_vals, gamma_par, delta_par, lambda_par)
print(probs)
# Compare with Beta CDF
print(stats::pbeta(q_vals, shape1 = gamma_par, shape2 = delta_par + 1))

# Calculate upper tail P(X > q)
probs_upper <- pmc(q_vals, gamma_par, delta_par, lambda_par,
                   lower_tail = FALSE)
print(probs_upper)
# Check: probs + probs_upper should be 1
print(probs + probs_upper)

# Calculate log CDF
log_probs <- pmc(q_vals, gamma_par, delta_par, lambda_par, log_p = TRUE)
print(log_probs)
# Check: should match log(probs)
print(log(probs))

# Compare with pgkw setting alpha = 1, beta = 1
probs_gkw <- pgkw(q_vals, alpha = 1.0, beta = 1.0, gamma = gamma_par,
                  delta = delta_par, lambda = lambda_par)
print(paste("Max difference:", max(abs(probs - probs_gkw)))) # Should be near zero

# Plot the CDF for different lambda values
curve_q <- seq(0.01, 0.99, length.out = 200)
curve_p1 <- pmc(curve_q, gamma = 2, delta = 3, lambda = 0.5)
```

```
curve_p2 <- pmc(curve_q, gamma = 2, delta = 3, lambda = 1.0) # Beta(2, 4)
curve_p3 <- pmc(curve_q, gamma = 2, delta = 3, lambda = 2.0)

plot(curve_q, curve_p2, type = "l", main = "Mc/Beta Power CDF (gamma=2, delta=3)",
     xlab = "q", ylab = "F(q)", col = "red", ylim = c(0, 1))
lines(curve_q, curve_p1, col = "blue")
lines(curve_q, curve_p3, col = "green")
legend("bottomright", legend = c("lambda=0.5", "lambda=1.0 (Beta)", "lambda=2.0"),
       col = c("blue", "red", "green"), lty = 1, bty = "n")
```

qbeta_ *Quantile Function of the Beta Distribution (gamma, delta+1 Parameterization)*

Description

Computes the quantile function (inverse CDF) for the standard Beta distribution, using a parameterization common in generalized distribution families. It finds the value q such that $P(X \leq q) = p$. The distribution is parameterized by `gamma` (γ) and `delta` (δ), corresponding to the standard Beta distribution with shape parameters `shape1 = gamma` and `shape2 = delta + 1`.

Usage

```
qbeta_(p, gamma, delta, lower_tail = TRUE, log_p = FALSE)
```

Arguments

`p`	Vector of probabilities (values between 0 and 1).
`gamma`	First shape parameter (`shape1`), $\gamma > 0$. Can be a scalar or a vector. Default: 1.0.
`delta`	Second shape parameter is `delta + 1` (`shape2`), requires $\delta \geq 0$ so that `shape2 >= 1`. Can be a scalar or a vector. Default: 0.0 (leading to `shape2 = 1`).
`lower_tail`	Logical; if `TRUE` (default), probabilities are $p = P(X \leq q)$, otherwise, probabilities are $p = P(X > q)$.
`log_p`	Logical; if `TRUE`, probabilities `p` are given as $\log(p)$. Default: `FALSE`.

Details

This function computes the quantiles of a Beta distribution with parameters `shape1 = gamma` and `shape2 = delta + 1`. It is equivalent to calling `stats::qbeta(p, shape1 = gamma, shape2 = delta + 1, lower.tail = lower_tail, log.p = log_p)`.

This distribution arises as a special case of the five-parameter Generalized Kumaraswamy (GKw) distribution (`qgkw`) obtained by setting $\alpha = 1$, $\beta = 1$, and $\lambda = 1$. It is therefore also equivalent to the McDonald (Mc)/Beta Power distribution (`qmc`) with $\lambda = 1$.

The function likely calls R's underlying `qbeta` function but ensures consistent parameter recycling and handling within the C++ environment, matching the style of other functions in the related families. Boundary conditions (p=0, p=1) are handled explicitly.

Value

A vector of quantiles corresponding to the given probabilities `p`. The length of the result is determined by the recycling rule applied to the arguments (`p`, `gamma`, `delta`). Returns:

- `0` for `p = 0` (or `p = -Inf` if `log_p = TRUE`, when `lower_tail = TRUE`).
- `1` for `p = 1` (or `p = 0` if `log_p = TRUE`, when `lower_tail = TRUE`).
- `NaN` for `p < 0` or `p > 1` (or corresponding log scale).
- `NaN` for invalid parameters (e.g., `gamma <= 0`, `delta < 0`).

Boundary return values are adjusted accordingly for `lower_tail = FALSE`.

Author(s)

Lopes, J. E.

References

Johnson, N. L., Kotz, S., & Balakrishnan, N. (1995). *Continuous Univariate Distributions, Volume 2* (2nd ed.). Wiley.

Cordeiro, G. M., & de Castro, M. (2011). A new family of generalized distributions. *Journal of Statistical Computation and Simulation*,

See Also

`qbeta` (standard R implementation), `qgkw` (parent distribution quantile function), `qmc` (McDonald/Beta Power quantile function), `dbeta_`, `pbeta_`, `rbeta_` (other functions for this parameterization, if they exist).

Examples

```
# Example values
p_vals <- c(0.1, 0.5, 0.9)
gamma_par <- 2.0 # Corresponds to shape1
delta_par <- 3.0 # Corresponds to shape2 - 1
shape1 <- gamma_par
shape2 <- delta_par + 1

# Calculate quantiles using qbeta_
quantiles <- qbeta_(p_vals, gamma_par, delta_par)
print(quantiles)

# Compare with stats::qbeta
quantiles_stats <- stats::qbeta(p_vals, shape1 = shape1, shape2 = shape2)
print(paste("Max difference vs stats::qbeta:", max(abs(quantiles - quantiles_stats))))
```

```
# Compare with qgkw setting alpha=1, beta=1, lambda=1
quantiles_gkw <- qgkw(p_vals, alpha = 1.0, beta = 1.0, gamma = gamma_par,
                      delta = delta_par, lambda = 1.0)
print(paste("Max difference vs qgkw:", max(abs(quantiles - quantiles_gkw))))

# Compare with qmc setting lambda=1
quantiles_mc <- qmc(p_vals, gamma = gamma_par, delta = delta_par, lambda = 1.0)
print(paste("Max difference vs qmc:", max(abs(quantiles - quantiles_mc))))

# Calculate quantiles for upper tail
quantiles_upper <- qbeta_(p_vals, gamma_par, delta_par, lower_tail = FALSE)
print(quantiles_upper)
print(stats::qbeta(p_vals, shape1, shape2, lower.tail = FALSE))

# Calculate quantiles from log probabilities
log_p_vals <- log(p_vals)
quantiles_logp <- qbeta_(log_p_vals, gamma_par, delta_par, log_p = TRUE)
print(quantiles_logp)
print(stats::qbeta(log_p_vals, shape1, shape2, log.p = TRUE))

# Verify inverse relationship with pbeta_
p_check <- 0.75
q_calc <- qbeta_(p_check, gamma_par, delta_par)
p_recalc <- pbeta_(q_calc, gamma_par, delta_par)
print(paste("Original p:", p_check, " Recalculated p:", p_recalc))
# abs(p_check - p_recalc) < 1e-9 # Should be TRUE

# Boundary conditions
print(qbeta_(c(0, 1), gamma_par, delta_par)) # Should be 0, 1
print(qbeta_(c(-Inf, 0), gamma_par, delta_par, log_p = TRUE)) # Should be 0, 1
```

qbkw — *Quantile Function of the Beta-Kumaraswamy (BKw) Distribution*

Description

Computes the quantile function (inverse CDF) for the Beta-Kumaraswamy (BKw) distribution with parameters `alpha` (α), `beta` (β), `gamma` (γ), and `delta` (δ). It finds the value `q` such that $P(X \leq q) = p$. This distribution is a special case of the Generalized Kumaraswamy (GKw) distribution where the parameter $\lambda = 1$.

Usage

