Package ‘gkwreg’

April 15, 2025

Title Generalized Kumaraswamy Regression Models for Bounded Data

Version 1.0.3

Description Implements regression models for bounded continuous data in the open interval (0,1) using the five-parameter Generalized Kumaraswamy distribution. Supports modeling all distribution parameters (alpha, beta, gamma, delta, lambda) as functions of predictors through various link functions. Provides efficient maximum likelihood estimation via Template Model Builder ('TMB'), offering comprehensive diagnostics, model comparison tools, and simulation methods. Particularly useful for analyzing proportions, rates, indices, and other bounded response data with complex distributional features not adequately captured by simpler models.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

Imports Formula, stats, graphics, TMB, Rcpp, RcppArmadillo, magrittr, ggplot2, ggpubr, gridExtra, rappdirs, patchwork, fmsb, scales, tidyr, reshape2, numDeriv

LinkingTo Rcpp, RcppArmadillo, RcppEigen

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

NeedsCompilation yes

Author Lopes J. E. [aut, cre]

Maintainer Lopes J. E. <evandeilton@gmail.com>

Repository CRAN

Date/Publication 2025-04-15 19:50:02 UTC

Contents

`AIC.gkwfit` *Calculate AIC or BIC for gkwfit Objects*

Description

Computes the Akaike Information Criterion (AIC) or variants like the Bayesian Information Criterion (BIC) for one or more fitted model objects of class "gkwfit".

Usage

```
## S3 method for class 'gkwfit'
AIC(object, ..., k = 2)
```

Arguments

`object`	An object of class "gkwfit", typically the result of a call to `gkwfit`.
`...`	Optionally, more fitted model objects of class "gkwfit".
`k`	Numeric scalar specifying the penalty per parameter. The default `k = 2` corresponds to the traditional AIC. Use `k = log(n)` (where n is the number of observations) for the BIC (Bayesian Information Criterion).

Details

This function calculates an information criterion based on the formula $-2 \times \log Likelihood + k \times df$, where df represents the number of estimated parameters in the model (degrees of freedom).

It relies on the `logLik.gkwfit` method to extract the log-likelihood and the degrees of freedom for each model.

When comparing multiple models fitted to the **same data**, the model with the lower AIC or BIC value is generally preferred. The function returns a sorted data frame to facilitate this comparison when multiple objects are provided.

Value

- If only one `object` is provided: A single numeric value representing the calculated criterion (AIC or BIC).
- If multiple objects are provided: A `data.frame` with rows corresponding to the models and columns for the degrees of freedom (`df`) and the calculated criterion value (named `AIC`, regardless of the value of `k`). The data frame is sorted in ascending order based on the criterion values. Row names are derived from the deparsed calls of the fitted models.

Author(s)

Lopes, J. E.

See Also

`gkwfit`, `AIC`, `logLik.gkwfit`, `BIC.gkwfit`

Examples

```
set.seed(2203)
y <- rkw(1000, alpha = 2.5, beta = 1.5)

# Fit different models to the same data
fit1_kw <- gkwfit(y, family = "kw", silent = TRUE)
fit2_bkw <- gkwfit(y, family = "bkw", silent = TRUE)
fit3_gkw <- gkwfit(y, family = "gkw", silent = TRUE)

# Calculate AIC for a single model
aic1 <- AIC(fit1_kw)
print(aic1)

# Compare AIC values for multiple models
aic_comparison <- c(AIC(fit1_kw), AIC(fit2_bkw), AIC(fit3_gkw))
print(aic_comparison)
```

`AIC.gkwreg` *Akaike's Information Criterion for GKw Regression Models*

Description

Calculates the Akaike Information Criterion (AIC) for one or more fitted Generalized Kumaraswamy (GKw) regression model objects (class "gkwreg"). AIC is commonly used for model selection, penalizing model complexity.