```
qbkw(p, alpha, beta, gamma, delta, lower_tail = TRUE, log_p = FALSE)
```

Arguments

`p`	Vector of probabilities (values between 0 and 1).
`alpha`	Shape parameter `alpha` > 0. Can be a scalar or a vector. Default: 1.0.
`beta`	Shape parameter `beta` > 0. Can be a scalar or a vector. Default: 1.0.
`gamma`	Shape parameter `gamma` > 0. Can be a scalar or a vector. Default: 1.0.
`delta`	Shape parameter `delta` >= 0. Can be a scalar or a vector. Default: 0.0.
`lower_tail`	Logical; if `TRUE` (default), probabilities are $p = P(X \leq q)$, otherwise, probabilities are $p = P(X > q)$.
`log_p`	Logical; if `TRUE`, probabilities `p` are given as $\log(p)$. Default: `FALSE`.

Details

The quantile function $Q(p)$ is the inverse of the CDF $F(q)$. The CDF for the BKw ($\lambda = 1$) distribution is $F(q) = I_{y(q)}(\gamma, \delta + 1)$, where $y(q) = 1 - (1 - q^\alpha)^\beta$ and $I_z(a, b)$ is the regularized incomplete beta function (see `pbkw`).

To find the quantile q, we first invert the outer Beta part: let $y = I_p^{-1}(\gamma, \delta + 1)$, where $I_p^{-1}(a, b)$ is the inverse of the regularized incomplete beta function, computed via `qbeta`. Then, we invert the inner Kumaraswamy part: $y = 1 - (1 - q^\alpha)^\beta$, which leads to $q = \{1 - (1 - y)^{1/\beta}\}^{1/\alpha}$. Substituting y gives the quantile function:

$$Q(p) = \left\{1 - \left[1 - I_p^{-1}(\gamma, \delta + 1)\right]^{1/\beta}\right\}^{1/\alpha}$$

The function uses this formula, calculating $I_p^{-1}(\gamma, \delta + 1)$ via `qbeta(p, gamma, delta + 1, ...)` while respecting the `lower_tail` and `log_p` arguments.

Value

A vector of quantiles corresponding to the given probabilities `p`. The length of the result is determined by the recycling rule applied to the arguments (`p`, `alpha`, `beta`, `gamma`, `delta`). Returns:

- `0` for `p = 0` (or `p = -Inf` if `log_p = TRUE`, when `lower_tail = TRUE`).
- `1` for `p = 1` (or `p = 0` if `log_p = TRUE`, when `lower_tail = TRUE`).
- `NaN` for `p < 0` or `p > 1` (or corresponding log scale).
- `NaN` for invalid parameters (e.g., `alpha <= 0`, `beta <= 0`, `gamma <= 0`, `delta < 0`).

Boundary return values are adjusted accordingly for `lower_tail = FALSE`.

Author(s)

Lopes, J. E.

References

Cordeiro, G. M., & de Castro, M. (2011). A new family of generalized distributions. *Journal of Statistical Computation and Simulation*

Kumaraswamy, P. (1980). A generalized probability density function for double-bounded random processes. *Journal of Hydrology*, *46*(1-2), 79-88.

See Also

qgkw (parent distribution quantile function), dbkw, pbkw, rbkw (other BKw functions), qbeta

Examples

```
# Example values
p_vals <- c(0.1, 0.5, 0.9)
alpha_par <- 2.0
beta_par <- 1.5
gamma_par <- 1.0
delta_par <- 0.5

# Calculate quantiles
quantiles <- qbkw(p_vals, alpha_par, beta_par, gamma_par, delta_par)
print(quantiles)

# Calculate quantiles for upper tail probabilities P(X > q) = p
quantiles_upper <- qbkw(p_vals, alpha_par, beta_par, gamma_par, delta_par,
                        lower_tail = FALSE)
print(quantiles_upper)
# Check: qbkw(p, ..., lt=F) == qbkw(1-p, ..., lt=T)
print(qbkw(1 - p_vals, alpha_par, beta_par, gamma_par, delta_par))

# Calculate quantiles from log probabilities
log_p_vals <- log(p_vals)
quantiles_logp <- qbkw(log_p_vals, alpha_par, beta_par, gamma_par, delta_par,
                       log_p = TRUE)
print(quantiles_logp)
# Check: should match original quantiles
print(quantiles)

# Compare with qgkw setting lambda = 1
quantiles_gkw <- qgkw(p_vals, alpha_par, beta_par, gamma = gamma_par,
                      delta = delta_par, lambda = 1.0)
print(paste("Max difference:", max(abs(quantiles - quantiles_gkw)))) # Should be near zero

# Verify inverse relationship with pbkw
p_check <- 0.75
q_calc <- qbkw(p_check, alpha_par, beta_par, gamma_par, delta_par)
p_recalc <- pbkw(q_calc, alpha_par, beta_par, gamma_par, delta_par)
print(paste("Original p:", p_check, " Recalculated p:", p_recalc))
# abs(p_check - p_recalc) < 1e-9 # Should be TRUE

# Boundary conditions
print(qbkw(c(0, 1), alpha_par, beta_par, gamma_par, delta_par)) # Should be 0, 1
print(qbkw(c(-Inf, 0), alpha_par, beta_par, gamma_par, delta_par, log_p = TRUE)) # Should be 0, 1
```

qekw *Quantile Function of the Exponentiated Kumaraswamy (EKw) Distribution*

Description

Computes the quantile function (inverse CDF) for the Exponentiated Kumaraswamy (EKw) distribution with parameters `alpha` (α), `beta` (β), and `lambda` (λ). It finds the value `q` such that $P(X \leq q) = p$. This distribution is a special case of the Generalized Kumaraswamy (GKw) distribution where $\gamma = 1$ and $\delta = 0$.

Usage

```
qekw(p, alpha, beta, lambda, lower_tail = TRUE, log_p = FALSE)
```

Arguments

`p`	Vector of probabilities (values between 0 and 1).
`alpha`	Shape parameter `alpha` > 0. Can be a scalar or a vector. Default: 1.0.
`beta`	Shape parameter `beta` > 0. Can be a scalar or a vector. Default: 1.0.
`lambda`	Shape parameter `lambda` > 0 (exponent parameter). Can be a scalar or a vector. Default: 1.0.
`lower_tail`	Logical; if `TRUE` (default), probabilities are $p = P(X \leq q)$, otherwise, probabilities are $p = P(X > q)$.
`log_p`	Logical; if `TRUE`, probabilities `p` are given as $\log(p)$. Default: `FALSE`.

Details

The quantile function $Q(p)$ is the inverse of the CDF $F(q)$. The CDF for the EKw ($\gamma = 1, \delta = 0$) distribution is $F(q) = [1 - (1 - q^\alpha)^\beta]^\lambda$ (see pekw). Inverting this equation for q yields the quantile function:

$$Q(p) = \left\{1 - \left[1 - p^{1/\lambda}\right]^{1/\beta}\right\}^{1/\alpha}$$

The function uses this closed-form expression and correctly handles the `lower_tail` and `log_p` arguments by transforming `p` appropriately before applying the formula. This is equivalent to the general GKw quantile function (qgkw) evaluated with $\gamma = 1, \delta = 0$.

Value

A vector of quantiles corresponding to the given probabilities `p`. The length of the result is determined by the recycling rule applied to the arguments (`p`, `alpha`, `beta`, `lambda`). Returns:

- `0` for `p = 0` (or `p = -Inf` if `log_p = TRUE`, when `lower_tail = TRUE`).
- `1` for `p = 1` (or `p = 0` if `log_p = TRUE`, when `lower_tail = TRUE`).
- `NaN` for `p < 0` or `p > 1` (or corresponding log scale).
- `NaN` for invalid parameters (e.g., `alpha <= 0`, `beta <= 0`, `lambda <= 0`).

Boundary return values are adjusted accordingly for `lower_tail = FALSE`.

Author(s)

Lopes, J. E.

References

Nadarajah, S., Cordeiro, G. M., & Ortega, E. M. (2012). The exponentiated Kumaraswamy distribution. *Journal of the Franklin Institute*, *349*(3),

Cordeiro, G. M., & de Castro, M. (2011). A new family of generalized distributions. *Journal of Statistical Computation and Simulation*,

Kumaraswamy, P. (1980). A generalized probability density function for double-bounded random processes. *Journal of Hydrology*, *46*(1-2), 79-88.

See Also

`qgkw` (parent distribution quantile function), `dekw`, `pekw`, `rekw` (other EKw functions), `qunif`

Examples

```
# Example values
p_vals <- c(0.1, 0.5, 0.9)
alpha_par <- 2.0
beta_par <- 3.0
lambda_par <- 1.5

# Calculate quantiles
quantiles <- qekw(p_vals, alpha_par, beta_par, lambda_par)
print(quantiles)

# Calculate quantiles for upper tail probabilities P(X > q) = p
quantiles_upper <- qekw(p_vals, alpha_par, beta_par, lambda_par,
                        lower_tail = FALSE)
print(quantiles_upper)
# Check: qekw(p, ..., lt=F) == qekw(1-p, ..., lt=T)
print(qekw(1 - p_vals, alpha_par, beta_par, lambda_par))

# Calculate quantiles from log probabilities
log_p_vals <- log(p_vals)
quantiles_logp <- qekw(log_p_vals, alpha_par, beta_par, lambda_par,
                       log_p = TRUE)
print(quantiles_logp)
# Check: should match original quantiles
print(quantiles)

# Compare with qgkw setting gamma = 1, delta = 0
quantiles_gkw <- qgkw(p_vals, alpha = alpha_par, beta = beta_par,
                      gamma = 1.0, delta = 0.0, lambda = lambda_par)
print(paste("Max difference:", max(abs(quantiles - quantiles_gkw)))) # Should be near zero

# Verify inverse relationship with pekw
p_check <- 0.75
q_calc <- qekw(p_check, alpha_par, beta_par, lambda_par)
```

```
p_recalc <- pekw(q_calc, alpha_par, beta_par, lambda_par)
print(paste("Original p:", p_check, " Recalculated p:", p_recalc))
# abs(p_check - p_recalc) < 1e-9 # Should be TRUE

# Boundary conditions
print(qekw(c(0, 1), alpha_par, beta_par, lambda_par)) # Should be 0, 1
print(qekw(c(-Inf, 0), alpha_par, beta_par, lambda_par, log_p = TRUE)) # Should be 0, 1
```

qgkw *Generalized Kumaraswamy Distribution Quantile Function*

Description

Computes the quantile function (inverse CDF) for the five-parameter Generalized Kumaraswamy (GKw) distribution. Finds the value x such that $P(X \leq x) = p$, where X follows the GKw distribution.

Usage

```
qgkw(p, alpha, beta, gamma, delta, lambda, lower_tail = TRUE, log_p = FALSE)
```

Arguments

p	Vector of probabilities (values between 0 and 1).
alpha	Shape parameter alpha > 0. Can be a scalar or a vector. Default: 1.0.
beta	Shape parameter beta > 0. Can be a scalar or a vector. Default: 1.0.
gamma	Shape parameter gamma > 0. Can be a scalar or a vector. Default: 1.0.
delta	Shape parameter delta >= 0. Can be a scalar or a vector. Default: 0.0.
lambda	Shape parameter lambda > 0. Can be a scalar or a vector. Default: 1.0.
lower_tail	Logical; if TRUE (default), probabilities are $P(X \leq x)$, otherwise, $P(X > x)$.
log_p	Logical; if TRUE, probabilities p are given as $\log(p)$. Default: FALSE.

Details

The quantile function $Q(p)$ is the inverse of the CDF $F(x)$. Given $F(x) = I_{y(x)}(\gamma, \delta + 1)$ where $y(x) = [1 - (1 - x^{\alpha})^{\beta}]^{\lambda}$, the quantile function is:

$$Q(p) = x = \left\{1 - \left[1 - \left(I_p^{-1}(\gamma, \delta + 1)\right)^{1/\lambda}\right]^{1/\beta}\right\}^{1/\alpha}$$

where $I_p^{-1}(a, b)$ is the inverse of the regularized incomplete beta function, which corresponds to the quantile function of the Beta distribution, qbeta.

The computation proceeds as follows:

1. Calculate `y = stats::qbeta(p, shape1 = gamma, shape2 = delta + 1, lower.tail = lower_tail, log.p = log_p)`.
2. Calculate $v = y^{1/\lambda}$.
3. Calculate $w = (1 - v)^{1/\beta}$. Note: Requires $v \leq 1$.
4. Calculate $q = (1 - w)^{1/\alpha}$. Note: Requires $w \leq 1$.

Numerical stability is maintained by handling boundary cases (`p = 0`, `p = 1`) directly and checking intermediate results (e.g., ensuring arguments to powers are non-negative).

Value

A vector of quantiles corresponding to the given probabilities `p`. The length of the result is determined by the recycling rule applied to the arguments (`p`, `alpha`, `beta`, `gamma`, `delta`, `lambda`). Returns:

- `0` for `p = 0` (or `p = -Inf` if `log_p = TRUE`).
- `1` for `p = 1` (or `p = 0` if `log_p = TRUE`).
- `NaN` for `p < 0` or `p > 1` (or corresponding log scale).
- `NaN` for invalid parameters (e.g., `alpha <= 0`, `beta <= 0`, `gamma <= 0`, `delta < 0`, `lambda <= 0`).

Author(s)

Lopes, J. E.

References

Cordeiro, G. M., & de Castro, M. (2011). A new family of generalized distributions. *Journal of Statistical Computation and Simulation*

Kumaraswamy, P. (1980). A generalized probability density function for double-bounded random processes. *Journal of Hydrology*, *46*(1-2), 79-88.

See Also

dgkw, pgkw, rgkw, qbeta

Examples

```
# Basic quantile calculation (median)
median_val <- qgkw(0.5, alpha = 2, beta = 3, gamma = 1, delta = 0, lambda = 1)
print(median_val)

# Computing multiple quantiles
probs <- c(0.01, 0.1, 0.25, 0.5, 0.75, 0.9, 0.99)
quantiles <- qgkw(probs, alpha = 2, beta = 3, gamma = 1, delta = 0, lambda = 1)
print(quantiles)

# Upper tail quantile (e.g., find x such that P(X > x) = 0.1, which is 90th percentile)
q90 <- qgkw(0.1, alpha = 2, beta = 3, gamma = 1, delta = 0, lambda = 1,
            lower_tail = FALSE)
```

```
print(q90)
# Check: should match quantile for p = 0.9 with lower_tail = TRUE
print(qgkw(0.9, alpha = 2, beta = 3, gamma = 1, delta = 0, lambda = 1))

# Log probabilities
median_logp <- qgkw(log(0.5), alpha = 2, beta = 3, gamma = 1, delta = 0, lambda = 1,
                    log_p = TRUE)
print(median_logp) # Should match median_val

# Vectorized parameters
alphas_vec <- c(0.5, 1.0, 2.0)
betas_vec <- c(1.0, 2.0, 3.0)
# Get median for 3 different GKw distributions
medians_vec <- qgkw(0.5, alpha = alphas_vec, beta = betas_vec, gamma = 1, delta = 0, lambda = 1)
print(medians_vec)

# Verify inverse relationship with pgkw
p_val <- 0.75
x_val <- qgkw(p_val, alpha = 2, beta = 3, gamma = 1, delta = 0, lambda = 1)
p_check <- pgkw(x_val, alpha = 2, beta = 3, gamma = 1, delta = 0, lambda = 1)
print(paste("Calculated p:", p_check, " (Expected:", p_val, ")"))
```

qkkw — *Quantile Function of the Kumaraswamy-Kumaraswamy (kkw) Distribution*

Description

Computes the quantile function (inverse CDF) for the Kumaraswamy-Kumaraswamy (kkw) distribution with parameters `alpha` (α), `beta` (β), `delta` (δ), and `lambda` (λ). It finds the value `q` such that $P(X \leq q) = p$. This distribution is a special case of the Generalized Kumaraswamy (GKw) distribution where the parameter $\gamma = 1$.

Usage

```
qkkw(p, alpha, beta, delta, lambda, lower_tail = TRUE, log_p = FALSE)
```

Arguments

`p`	Vector of probabilities (values between 0 and 1).
`alpha`	Shape parameter `alpha` > 0. Can be a scalar or a vector. Default: 1.0.
`beta`	Shape parameter `beta` > 0. Can be a scalar or a vector. Default: 1.0.
`delta`	Shape parameter `delta` >= 0. Can be a scalar or a vector. Default: 0.0.
`lambda`	Shape parameter `lambda` > 0. Can be a scalar or a vector. Default: 1.0.
`lower_tail`	Logical; if `TRUE` (default), probabilities are $p = P(X \leq q)$, otherwise, probabilities are $p = P(X > q)$.
`log_p`	Logical; if `TRUE`, probabilities `p` are given as $\log(p)$. Default: `FALSE`.

Details

The quantile function $Q(p)$ is the inverse of the CDF $F(q)$. The CDF for the kkw ($\gamma = 1$) distribution is (see pkkw):

$$F(q) = 1 - \left\{1 - \left[1 - (1 - q^\alpha)^\beta\right]^\lambda\right\}^{\delta+1}$$

Inverting this equation for q yields the quantile function:

$$Q(p) = \left[1 - \left\{1 - \left[1 - (1 - p)^{1/(\delta+1)}\right]^{1/\lambda}\right\}^{1/\beta}\right]^{1/\alpha}$$

The function uses this closed-form expression and correctly handles the `lower_tail` and `log_p` arguments by transforming `p` appropriately before applying the formula.

Value

A vector of quantiles corresponding to the given probabilities `p`. The length of the result is determined by the recycling rule applied to the arguments (`p`, `alpha`, `beta`, `delta`, `lambda`). Returns:

- `0` for `p = 0` (or `p = -Inf` if `log_p = TRUE`, when `lower_tail = TRUE`).
- `1` for `p = 1` (or `p = 0` if `log_p = TRUE`, when `lower_tail = TRUE`).
- `NaN` for `p < 0` or `p > 1` (or corresponding log scale).
- `NaN` for invalid parameters (e.g., `alpha <= 0`, `beta <= 0`, `delta < 0`, `lambda <= 0`).

Boundary return values are adjusted accordingly for `lower_tail = FALSE`.

Author(s)

Lopes, J. E.

References

Cordeiro, G. M., & de Castro, M. (2011). A new family of generalized distributions. *Journal of Statistical Computation and Simulation*,

Kumaraswamy, P. (1980). A generalized probability density function for double-bounded random processes. *Journal of Hydrology*, *46*(1-2), 79-88.

See Also

qgkw (parent distribution quantile function), dkkw, pkkw, rkkw, qbeta

Examples