Usage

```
## S3 method for class 'gkwreg'
AIC(object, ..., k = 2)
```

Arguments

object	An object of class "gkwreg", typically the result of a call to gkwreg.
...	Optionally, one or more additional fitted model objects of class "gkwreg", for which AIC should also be calculated.
k	Numeric, the penalty per parameter. The default k = 2 corresponds to the traditional AIC. Using k = log(nobs) would yield BIC (though using BIC.gkwreg is preferred for that).

Details

The AIC is calculated based on the maximized log-likelihood (L) and the number of estimated parameters (p) in the model:

$$AIC = -2\log(L) + k \times p$$

This function retrieves the log-likelihood and the number of parameters (`df`) using the `logLik.gkwreg` method for the fitted `gkwreg` object(s). Models with lower AIC values are generally preferred, as they indicate a better balance between goodness of fit and model parsimony.

When comparing multiple models passed via `...`, the function relies on `AIC`'s default method for creating a comparison table, which in turn calls `logLik` for each provided object.

For small sample sizes relative to the number of parameters, the second-order AIC (AICc) might be more appropriate:

$$AICc = AIC + \frac{2p(p+1)}{n-p-1}$$

where n is the number of observations. AICc is not directly computed by this function but can be calculated manually using the returned AIC, p (from `attr(logLik(object), "df")`), and n (from `attr(logLik(object), "nobs")`).

Value

If just one `object` is provided, returns a single numeric AIC value. If multiple objects are provided via `...`, returns a `data.frame` with rows corresponding to the models and columns for the degrees of freedom (`df`) and the AIC values, sorted by AIC.

Author(s)

Lopes, J. E.

References

Akaike, H. (1974). A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, **19**(6), 716-723.

Burnham, K. P., & Anderson, D. R. (2002). *Model Selection and Multimodel Inference: A Practical Information-Theoretic Approach* (2nd ed.). Springer-Verlag.

See Also

`gkwreg`, `logLik.gkwreg`, `BIC.gkwreg`, `AIC`

Examples

```
# Assume 'df' exists with response 'y' and predictors 'x1', 'x2', 'x3'
# and that rkw() is available and data is appropriate (0 < y < 1).
set.seed(123)
n <- 100
x1 <- runif(n)
x2 <- rnorm(n)
x3 <- factor(rbinom(n, 1, 0.4))
alpha <- exp(0.5 + 0.2 * x1)
beta <- exp(1.0 - 0.1 * x2 + 0.3 * (x3 == "1"))
y <- rkw(n, alpha = alpha, beta = beta) # Placeholder if rkw not available
y <- pmax(pmin(y, 1 - 1e-7), 1e-7)
df <- data.frame(y = y, x1 = x1, x2 = x2, x3 = x3)
```

```
# Fit two competing models
kw_reg1 <- gkwreg(y ~ x1 | x2, data = df, family = "kw")
kw_reg2 <- gkwreg(y ~ x1 | x2 + x3, data = df, family = "kw") # More complex beta model
kw_reg3 <- gkwreg(y ~ 1 | x2 + x3, data = df, family = "kw") # Simpler alpha model

# Calculate AIC for a single model
aic1 <- AIC(kw_reg1)
print(aic1)

# Compare models using AIC (lower is better)
model_comparison_aic <- c(AIC(kw_reg1), AIC(kw_reg2), AIC(kw_reg3))
print(model_comparison_aic)

# Calculate AIC with a different penalty (e.g., k=4)
aic1_k4 <- AIC(kw_reg1, k = 4)
print(aic1_k4)
```

---

anova.gkwfit *Compare Fitted gkwfit Models using Likelihood Ratio Tests*

Description

Computes Likelihood Ratio Tests (LRT) to compare two or more nested models fitted using `gkwfit`. It produces a table summarizing the models and the test statistics.

Usage

```
## S3 method for class 'gkwfit'
anova(object, ...)
```

Arguments

`object`	An object of class "gkwfit", representing the first fitted model.
`...`	One or more additional objects of class "gkwfit", representing subsequent fitted models, assumed to be nested within each other or the first model.