```
# Example values
p_vals <- c(0.1, 0.5, 0.9)
alpha_par <- 2.0
beta_par <- 3.0
delta_par <- 0.5
lambda_par <- 1.5
```

```
# Calculate quantiles
quantiles <- qkkw(p_vals, alpha_par, beta_par, delta_par, lambda_par)
print(quantiles)

# Calculate quantiles for upper tail probabilities P(X > q) = p
# e.g., for p=0.1, find q such that P(X > q) = 0.1 (90th percentile)
quantiles_upper <- qkkw(p_vals, alpha_par, beta_par, delta_par, lambda_par,
                         lower_tail = FALSE)
print(quantiles_upper)
# Check: qkkw(p, ..., lt=F) == qkkw(1-p, ..., lt=T)
print(qkkw(1 - p_vals, alpha_par, beta_par, delta_par, lambda_par))

# Calculate quantiles from log probabilities
log_p_vals <- log(p_vals)
quantiles_logp <- qkkw(log_p_vals, alpha_par, beta_par, delta_par, lambda_par,
                        log_p = TRUE)
print(quantiles_logp)
# Check: should match original quantiles
print(quantiles)

# Compare with qgkw setting gamma = 1
quantiles_gkw <- qgkw(p_vals, alpha_par, beta_par, gamma = 1.0,
                      delta_par, lambda_par)
print(paste("Max difference:", max(abs(quantiles - quantiles_gkw)))) # Should be near zero

# Verify inverse relationship with pkkw
p_check <- 0.75
q_calc <- qkkw(p_check, alpha_par, beta_par, delta_par, lambda_par)
p_recalc <- pkkw(q_calc, alpha_par, beta_par, delta_par, lambda_par)
print(paste("Original p:", p_check, " Recalculated p:", p_recalc))
# abs(p_check - p_recalc) < 1e-9 # Should be TRUE

# Boundary conditions
print(qkkw(c(0, 1), alpha_par, beta_par, delta_par, lambda_par)) # Should be 0, 1
print(qkkw(c(-Inf, 0), alpha_par, beta_par, delta_par, lambda_par, log_p = TRUE)) # Should be 0, 1
```

---

qkw *Quantile Function of the Kumaraswamy (Kw) Distribution*

Description

Computes the quantile function (inverse CDF) for the two-parameter Kumaraswamy (Kw) distribution with shape parameters `alpha` (α) and `beta` (β). It finds the value `q` such that $P(X \leq q) = p$.

Usage

```
qkw(p, alpha, beta, lower_tail = TRUE, log_p = FALSE)
```

Arguments

`p`	Vector of probabilities (values between 0 and 1).
`alpha`	Shape parameter `alpha` > 0. Can be a scalar or a vector. Default: 1.0.
`beta`	Shape parameter `beta` > 0. Can be a scalar or a vector. Default: 1.0.
`lower_tail`	Logical; if `TRUE` (default), probabilities are $p = P(X \le q)$, otherwise, probabilities are $p = P(X > q)$.
`log_p`	Logical; if `TRUE`, probabilities `p` are given as $\log(p)$. Default: `FALSE`.

Details

The quantile function $Q(p)$ is the inverse of the CDF $F(q)$. The CDF for the Kumaraswamy distribution is $F(q) = 1 - (1 - q^\alpha)^\beta$ (see `pkw`). Inverting this equation for q yields the quantile function:

$$Q(p) = \left\{1 - (1 - p)^{1/\beta}\right\}^{1/\alpha}$$

The function uses this closed-form expression and correctly handles the `lower_tail` and `log_p` arguments by transforming `p` appropriately before applying the formula. This is equivalent to the general GKw quantile function (`qgkw`) evaluated with $\gamma = 1, \delta = 0, \lambda = 1$.

Value

A vector of quantiles corresponding to the given probabilities `p`. The length of the result is determined by the recycling rule applied to the arguments (`p`, `alpha`, `beta`). Returns:

- `0` for `p = 0` (or `p = -Inf` if `log_p = TRUE`, when `lower_tail = TRUE`).
- `1` for `p = 1` (or `p = 0` if `log_p = TRUE`, when `lower_tail = TRUE`).
- `NaN` for `p < 0` or `p > 1` (or corresponding log scale).
- `NaN` for invalid parameters (e.g., `alpha <= 0`, `beta <= 0`).

Boundary return values are adjusted accordingly for `lower_tail = FALSE`.

Author(s)

Lopes, J. E.

References

Kumaraswamy, P. (1980). A generalized probability density function for double-bounded random processes. *Journal of Hydrology*, *46*(1-2), 79-88.

Jones, M. C. (2009). Kumaraswamy's distribution: A beta-type distribution with some tractability advantages. *Statistical Methodology*, *6*(1), 70-81.

See Also

`qgkw` (parent distribution quantile function), `dkw`, `pkw`, `rkw` (other Kw functions), `qbeta`, `qunif`

Examples

```
# Example values
p_vals <- c(0.1, 0.5, 0.9)
alpha_par <- 2.0
beta_par <- 3.0

# Calculate quantiles using qkw
quantiles <- qkw(p_vals, alpha_par, beta_par)
print(quantiles)

# Calculate quantiles for upper tail probabilities P(X > q) = p
quantiles_upper <- qkw(p_vals, alpha_par, beta_par, lower_tail = FALSE)
print(quantiles_upper)

# Calculate quantiles from log probabilities
log_p_vals <- log(p_vals)
quantiles_logp <- qkw(log_p_vals, alpha_par, beta_par, log_p = TRUE)
print(quantiles_logp)
# Check: should match original quantiles
print(quantiles)

# Compare with qgkw setting gamma = 1, delta = 0, lambda = 1
quantiles_gkw <- qgkw(p_vals, alpha = alpha_par, beta = beta_par,
                     gamma = 1.0, delta = 0.0, lambda = 1.0)
print(paste("Max difference:", max(abs(quantiles - quantiles_gkw)))) # Should be near zero

# Verify inverse relationship with pkw
p_check <- 0.75
q_calc <- qkw(p_check, alpha_par, beta_par)
p_recalc <- pkw(q_calc, alpha_par, beta_par)
print(paste("Original p:", p_check, " Recalculated p:", p_recalc))
# abs(p_check - p_recalc) < 1e-9 # Should be TRUE

# Boundary conditions
print(qkw(c(0, 1), alpha_par, beta_par)) # Should be 0, 1
print(qkw(c(-Inf, 0), alpha_par, beta_par, log_p = TRUE)) # Should be 0, 1
```

---

qmc *Quantile Function of the McDonald (Mc)/Beta Power Distribution*

Description

Computes the quantile function (inverse CDF) for the McDonald (Mc) distribution (also known as Beta Power) with parameters `gamma` (γ), `delta` (δ), and `lambda` (λ). It finds the value q such that $P(X \leq q) = p$. This distribution is a special case of the Generalized Kumaraswamy (GKw) distribution where $\alpha = 1$ and $\beta = 1$.

Usage

```
qmc(p, gamma, delta, lambda, lower_tail = TRUE, log_p = FALSE)
```

Arguments

`p`	Vector of probabilities (values between 0 and 1).
`gamma`	Shape parameter `gamma` > 0. Can be a scalar or a vector. Default: 1.0.
`delta`	Shape parameter `delta` >= 0. Can be a scalar or a vector. Default: 0.0.
`lambda`	Shape parameter `lambda` > 0. Can be a scalar or a vector. Default: 1.0.
`lower_tail`	Logical; if `TRUE` (default), probabilities are $p = P(X \leq q)$, otherwise, probabilities are $p = P(X > q)$.
`log_p`	Logical; if `TRUE`, probabilities `p` are given as $\log(p)$. Default: `FALSE`.

Details

The quantile function $Q(p)$ is the inverse of the CDF $F(q)$. The CDF for the Mc ($\alpha = 1, \beta = 1$) distribution is $F(q) = I_{q^\lambda}(\gamma, \delta + 1)$, where $I_z(a, b)$ is the regularized incomplete beta function (see `pmc`).

To find the quantile q, we first invert the Beta function part: let $y = I_p^{-1}(\gamma, \delta + 1)$, where $I_p^{-1}(a, b)$ is the inverse computed via `qbeta`. We then solve $q^\lambda = y$ for q, yielding the quantile function:

$$Q(p) = \left[I_p^{-1}(\gamma, \delta + 1)\right]^{1/\lambda}$$

The function uses this formula, calculating $I_p^{-1}(\gamma, \delta + 1)$ via `qbeta(p, gamma, delta + 1, ...)` while respecting the `lower_tail` and `log_p` arguments. This is equivalent to the general GKw quantile function (`qgkw`) evaluated with $\alpha = 1, \beta = 1$.

Value

A vector of quantiles corresponding to the given probabilities `p`. The length of the result is determined by the recycling rule applied to the arguments (`p`, `gamma`, `delta`, `lambda`). Returns:

- `0` for `p = 0` (or `p = -Inf` if `log_p = TRUE`, when `lower_tail = TRUE`).
- `1` for `p = 1` (or `p = 0` if `log_p = TRUE`, when `lower_tail = TRUE`).
- `NaN` for `p < 0` or `p > 1` (or corresponding log scale).
- `NaN` for invalid parameters (e.g., `gamma <= 0`, `delta < 0`, `lambda <= 0`).

Boundary return values are adjusted accordingly for `lower_tail = FALSE`.

Author(s)

Lopes, J. E.

References

McDonald, J. B. (1984). Some generalized functions for the size distribution of income. *Econometrica*, 52(3), 647-663.

Cordeiro, G. M., & de Castro, M. (2011). A new family of generalized distributions. *Journal of Statistical Computation and Simulation*,

Kumaraswamy, P. (1980). A generalized probability density function for double-bounded random processes. *Journal of Hydrology*, *46*(1-2), 79-88.

See Also

qgkw (parent distribution quantile function), dmc, pmc, rmc (other Mc functions), qbeta

Examples