Details

This function performs pairwise likelihood ratio tests between consecutively ordered models (ordered by their degrees of freedom). It assumes the models are nested and are fitted to the same dataset. A warning is issued if the number of observations differs between models.

The Likelihood Ratio statistic is calculated as $LR = 2 \times (\log L_{complex} - \log L_{simple})$. This statistic is compared to a Chi-squared distribution with degrees of freedom equal to the difference in the number of parameters between the two models ($\Delta df = df_{complex} - df_{simple}$).

The output table includes the number of parameters (`N.Par`), AIC, BIC, log-likelihood (`LogLik`), the test description (`Test`), the LR statistic (`LR stat.`), and the p-value (`Pr(>Chi)`). Models are ordered by increasing complexity (number of parameters).

Warnings are issued if models do not appear correctly nested based on degrees of freedom or if the log-likelihood decreases for a more complex model, as the LRT results may not be meaningful in such cases.

The function relies on a working `logLik.gkwfit` method to extract necessary information (log-likelihood, df, nobs).

Value

An object of class `c("anova.gkwfit", "anova", "data.frame")`. This data frame contains rows for each model and columns summarizing the fit and the pairwise likelihood ratio tests. It includes:

`N.Par`	Number of estimated parameters (degrees of freedom).
`AIC`	Akaike Information Criterion.
`BIC`	Bayesian Information Criterion.
`LogLik`	Log-likelihood value.
`Test`	Description of the pairwise comparison (e.g., "1 vs 2").
`LR stat.`	Likelihood Ratio test statistic.
`Pr(>Chi)`	P-value from the Chi-squared test.

The table is printed using a method that mimics `print.anova`.

Author(s)

Lopes, J. E.

See Also

`gkwfit`, `logLik.gkwfit`, `AIC.gkwfit`, `BIC.gkwfit`, `anova`

Examples

```
## Not run:
# Load required packages
library(ggplot2)
library(patchwork)
library(betareg)
# Generate data from GKw distribution
set.seed(2203)
n <- 1000
y <- rgkw(n, alpha = 2, beta = 3, gamma = 1.5, delta = 0.2, lambda = 1.2)

# Fit models from GKw family respecting their parameter structures
# Full GKw model: 5 parameters (alpha, beta, gamma, delta, lambda)
fit_gkw <- gkwfit(data = y, family = "gkw", plot = FALSE)

# BKw model: 4 parameters (alpha, beta, gamma, delta)
fit_bkw <- gkwfit(data = y, family = "bkw", plot = FALSE)

# KKw model: 4 parameters (alpha, beta, delta, lambda)
```

```
fit_kkw <- gkwfit(data = y, family = "kkw", plot = FALSE)

# EKw model: 3 parameters (alpha, beta, lambda)
fit_ekw <- gkwfit(data = y, family = "ekw", plot = FALSE)

# Mc model: 3 parameters (gamma, delta, lambda)
fit_mc <- gkwfit(data = y, family = "mc", plot = FALSE)

# Kw model: 2 parameters (alpha, beta)
fit_kw <- gkwfit(data = y, family = "kw", plot = FALSE)

# Beta model: 2 parameters (gamma, delta)
fit_beta <- gkwfit(data = y, family = "beta", plot = FALSE)

# Test 1: BKw vs GKw (testing lambda)
# H0: lambda=1 (BKw) vs H1: lambda!=1 (GKw)
cat("=== Testing BKw vs GKw (adding lambda parameter) ===\n")
test_bkw_gkw <- anova(fit_bkw, fit_gkw)
print(test_bkw_gkw)

# Test 2: KKw vs GKw (testing gamma)
# H0: gamma=1 (KKw) vs H1: gamma!=1 (GKw)
cat("\n=== Testing KKw vs GKw (adding gamma parameter) ===\n")
test_kkw_gkw <- anova(fit_kkw, fit_gkw)
print(test_kkw_gkw)

# Test 3: Kw vs EKw (testing lambda)
# H0: lambda=1 (Kw) vs H1: lambda!=1 (EKw)
cat("\n=== Testing Kw vs EKw (adding lambda parameter) ===\n")
test_kw_ekw <- anova(fit_kw, fit_ekw)
print(test_kw_ekw)

# Test 4: Beta vs Mc (testing lambda)
# H0: lambda=1 (Beta) vs H1: lambda!=1 (Mc)
cat("\n=== Testing Beta vs Mc (adding lambda parameter) ===\n")
test_beta_mc <- anova(fit_beta, fit_mc)
print(test_beta_mc)

# Visualize model comparison
# Create dataframe summarizing all models
models_df <- data.frame(
  Model = c("GKw", "BKw", "KKw", "EKw", "Mc", "Kw", "Beta"),
  Parameters = c(
    paste("alpha,beta,gamma,delta,lambda"),
    paste("alpha,beta,gamma,delta"),
    paste("alpha,beta,delta,lambda"),
    paste("alpha,beta,lambda"),
    paste("gamma,delta,lambda"),
    paste("alpha,beta"),
    paste("gamma,delta")
  ),
  Param_count = c(5, 4, 4, 3, 3, 2, 2),
  LogLik = c(
```

```
    as.numeric(logLik(fit_gkw)),
    as.numeric(logLik(fit_bkw)),
    as.numeric(logLik(fit_kkw)),
    as.numeric(logLik(fit_ekw)),
    as.numeric(logLik(fit_mc)),
    as.numeric(logLik(fit_kw)),
    as.numeric(logLik(fit_beta))
  ),
  AIC = c(
    fit_gkw$AIC,
    fit_bkw$AIC,
    fit_kkw$AIC,
    fit_ekw$AIC,
    fit_mc$AIC,
    fit_kw$AIC,
    fit_beta$AIC
  ),
  BIC = c(
    fit_gkw$BIC,
    fit_bkw$BIC,
    fit_kkw$BIC,
    fit_ekw$BIC,
    fit_mc$BIC,
    fit_kw$BIC,
    fit_beta$BIC
  )
)

# Sort by AIC
models_df <- models_df[order(models_df$AIC), ]
print(models_df)

# Create comprehensive visualization
# Plot showing model hierarchy and information criteria
p1 <- ggplot(models_df, aes(x = Param_count, y = LogLik, label = Model)) +
  geom_point(size = 3) +
  geom_text(vjust = -0.8) +
  labs(
    title = "Log-likelihood vs Model Complexity",
    x = "Number of Parameters",
    y = "Log-likelihood"
  ) +
  theme_minimal()

# Create information criteria comparison
models_df_long <- tidyr::pivot_longer(
  models_df,
  cols = c("AIC", "BIC"),
  names_to = "Criterion",
  values_to = "Value"
)

p2 <- ggplot(models_df_long, aes(x = reorder(Model, -Value), y = Value, fill = Criterion)) +
```

```
  geom_bar(stat = "identity", position = "dodge") +
  labs(
    title = "Information Criteria Comparison",
    x = "Model",
    y = "Value (lower is better)"
  ) +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

# Print plots
print(p1 + p2)