```
# Example values
p_vals <- c(0.1, 0.5, 0.9)
gamma_par <- 2.0
delta_par <- 1.5
lambda_par <- 1.0 # Equivalent to Beta(gamma, delta+1)

# Calculate quantiles using qmc
quantiles <- qmc(p_vals, gamma_par, delta_par, lambda_par)
print(quantiles)
# Compare with Beta quantiles
print(stats::qbeta(p_vals, shape1 = gamma_par, shape2 = delta_par + 1))

# Calculate quantiles for upper tail probabilities P(X > q) = p
quantiles_upper <- qmc(p_vals, gamma_par, delta_par, lambda_par,
                       lower_tail = FALSE)
print(quantiles_upper)
# Check: qmc(p, ..., lt=F) == qmc(1-p, ..., lt=T)
print(qmc(1 - p_vals, gamma_par, delta_par, lambda_par))

# Calculate quantiles from log probabilities
log_p_vals <- log(p_vals)
quantiles_logp <- qmc(log_p_vals, gamma_par, delta_par, lambda_par, log_p = TRUE)
print(quantiles_logp)
# Check: should match original quantiles
print(quantiles)

# Compare with qgkw setting alpha = 1, beta = 1
quantiles_gkw <- qgkw(p_vals, alpha = 1.0, beta = 1.0, gamma = gamma_par,
                      delta = delta_par, lambda = lambda_par)
print(paste("Max difference:", max(abs(quantiles - quantiles_gkw)))) # Should be near zero

# Verify inverse relationship with pmc
p_check <- 0.75
q_calc <- qmc(p_check, gamma_par, delta_par, lambda_par) # Use lambda != 1
p_recalc <- pmc(q_calc, gamma_par, delta_par, lambda_par)
print(paste("Original p:", p_check, " Recalculated p:", p_recalc))
# abs(p_check - p_recalc) < 1e-9 # Should be TRUE
# Boundary conditions
print(qmc(c(0, 1), gamma_par, delta_par, lambda_par)) # Should be 0, 1
print(qmc(c(-Inf, 0), gamma_par, delta_par, lambda_par, log_p = TRUE)) # Should be 0, 1
```

`rbeta_` *Random Generation for the Beta Distribution (gamma, delta+1 Parameterization)*

Description

Generates random deviates from the standard Beta distribution, using a parameterization common in generalized distribution families. The distribution is parameterized by `gamma` (γ) and `delta` (δ), corresponding to the standard Beta distribution with shape parameters `shape1 = gamma` and `shape2 = delta + 1`. This is a special case of the Generalized Kumaraswamy (GKw) distribution where $\alpha = 1$, $\beta = 1$, and $\lambda = 1$.

Usage

```
rbeta_(n, gamma, delta)
```

Arguments

`n`	Number of observations. If `length(n) > 1`, the length is taken to be the number required. Must be a non-negative integer.
`gamma`	First shape parameter (`shape1`), $\gamma > 0$. Can be a scalar or a vector. Default: 1.0.
`delta`	Second shape parameter is `delta + 1` (`shape2`), requires $\delta \geq 0$ so that `shape2 >= 1`. Can be a scalar or a vector. Default: 0.0 (leading to `shape2 = 1`, i.e., Uniform).

Details

This function generates samples from a Beta distribution with parameters `shape1 = gamma` and `shape2 = delta + 1`. It is equivalent to calling `stats::rbeta(n, shape1 = gamma, shape2 = delta + 1)`.

This distribution arises as a special case of the five-parameter Generalized Kumaraswamy (GKw) distribution (`rgkw`) obtained by setting $\alpha = 1$, $\beta = 1$, and $\lambda = 1$. It is therefore also equivalent to the McDonald (Mc)/Beta Power distribution (`rmc`) with $\lambda = 1$.

The function likely calls R's underlying `rbeta` function but ensures consistent parameter recycling and handling within the C++ environment, matching the style of other functions in the related families.

Value

A numeric vector of length `n` containing random deviates from the Beta($\gamma, \delta + 1$) distribution, with values in (0, 1). The length of the result is determined by `n` and the recycling rule applied to the parameters (`gamma`, `delta`). Returns `NaN` if parameters are invalid (e.g., `gamma <= 0`, `delta < 0`).

Author(s)

Lopes, J. E.

References

Johnson, N. L., Kotz, S., & Balakrishnan, N. (1995). *Continuous Univariate Distributions, Volume 2* (2nd ed.). Wiley.

Cordeiro, G. M., & de Castro, M. (2011). A new family of generalized distributions. *Journal of Statistical Computation and Simulation*,

Devroye, L. (1986). *Non-Uniform Random Variate Generation*. Springer-Verlag.

See Also

`rbeta` (standard R implementation), `rgkw` (parent distribution random generation), `rmc` (McDonald/Beta Power random generation), `dbeta_`, `pbeta_`, `qbeta_` (other functions for this parameterization, if they exist).

Examples

```
set.seed(2030) # for reproducibility

# Generate 1000 samples using rbeta_
gamma_par <- 2.0 # Corresponds to shape1
delta_par <- 3.0 # Corresponds to shape2 - 1
shape1 <- gamma_par
shape2 <- delta_par + 1

x_sample <- rbeta_(1000, gamma = gamma_par, delta = delta_par)
summary(x_sample)

# Compare with stats::rbeta
x_sample_stats <- stats::rbeta(1000, shape1 = shape1, shape2 = shape2)
# Visually compare histograms or QQ-plots
hist(x_sample, main="rbeta_ Sample", freq=FALSE, breaks=30)
curve(dbeta_(x, gamma_par, delta_par), add=TRUE, col="red", lwd=2)
hist(x_sample_stats, main="stats::rbeta Sample", freq=FALSE, breaks=30)
curve(stats::dbeta(x, shape1, shape2), add=TRUE, col="blue", lwd=2)
# Compare summary stats (should be similar due to randomness)
print(summary(x_sample))
print(summary(x_sample_stats))

# Compare summary stats with rgkw(alpha=1, beta=1, lambda=1)
x_sample_gkw <- rgkw(1000, alpha = 1.0, beta = 1.0, gamma = gamma_par,
                     delta = delta_par, lambda = 1.0)
```

```
print("Summary stats for rgkw(a=1,b=1,l=1) sample:")
print(summary(x_sample_gkw))

# Compare summary stats with rmc(lambda=1)
x_sample_mc <- rmc(1000, gamma = gamma_par, delta = delta_par, lambda = 1.0)
print("Summary stats for rmc(l=1) sample:")
print(summary(x_sample_mc))
```

---

rbkw *Random Number Generation for the Beta-Kumaraswamy (BKw) Distribution*

Description

Generates random deviates from the Beta-Kumaraswamy (BKw) distribution with parameters `alpha` (α), `beta` (β), `gamma` (γ), and `delta` (δ). This distribution is a special case of the Generalized Kumaraswamy (GKw) distribution where the parameter $\lambda = 1$.

Usage