# ============================================================================
# Manual LR tests to demonstrate underlying calculations
# ============================================================================
# Function to perform manual likelihood ratio test
manual_lr_test <- function(model_restricted, model_full, alpha = 0.05) {
  # Extract log-likelihoods
  ll_restricted <- as.numeric(logLik(model_restricted))
  ll_full <- as.numeric(logLik(model_full))

  # Calculate test statistic
  lr_stat <- -2 * (ll_restricted - ll_full)

  # Calculate degrees of freedom (parameter difference)
  df <- length(coef(model_full)) - length(coef(model_restricted))

  # Calculate p-value
  p_value <- pchisq(lr_stat, df = df, lower.tail = FALSE)

  # Return results
  list(
    lr_statistic = lr_stat,
    df = df,
    p_value = p_value,
    significant = p_value < alpha,
    critical_value = qchisq(1 - alpha, df = df)
  )
}

# Example: Manual LR test for BKw vs GKw (testing lambda parameter)
cat("\n=== Manual LR test: BKw vs GKw ===\n")
lr_bkw_gkw <- manual_lr_test(fit_bkw, fit_gkw)
cat("LR statistic:", lr_bkw_gkw$lr_statistic, "\n")
cat("Degrees of freedom:", lr_bkw_gkw$df, "\n")
cat("P-value:", lr_bkw_gkw$p_value, "\n")
cat("Critical value (alpha=0.05):", lr_bkw_gkw$critical_value, "\n")
cat("Decision:", ifelse(lr_bkw_gkw$significant,
  "Reject H0: Lambda is significantly different from 1",
  "Fail to reject H0: Lambda is not significantly different from 1"
), "\n")

# Example: Manual LR test for Kw vs EKw (testing lambda parameter)
```

```
cat("\n=== Manual LR test: Kw vs EKw ===\n")
lr_kw_ekw <- manual_lr_test(fit_kw, fit_ekw)
cat("LR statistic:", lr_kw_ekw$lr_statistic, "\n")
cat("Degrees of freedom:", lr_kw_ekw$df, "\n")
cat("P-value:", lr_kw_ekw$p_value, "\n")
cat("Critical value (alpha=0.05):", lr_kw_ekw$critical_value, "\n")
cat("Decision:", ifelse(lr_kw_ekw$significant,
  "Reject H0: Lambda is significantly different from 1",
  "Fail to reject H0: Lambda is not significantly different from 1"
), "\n")

# ============================================================================
# Real data application with correct model nesting
# ============================================================================
if (requireNamespace("betareg", quietly = TRUE)) {
  data("ReadingSkills", package = "betareg")
  y <- ReadingSkills$accuracy

  # Fit models
  rs_gkw <- gkwfit(data = y, family = "gkw", plot = FALSE)
  rs_bkw <- gkwfit(data = y, family = "bkw", plot = FALSE)
  rs_kkw <- gkwfit(data = y, family = "kkw", plot = FALSE)
  rs_kw <- gkwfit(data = y, family = "kw", plot = FALSE)
  rs_beta <- gkwfit(data = y, family = "beta", plot = FALSE)

  # Test nested models
  cat("\n=== Real data: Testing BKw vs GKw (adding lambda) ===\n")
  rs_test_bkw_gkw <- anova(rs_bkw, rs_gkw)
  print(rs_test_bkw_gkw)

  cat("\n=== Real data: Testing Kw vs KKw (adding delta and lambda) ===\n")
  rs_test_kw_kkw <- anova(rs_kw, rs_kkw)
  print(rs_test_kw_kkw)

  # Compare non-nested models with information criteria
  cat("\n=== Real data: Comparing non-nested Beta vs Kw ===\n")
  rs_compare_beta_kw <- anova(rs_beta, rs_kw)
  print(rs_compare_beta_kw)

  # Summarize all models
  cat("\n=== Real data: Model comparison summary ===\n")
  models_rs <- c("GKw", "BKw", "KKw", "Kw", "Beta")
  aic_values <- c(rs_gkw$AIC, rs_bkw$AIC, rs_kkw$AIC, rs_kw$AIC, rs_beta$AIC)
  bic_values <- c(rs_gkw$BIC, rs_bkw$BIC, rs_kkw$BIC, rs_kw$BIC, rs_beta$BIC)
  loglik_values <- c(
    as.numeric(logLik(rs_gkw)),
    as.numeric(logLik(rs_bkw)),
    as.numeric(logLik(rs_kkw)),
    as.numeric(logLik(rs_kw)),
    as.numeric(logLik(rs_beta))
  )

  df_rs <- data.frame(
```

```
    Model = models_rs,
    LogLik = loglik_values,
    AIC = aic_values,
    BIC = bic_values
  )
  df_rs <- df_rs[order(df_rs$AIC), ]
  print(df_rs)

  # Determine the best model for the data
  best_model <- df_rs$Model[which.min(df_rs$AIC)]
  cat("\nBest model based on AIC:", best_model, "\n")
}

## End(Not run)
```

`BIC.gkwfit` *Calculate Bayesian Information Criterion (BIC) for gkwfit Objects*

Description

Computes the Bayesian Information Criterion (BIC), sometimes called the Schwarz criterion (SIC), for one or more fitted model objects of class `"gkwfit"`.

Usage

```
## S3 method for class 'gkwfit'
BIC(object, ...)
```

Arguments

`object`	An object of class `"gkwfit"`, typically the result of a call to `gkwfit`.
`...`	Optionally, more fitted model objects of class `"gkwfit"`.

Details

This function calculates the BIC based on the formula $-2 \times \log Likelihood + \log(n) \times df$, where n is the number of observations and df represents the number of estimated parameters in the model (degrees of freedom).

It relies on the `logLik.gkwfit` method to extract the log-likelihood, the degrees of freedom (`df`), and the number of observations (`nobs`) for each model. Ensure that `logLik.gkwfit` is defined and returns a valid `"logLik"` object with appropriate attributes.

When comparing multiple models fitted to the **same data**, the model with the lower BIC value is generally preferred, as BIC tends to penalize model complexity more heavily than AIC for larger sample sizes. The function returns a sorted data frame to facilitate this comparison when multiple objects are provided. A warning is issued if models were fitted to different numbers of observations.

Value

- If only one `object` is provided: A single numeric value, the calculated BIC.
- If multiple objects are provided: A `data.frame` with rows corresponding to the models and columns for the degrees of freedom (`df`) and the calculated BIC value (named `BIC`). The data frame is sorted in ascending order based on the BIC values. Row names are generated from the deparsed calls or the names of the arguments passed to BIC.

Author(s)

Lopes, J. E. (with refinements)

See Also

`gkwfit`, `BIC`, `logLik.gkwfit`, `AIC.gkwfit`

Examples

```
set.seed(2203)
y <- rkw(1000, alpha = 2.5, beta = 1.5)

# Fit different models to the same data
fit1_kw <- gkwfit(y, family = "kw", silent = TRUE)
fit2_bkw <- gkwfit(y, family = "bkw", silent = TRUE)
fit3_gkw <- gkwfit(y, family = "gkw", silent = TRUE)

# Calculate BIC for a single model
bic1 <- BIC(fit1_kw)
print(bic1)

# Compare BIC values for multiple models
bic_comparison <- c(BIC(fit1_kw), BIC(fit2_bkw), BIC(fit3_gkw))
print(bic_comparison)
```

`BIC.gkwreg` *Bayesian Information Criterion for GKw Regression Models*

Description

Calculates the Bayesian Information Criterion (BIC), also known as Schwarz's Bayesian Criterion (SBC), for one or more fitted Generalized Kumaraswamy (GKw) regression model objects (class `"gkwreg"`). BIC is used for model selection and tends to penalize model complexity more heavily than AIC, especially for larger datasets.

Usage

```
## S3 method for class 'gkwreg'
BIC(object, ...)
```

Arguments

object	An object of class "gkwreg", typically the result of a call to gkwreg.
...	Optionally, one or more additional fitted model objects of class "gkwreg", for which BIC should also be calculated.

Details

The BIC is calculated based on the maximized log-likelihood (L), the number of estimated parameters (p) in the model, and the number of observations (n):

$$BIC = -2\log(L) + p \times \log(n)$$

This function retrieves the log-likelihood, the number of parameters (df), and the number of observations (nobs) using the logLik.gkwreg method for the fitted gkwreg object(s).

Models with lower BIC values are generally preferred. The penalty term $p\log(n)$ increases more rapidly with sample size n compared to AIC's penalty $2p$, meaning BIC favors simpler models more strongly in larger samples. BIC can be motivated from a Bayesian perspective as an approximation related to Bayes factors.