```
rbkw(n, alpha, beta, gamma, delta)
```

Arguments

n	Number of observations. If `length(n) > 1`, the length is taken to be the number required. Must be a non-negative integer.
alpha	Shape parameter `alpha` > 0. Can be a scalar or a vector. Default: 1.0.
beta	Shape parameter `beta` > 0. Can be a scalar or a vector. Default: 1.0.
gamma	Shape parameter `gamma` > 0. Can be a scalar or a vector. Default: 1.0.
delta	Shape parameter `delta` >= 0. Can be a scalar or a vector. Default: 0.0.

Details

The generation method uses the relationship between the GKw distribution and the Beta distribution. The general procedure for GKw (`rgkw`) is: If $W \sim \mathrm{Beta}(\gamma, \delta + 1)$, then $X = \{1 - [1 - W^{1/\lambda}]^{1/\beta}\}^{1/\alpha}$ follows the GKw($\alpha, \beta, \gamma, \delta, \lambda$) distribution.

For the BKw distribution, $\lambda = 1$. Therefore, the algorithm simplifies to:

1. Generate $V \sim \mathrm{Beta}(\gamma, \delta + 1)$ using `rbeta`.
2. Compute the BKw variate $X = \{1 - (1 - V)^{1/\beta}\}^{1/\alpha}$.

This procedure is implemented efficiently, handling parameter recycling as needed.

Value

A vector of length `n` containing random deviates from the BKw distribution. The length of the result is determined by `n` and the recycling rule applied to the parameters (`alpha`, `beta`, `gamma`, `delta`). Returns `NaN` if parameters are invalid (e.g., `alpha <= 0`, `beta <= 0`, `gamma <= 0`, `delta < 0`).

Author(s)

Lopes, J. E.

References

Cordeiro, G. M., & de Castro, M. (2011). A new family of generalized distributions. *Journal of Statistical Computation and Simulation*,

Kumaraswamy, P. (1980). A generalized probability density function for double-bounded random processes. *Journal of Hydrology*, *46*(1-2), 79-88.

Devroye, L. (1986). *Non-Uniform Random Variate Generation*. Springer-Verlag. (General methods for random variate generation).

See Also

`rgkw` (parent distribution random generation), `dbkw`, `pbkw`, `qbkw` (other BKw functions), `rbeta`

Examples

```
set.seed(2026) # for reproducibility

# Generate 1000 random values from a specific BKw distribution
alpha_par <- 2.0
beta_par <- 1.5
gamma_par <- 1.0
delta_par <- 0.5

x_sample_bkw <- rbkw(1000, alpha = alpha_par, beta = beta_par,
                     gamma = gamma_par, delta = delta_par)
summary(x_sample_bkw)

# Histogram of generated values compared to theoretical density
hist(x_sample_bkw, breaks = 30, freq = FALSE, # freq=FALSE for density
     main = "Histogram of BKw Sample", xlab = "x", ylim = c(0, 2.5))
curve(dbkw(x, alpha = alpha_par, beta = beta_par, gamma = gamma_par,
           delta = delta_par),
      add = TRUE, col = "red", lwd = 2, n = 201)
legend("topright", legend = "Theoretical PDF", col = "red", lwd = 2, bty = "n")

# Comparing empirical and theoretical quantiles (Q-Q plot)
prob_points <- seq(0.01, 0.99, by = 0.01)
theo_quantiles <- qbkw(prob_points, alpha = alpha_par, beta = beta_par,
                       gamma = gamma_par, delta = delta_par)
emp_quantiles <- quantile(x_sample_bkw, prob_points, type = 7)
```

```
plot(theo_quantiles, emp_quantiles, pch = 16, cex = 0.8,
     main = "Q-Q Plot for BKw Distribution",
     xlab = "Theoretical Quantiles", ylab = "Empirical Quantiles (n=1000)")
abline(a = 0, b = 1, col = "blue", lty = 2)

# Compare summary stats with rgkw(..., lambda=1, ...)
# Note: individual values will differ due to randomness
x_sample_gkw <- rgkw(1000, alpha = alpha_par, beta = beta_par, gamma = gamma_par,
                     delta = delta_par, lambda = 1.0)
print("Summary stats for rbkw sample:")
print(summary(x_sample_bkw))
print("Summary stats for rgkw(lambda=1) sample:")
print(summary(x_sample_gkw)) # Should be similar
```

---

rekw *Random Number Generation for the Exponentiated Kumaraswamy (EKw) Distribution*

Description

Generates random deviates from the Exponentiated Kumaraswamy (EKw) distribution with parameters `alpha` (α), `beta` (β), and `lambda` (λ). This distribution is a special case of the Generalized Kumaraswamy (GKw) distribution where $\gamma = 1$ and $\delta = 0$.

Usage

```
rekw(n, alpha, beta, lambda)
```

Arguments

`n`	Number of observations. If `length(n) > 1`, the length is taken to be the number required. Must be a non-negative integer.
`alpha`	Shape parameter `alpha` > 0. Can be a scalar or a vector. Default: 1.0.
`beta`	Shape parameter `beta` > 0. Can be a scalar or a vector. Default: 1.0.
`lambda`	Shape parameter `lambda` > 0 (exponent parameter). Can be a scalar or a vector. Default: 1.0.

Details

The generation method uses the inverse transform (quantile) method. That is, if U is a random variable following a standard Uniform distribution on (0, 1), then $X = Q(U)$ follows the EKw distribution, where $Q(u)$ is the EKw quantile function (`qekw`):

$$Q(u) = \left\{1 - \left[1 - u^{1/\lambda}\right]^{1/\beta}\right\}^{1/\alpha}$$

This is computationally equivalent to the general GKw generation method (`rgkw`) when specialized for $\gamma = 1, \delta = 0$, as the required Beta(1, 1) random variate is equivalent to a standard Uniform(0, 1) variate. The implementation generates U using `runif` and applies the transformation above.

Value

A vector of length `n` containing random deviates from the EKw distribution. The length of the result is determined by `n` and the recycling rule applied to the parameters (`alpha`, `beta`, `lambda`). Returns `NaN` if parameters are invalid (e.g., `alpha <= 0`, `beta <= 0`, `lambda <= 0`).

Author(s)

Lopes, J. E.

References

Nadarajah, S., Cordeiro, G. M., & Ortega, E. M. (2012). The exponentiated Kumaraswamy distribution. *Journal of the Franklin Institute*, *349*(3),

Cordeiro, G. M., & de Castro, M. (2011). A new family of generalized distributions. *Journal of Statistical Computation and Simulation*,

Kumaraswamy, P. (1980). A generalized probability density function for double-bounded random processes. *Journal of Hydrology*, *46*(1-2), 79-88.

Devroye, L. (1986). *Non-Uniform Random Variate Generation*. Springer-Verlag. (General methods for random variate generation).

See Also

`rgkw` (parent distribution random generation), `dekw`, `pekw`, `qekw` (other EKw functions), `runif`

Examples

```
set.seed(2027) # for reproducibility

# Generate 1000 random values from a specific EKw distribution
alpha_par <- 2.0
beta_par <- 3.0
lambda_par <- 1.5

x_sample_ekw <- rekw(1000, alpha = alpha_par, beta = beta_par, lambda = lambda_par)
summary(x_sample_ekw)

# Histogram of generated values compared to theoretical density
hist(x_sample_ekw, breaks = 30, freq = FALSE, # freq=FALSE for density
     main = "Histogram of EKw Sample", xlab = "x", ylim = c(0, 3.0))
curve(dekw(x, alpha = alpha_par, beta = beta_par, lambda = lambda_par),
      add = TRUE, col = "red", lwd = 2, n = 201)
legend("topright", legend = "Theoretical PDF", col = "red", lwd = 2, bty = "n")

# Comparing empirical and theoretical quantiles (Q-Q plot)
prob_points <- seq(0.01, 0.99, by = 0.01)
```

```
theo_quantiles <- qekw(prob_points, alpha = alpha_par, beta = beta_par,
                       lambda = lambda_par)
emp_quantiles <- quantile(x_sample_ekw, prob_points, type = 7)

plot(theo_quantiles, emp_quantiles, pch = 16, cex = 0.8,
     main = "Q-Q Plot for EKw Distribution",
     xlab = "Theoretical Quantiles", ylab = "Empirical Quantiles (n=1000)")
abline(a = 0, b = 1, col = "blue", lty = 2)

# Compare summary stats with rgkw(..., gamma=1, delta=0, ...)
# Note: individual values will differ due to randomness
x_sample_gkw <- rgkw(1000, alpha = alpha_par, beta = beta_par, gamma = 1.0,
                     delta = 0.0, lambda = lambda_par)
print("Summary stats for rekw sample:")
print(summary(x_sample_ekw))
print("Summary stats for rgkw(gamma=1, delta=0) sample:")
print(summary(x_sample_gkw)) # Should be similar
```

rgkw *Generalized Kumaraswamy Distribution Random Generation*

Description

Generates random deviates from the five-parameter Generalized Kumaraswamy (GKw) distribution defined on the interval (0, 1).

Usage