When comparing multiple models passed via ..., the function relies on BIC's default method for creating a comparison table, which in turn calls logLik for each provided object.

Value

If just one object is provided, returns a single numeric BIC value. If multiple objects are provided via ..., returns a data.frame with rows corresponding to the models and columns for the degrees of freedom (df) and the BIC values, sorted by BIC.

Author(s)

Lopes, J. E.

References

Schwarz, G. (1978). Estimating the dimension of a model. *The Annals of Statistics*, **6**(2), 461-464.

See Also

gkwreg, logLik.gkwreg, AIC.gkwreg, BIC

Examples

```
# Assume 'df' exists with response 'y' and predictors 'x1', 'x2', 'x3'
# and that rkw() is available and data is appropriate (0 < y < 1).
set.seed(123)
n <- 100
x1 <- runif(n)
x2 <- rnorm(n)
x3 <- factor(rbinom(n, 1, 0.4))
alpha <- exp(0.5 + 0.2 * x1)
beta <- exp(1.0 - 0.1 * x2 + 0.3 * (x3 == "1"))
y <- rkw(n, alpha = alpha, beta = beta) # Placeholder if rkw not available
y <- pmax(pmin(y, 1 - 1e-7), 1e-7)
df <- data.frame(y = y, x1 = x1, x2 = x2, x3 = x3)

# Fit two competing models
kw_reg1 <- gkwreg(y ~ x1 | x2, data = df, family = "kw")
kw_reg2 <- gkwreg(y ~ x1 | x2 + x3, data = df, family = "kw") # More complex beta model
kw_reg3 <- gkwreg(y ~ 1 | x2 + x3, data = df, family = "kw") # Simpler alpha model

# Calculate BIC for a single model
bic1 <- BIC(kw_reg1)
print(bic1)

# Compare models using BIC (lower is better)
model_comparison_bic <- c(BIC(kw_reg1), BIC(kw_reg2), BIC(kw_reg3))
print(model_comparison_bic)
```

---

calculateCoxSnellResiduals

Calculate Cox-Snell Residuals

Description

Computes Cox-Snell residuals defined as -log(1 - F(y)), where F is the cumulative distribution function.

Usage

```
calculateCoxSnellResiduals(y, params, family = "gkw")
```

Arguments

`y`	NumericVector of observations.
`params`	NumericMatrix with parameters (columns: alpha, beta, gamma, delta, lambda).
`family`	String specifying the distribution family (default: "gkw").

Value

NumericVector of Cox-Snell residuals.

`calculateDensities` *Calculate Densities for Distribution*

Description

Evaluates the density (or its logarithm) for each observation given the parameters.

Usage

```
calculateDensities(y, params, family = "gkw", log = FALSE)
```

Arguments

`y`	NumericVector of observations.
`params`	NumericMatrix with parameters (columns: alpha, beta, gamma, delta, lambda).
`family`	String specifying the distribution family (default: "gkw").
`log`	Logical indicating whether to return the log-density (default FALSE).

Value

NumericVector containing the evaluated densities.

`calculateDevianceResiduals`
Calculate Deviance Residuals

Description

Computes deviance residuals based on the log-likelihood of the observations.

Usage

```
calculateDevianceResiduals(y, fitted, params, family = "gkw")
```

Arguments

`y`	NumericVector of observations.
`fitted`	NumericVector of fitted values (means).
`params`	NumericMatrix with parameters (columns: alpha, beta, gamma, delta, lambda).
`family`	String specifying the distribution family (default: "gkw").

Value

NumericVector of deviance residuals.

calculateMeans *Calculate Means for Distribution*

Description

Computes the mean of the distribution for each observation using numerical integration (quadrature) with caching to avoid redundant calculations.

Usage

```
calculateMeans(params, family = "gkw")
```

Arguments

params	NumericMatrix with parameters (columns: alpha, beta, gamma, delta, lambda).
family	String specifying the distribution family (default: "gkw").