```
rgkw(n, alpha, beta, gamma, delta, lambda)
```

Arguments

`n`	Number of observations. If `length(n)` > 1, the length is taken to be the number required. Must be a non-negative integer.
`alpha`	Shape parameter `alpha` > 0. Can be a scalar or a vector. Default: 1.0.
`beta`	Shape parameter `beta` > 0. Can be a scalar or a vector. Default: 1.0.
`gamma`	Shape parameter `gamma` > 0. Can be a scalar or a vector. Default: 1.0.
`delta`	Shape parameter `delta` >= 0. Can be a scalar or a vector. Default: 0.0.
`lambda`	Shape parameter `lambda` > 0. Can be a scalar or a vector. Default: 1.0.

Details

The generation method relies on the transformation property: if $V \sim \mathrm{Beta}(\gamma, \delta + 1)$, then the random variable `X` defined as

$$X = \left\{ 1 - \left[1 - V^{1/\lambda} \right]^{1/\beta} \right\}^{1/\alpha}$$

follows the GKw($\alpha, \beta, \gamma, \delta, \lambda$) distribution.

The algorithm proceeds as follows:

1. Generate `V` from `stats::rbeta(n, shape1 = gamma, shape2 = delta + 1)`.
2. Calculate $v = V^{1/\lambda}$.
3. Calculate $w = (1 - v)^{1/\beta}$.
4. Calculate $x = (1 - w)^{1/\alpha}$.

Parameters (`alpha`, `beta`, `gamma`, `delta`, `lambda`) are recycled to match the length required by `n`. Numerical stability is maintained by handling potential edge cases during the transformations.

Value

A vector of length `n` containing random deviates from the GKw distribution. The length of the result is determined by `n` and the recycling rule applied to the parameters (`alpha`, `beta`, `gamma`, `delta`, `lambda`). Returns `NaN` if parameters are invalid (e.g., `alpha <= 0`, `beta <= 0`, `gamma <= 0`, `delta < 0`, `lambda <= 0`).

Author(s)

Lopes, J. E.

References

Cordeiro, G. M., & de Castro, M. (2011). A new family of generalized distributions. *Journal of Statistical Computation and Simulation*

Kumaraswamy, P. (1980). A generalized probability density function for double-bounded random processes. *Journal of Hydrology*, *46*(1-2), 79-88.

See Also

`dgkw`, `pgkw`, `qgkw`, `rbeta`, `set.seed`

Examples

```
set.seed(1234) # for reproducibility

# Generate 1000 random values from a specific GKw distribution (Kw case)
x_sample <- rgkw(1000, alpha = 2, beta = 3, gamma = 1, delta = 0, lambda = 1)
summary(x_sample)

# Histogram of generated values compared to theoretical density
```

```
hist(x_sample, breaks = 30, freq = FALSE, # freq=FALSE for density scale
     main = "Histogram of GKw(2,3,1,0,1) Sample", xlab = "x", ylim = c(0, 2.5))
curve(dgkw(x, alpha = 2, beta = 3, gamma = 1, delta = 0, lambda = 1),
      add = TRUE, col = "red", lwd = 2, n = 201)
legend("topright", legend = "Theoretical PDF", col = "red", lwd = 2, bty = "n")

# Comparing empirical and theoretical quantiles (Q-Q plot)
prob_points <- seq(0.01, 0.99, by = 0.01)
theo_quantiles <- qgkw(prob_points, alpha = 2, beta = 3, gamma = 1, delta = 0, lambda = 1)
emp_quantiles <- quantile(x_sample, prob_points)

plot(theo_quantiles, emp_quantiles, pch = 16, cex = 0.8,
     main = "Q-Q Plot for GKw(2,3,1,0,1)",
     xlab = "Theoretical Quantiles", ylab = "Empirical Quantiles (n=1000)")
abline(a = 0, b = 1, col = "blue", lty = 2)

# Using vectorized parameters: generate 1 value for each alpha
alphas_vec <- c(0.5, 1.0, 2.0)
n_param <- length(alphas_vec)
samples_vec <- rgkw(n_param, alpha = alphas_vec, beta = 2, gamma = 1, delta = 0, lambda = 1)
print(samples_vec) # One sample for each alpha value
# Result length matches n=3, parameters alpha recycled accordingly

# Example with invalid parameters (should produce NaN)
invalid_sample <- rgkw(1, alpha = -1, beta = 2, gamma = 1, delta = 0, lambda = 1)
print(invalid_sample)
```

rkkw *Random Number Generation for the kkw Distribution*

Description

Generates random deviates from the Kumaraswamy-Kumaraswamy (kkw) distribution with parameters `alpha` (α), `beta` (β), `delta` (δ), and `lambda` (λ). This distribution is a special case of the Generalized Kumaraswamy (GKw) distribution where the parameter $\gamma = 1$.

Usage

```
rkkw(n, alpha, beta, delta, lambda)
```

Arguments

`n`	Number of observations. If `length(n)` > 1, the length is taken to be the number required. Must be a non-negative integer.
`alpha`	Shape parameter `alpha` > 0. Can be a scalar or a vector. Default: 1.0.
`beta`	Shape parameter `beta` > 0. Can be a scalar or a vector. Default: 1.0.
`delta`	Shape parameter `delta` >= 0. Can be a scalar or a vector. Default: 0.0.
`lambda`	Shape parameter `lambda` > 0. Can be a scalar or a vector. Default: 1.0.

Details

The generation method uses the inverse transform method based on the quantile function (qkkw). The kkw quantile function is:

$$Q(p) = \left[1 - \left\{1 - \left[1 - (1-p)^{1/(\delta+1)}\right]^{1/\lambda}\right\}^{1/\beta}\right]^{1/\alpha}$$

Random deviates are generated by evaluating $Q(p)$ where p is a random variable following the standard Uniform distribution on (0, 1) (runif).

This is equivalent to the general method for the GKw distribution (rgkw) specialized for $\gamma = 1$. The GKw method generates $W \sim \mathrm{Beta}(\gamma, \delta+1)$ and then applies transformations. When $\gamma = 1$, $W \sim \mathrm{Beta}(1, \delta+1)$, which can be generated via $W = 1 - V^{1/(\delta+1)}$ where $V \sim \mathrm{Unif}(0,1)$. Substituting this W into the GKw transformation yields the same result as evaluating $Q(1-V)$ above (noting $p = 1 - V$ is also Uniform).

Value

A vector of length n containing random deviates from the kkw distribution. The length of the result is determined by n and the recycling rule applied to the parameters (alpha, beta, delta, lambda). Returns NaN if parameters are invalid (e.g., alpha <= 0, beta <= 0, delta < 0, lambda <= 0).

Author(s)

Lopes, J. E.

References

Cordeiro, G. M., & de Castro, M. (2011). A new family of generalized distributions. *Journal of Statistical Computation and Simulation*

Kumaraswamy, P. (1980). A generalized probability density function for double-bounded random processes. *Journal of Hydrology*, *46*(1-2), 79-88.

Devroye, L. (1986). *Non-Uniform Random Variate Generation*. Springer-Verlag. (General methods for random variate generation).

See Also

rgkw (parent distribution random generation), dkkw, pkkw, qkkw, runif, rbeta

Examples

```
set.seed(2025) # for reproducibility

# Generate 1000 random values from a specific kkw distribution
alpha_par <- 2.0
beta_par <- 3.0
delta_par <- 0.5
lambda_par <- 1.5

x_sample_kkw <- rkkw(1000, alpha = alpha_par, beta = beta_par,
                       delta = delta_par, lambda = lambda_par)
summary(x_sample_kkw)

# Histogram of generated values compared to theoretical density
hist(x_sample_kkw, breaks = 30, freq = FALSE, # freq=FALSE for density
     main = "Histogram of kkw Sample", xlab = "x", ylim = c(0, 3.5))
curve(dkkw(x, alpha = alpha_par, beta = beta_par, delta = delta_par,
            lambda = lambda_par),
      add = TRUE, col = "red", lwd = 2, n = 201)
legend("topright", legend = "Theoretical PDF", col = "red", lwd = 2, bty = "n")

# Comparing empirical and theoretical quantiles (Q-Q plot)
prob_points <- seq(0.01, 0.99, by = 0.01)
theo_quantiles <- qkkw(prob_points, alpha = alpha_par, beta = beta_par,
                        delta = delta_par, lambda = lambda_par)
emp_quantiles <- quantile(x_sample_kkw, prob_points, type = 7) # type 7 is default

plot(theo_quantiles, emp_quantiles, pch = 16, cex = 0.8,
     main = "Q-Q Plot for kkw Distribution",
     xlab = "Theoretical Quantiles", ylab = "Empirical Quantiles (n=1000)")
abline(a = 0, b = 1, col = "blue", lty = 2)

# Compare summary stats with rgkw(..., gamma=1, ...)
# Note: individual values will differ due to randomness
x_sample_gkw <- rgkw(1000, alpha = alpha_par, beta = beta_par, gamma = 1.0,
                     delta = delta_par, lambda = lambda_par)
print("Summary stats for rkkw sample:")
print(summary(x_sample_kkw))
print("Summary stats for rgkw(gamma=1) sample:")
print(summary(x_sample_gkw)) # Should be similar
```

rkw — *Random Number Generation for the Kumaraswamy (Kw) Distribution*

Description

Generates random deviates from the two-parameter Kumaraswamy (Kw) distribution with shape parameters `alpha` (α) and `beta` (β).

Usage

```
rkw(n, alpha, beta)
```

Arguments

`n`	Number of observations. If `length(n)` > 1, the length is taken to be the number required. Must be a non-negative integer.
`alpha`	Shape parameter `alpha` > 0. Can be a scalar or a vector. Default: 1.0.
`beta`	Shape parameter `beta` > 0. Can be a scalar or a vector. Default: 1.0.

Details

The generation method uses the inverse transform (quantile) method. That is, if U is a random variable following a standard Uniform distribution on (0, 1), then $X = Q(U)$ follows the Kw distribution, where $Q(p)$ is the Kw quantile function (`qkw`):

$$Q(p) = \left\{ 1 - (1-p)^{1/\beta} \right\}^{1/\alpha}$$

The implementation generates U using `runif` and applies this transformation. This is equivalent to the general GKw generation method (`rgkw`) evaluated at $\gamma = 1, \delta = 0, \lambda = 1$.

Value

A vector of length `n` containing random deviates from the Kw distribution, with values in (0, 1). The length of the result is determined by `n` and the recycling rule applied to the parameters (`alpha`, `beta`). Returns `NaN` if parameters are invalid (e.g., `alpha <= 0`, `beta <= 0`).

Author(s)

Lopes, J. E.

References

Kumaraswamy, P. (1980). A generalized probability density function for double-bounded random processes. *Journal of Hydrology*, *46*(1-2), 79-88.

Jones, M. C. (2009). Kumaraswamy's distribution: A beta-type distribution with some tractability advantages. *Statistical Methodology*, *6*(1), 70-81.

Devroye, L. (1986). *Non-Uniform Random Variate Generation*. Springer-Verlag. (General methods for random variate generation).

See Also

`rgkw` (parent distribution random generation), `dkw`, `pkw`, `qkw` (other Kw functions), `runif`

Examples

```
set.seed(2029) # for reproducibility

# Generate 1000 random values from a specific Kw distribution
alpha_par <- 2.0
beta_par <- 3.0

x_sample_kw <- rkw(1000, alpha = alpha_par, beta = beta_par)
summary(x_sample_kw)

# Histogram of generated values compared to theoretical density
hist(x_sample_kw, breaks = 30, freq = FALSE, # freq=FALSE for density
     main = "Histogram of Kw Sample", xlab = "x", ylim = c(0, 2.5))
curve(dkw(x, alpha = alpha_par, beta = beta_par),
      add = TRUE, col = "red", lwd = 2, n = 201)
```

```
legend("top", legend = "Theoretical PDF", col = "red", lwd = 2, bty = "n")

# Comparing empirical and theoretical quantiles (Q-Q plot)
prob_points <- seq(0.01, 0.99, by = 0.01)
theo_quantiles <- qkw(prob_points, alpha = alpha_par, beta = beta_par)
emp_quantiles <- quantile(x_sample_kw, prob_points, type = 7)

plot(theo_quantiles, emp_quantiles, pch = 16, cex = 0.8,
     main = "Q-Q Plot for Kw Distribution",
     xlab = "Theoretical Quantiles", ylab = "Empirical Quantiles (n=1000)")
abline(a = 0, b = 1, col = "blue", lty = 2)

# Compare summary stats with rgkw(..., gamma=1, delta=0, lambda=1)
# Note: individual values will differ due to randomness
x_sample_gkw <- rgkw(1000, alpha = alpha_par, beta = beta_par, gamma = 1.0,
                     delta = 0.0, lambda = 1.0)
print("Summary stats for rkw sample:")
print(summary(x_sample_kw))
print("Summary stats for rgkw(gamma=1, delta=0, lambda=1) sample:")
print(summary(x_sample_gkw)) # Should be similar
```

`rmc` *Random Number Generation for the McDonald (Mc)/Beta Power Distribution*

Description

Generates random deviates from the McDonald (Mc) distribution (also known as Beta Power) with parameters `gamma` (γ), `delta` (δ), and `lambda` (λ). This distribution is a special case of the Generalized Kumaraswamy (GKw) distribution where $\alpha = 1$ and $\beta = 1$.

Usage

```
rmc(n, gamma, delta, lambda)
```

Arguments

`n`	Number of observations. If `length(n) > 1`, the length is taken to be the number required. Must be a non-negative integer.
`gamma`	Shape parameter `gamma` > 0. Can be a scalar or a vector. Default: 1.0.
`delta`	Shape parameter `delta` >= 0. Can be a scalar or a vector. Default: 0.0.
`lambda`	Shape parameter `lambda` > 0. Can be a scalar or a vector. Default: 1.0.

Details

The generation method uses the relationship between the GKw distribution and the Beta distribution. The general procedure for GKw (rgkw) is: If $W \sim \mathrm{Beta}(\gamma, \delta + 1)$, then $X = \{1 - [1 - W^{1/\lambda}]^{1/\beta}\}^{1/\alpha}$ follows the GKw$(\alpha, \beta, \gamma, \delta, \lambda)$ distribution.

For the Mc distribution, $\alpha = 1$ and $\beta = 1$. Therefore, the algorithm simplifies significantly:

1. Generate $U \sim \mathrm{Beta}(\gamma, \delta + 1)$ using `rbeta`.
2. Compute the Mc variate $X = U^{1/\lambda}$.

This procedure is implemented efficiently, handling parameter recycling as needed.

Value

A vector of length `n` containing random deviates from the Mc distribution, with values in (0, 1). The length of the result is determined by `n` and the recycling rule applied to the parameters (`gamma`, `delta`, `lambda`). Returns `NaN` if parameters are invalid (e.g., `gamma <= 0`, `delta < 0`, `lambda <= 0`).

Author(s)

Lopes, J. E.

References

McDonald, J. B. (1984). Some generalized functions for the size distribution of income. *Econometrica*, 52(3), 647-663.

Cordeiro, G. M., & de Castro, M. (2011). A new family of generalized distributions. *Journal of Statistical Computation and Simulation*,

Kumaraswamy, P. (1980). A generalized probability density function for double-bounded random processes. *Journal of Hydrology*, *46*(1-2), 79-88.

Devroye, L. (1986). *Non-Uniform Random Variate Generation*. Springer-Verlag. (General methods for random variate generation).

See Also

`rgkw` (parent distribution random generation), `dmc`, `pmc`, `qmc` (other Mc functions), `rbeta`

Examples

```
set.seed(2028) # for reproducibility

# Generate 1000 random values from a specific Mc distribution
gamma_par <- 2.0
delta_par <- 1.5
lambda_par <- 1.0 # Equivalent to Beta(gamma, delta+1)

x_sample_mc <- rmc(1000, gamma = gamma_par, delta = delta_par,
                   lambda = lambda_par)
summary(x_sample_mc)
```

```
# Histogram of generated values compared to theoretical density
hist(x_sample_mc, breaks = 30, freq = FALSE, # freq=FALSE for density
     main = "Histogram of Mc Sample (Beta Case)", xlab = "x")
curve(dmc(x, gamma = gamma_par, delta = delta_par, lambda = lambda_par),
      add = TRUE, col = "red", lwd = 2, n = 201)
curve(stats::dbeta(x, gamma_par, delta_par + 1), add=TRUE, col="blue", lty=2)
legend("topright", legend = c("Theoretical Mc PDF", "Theoretical Beta PDF"),
       col = c("red", "blue"), lwd = c(2,1), lty=c(1,2), bty = "n")

# Comparing empirical and theoretical quantiles (Q-Q plot)
lambda_par_qq <- 0.7 # Use lambda != 1 for non-Beta case
x_sample_mc_qq <- rmc(1000, gamma = gamma_par, delta = delta_par,
                      lambda = lambda_par_qq)
prob_points <- seq(0.01, 0.99, by = 0.01)
theo_quantiles <- qmc(prob_points, gamma = gamma_par, delta = delta_par,
                      lambda = lambda_par_qq)
emp_quantiles <- quantile(x_sample_mc_qq, prob_points, type = 7)

plot(theo_quantiles, emp_quantiles, pch = 16, cex = 0.8,
     main = "Q-Q Plot for Mc Distribution",
     xlab = "Theoretical Quantiles", ylab = "Empirical Quantiles (n=1000)")
abline(a = 0, b = 1, col = "blue", lty = 2)

# Compare summary stats with rgkw(..., alpha=1, beta=1, ...)
# Note: individual values will differ due to randomness
x_sample_gkw <- rgkw(1000, alpha = 1.0, beta = 1.0, gamma = gamma_par,
                     delta = delta_par, lambda = lambda_par_qq)
print("Summary stats for rmc sample:")
print(summary(x_sample_mc_qq))
print("Summary stats for rgkw(alpha=1, beta=1) sample:")
print(summary(x_sample_gkw)) # Should be similar
```

Index