Value

NumericVector containing the calculated means for each observation.

calculateModifiedDevianceResiduals *Calculate Modified Deviance Residuals*

Description

Adjusts deviance residuals to have a distribution closer to N(0,1) by standardizing them.

Usage

```
calculateModifiedDevianceResiduals(y, fitted, params, family = "gkw")
```

Arguments

y	NumericVector of observations.
fitted	NumericVector of fitted values (means).
params	NumericMatrix with parameters (columns: alpha, beta, gamma, delta, lambda).
family	String specifying the distribution family (default: "gkw").

Value

NumericVector of modified deviance residuals.

calculateParameters — *Calculate Parameters for the Generalized Kumaraswamy Distribution*

Description

Computes the parameters (alpha, beta, gamma, delta, lambda) for each observation based on design matrices and regression coefficients, applying a positive link function as specified by link types and scale factors.

Usage

```
calculateParameters(
  X1,
  X2,
  X3,
  X4,
  X5,
  beta1,
  beta2,
  beta3,
  beta4,
  beta5,
  link_types,
  scale_factors,
  family = "gkw"
)
```

Arguments

`X1`	NumericMatrix design matrix for alpha.
`X2`	NumericMatrix design matrix for beta.
`X3`	NumericMatrix design matrix for gamma.
`X4`	NumericMatrix design matrix for delta.
`X5`	NumericMatrix design matrix for lambda.
`beta1`	NumericVector regression coefficients for X1.
`beta2`	NumericVector regression coefficients for X2.
`beta3`	NumericVector regression coefficients for X3.
`beta4`	NumericVector regression coefficients for X4.
`beta5`	NumericVector regression coefficients for X5.
`link_types`	IntegerVector containing the link function type for each parameter.
`scale_factors`	NumericVector with scale factors for each parameter.
`family`	String specifying the distribution family (default: "gkw").

Value

NumericMatrix with n rows and 5 columns corresponding to the calculated parameters.

`calculatePartialResiduals` *Calculate Partial Residuals*

Description

Computes partial residuals for a selected covariate by adding the product of the regression coefficient and the corresponding design matrix value to the raw residual.

Usage

```
calculatePartialResiduals(y, fitted, X, beta, covariate_idx)
```

Arguments

`y`	NumericVector of observations.
`fitted`	NumericVector of fitted values.
`X`	NumericMatrix of design matrix values.
`beta`	NumericVector of regression coefficients.
`covariate_idx`	Integer index for the selected covariate (0-indexed).

Value

NumericVector of partial residuals.

`calculatePearsonResiduals` *Calculate Pearson Residuals*

Description

Computes the Pearson residuals based on the observed values, fitted means, and the approximate variance of the distribution.

Usage

```
calculatePearsonResiduals(y, fitted, params, family = "gkw")
```

Arguments

y	NumericVector of observations.
fitted	NumericVector of fitted values (means).
params	NumericMatrix with parameters (columns: alpha, beta, gamma, delta, lambda).
family	String specifying the distribution family (default: "gkw").

Value

NumericVector of Pearson residuals.

---

calculateProbabilities

Calculate Cumulative Probabilities for Distribution

Description

Computes the cumulative probabilities (CDF) for each observation given the parameters.

Usage

```
calculateProbabilities(y, params, family = "gkw")
```

Arguments

y	NumericVector of observations.
params	NumericMatrix with parameters (columns: alpha, beta, gamma, delta, lambda).
family	String specifying the distribution family (default: "gkw").

Value

NumericVector containing the evaluated cumulative probabilities.

`calculateQuantileResiduals` *Calculate Quantile Residuals*

Description

Computes quantile residuals by transforming the cumulative distribution function (CDF) values to the standard normal quantiles.

Usage

```
calculateQuantileResiduals(y, params, family = "gkw")
```

Arguments

`y`	NumericVector of observations.
`params`	NumericMatrix with parameters (columns: alpha, beta, gamma, delta, lambda).
`family`	String specifying the distribution family (default: "gkw").

Value

NumericVector of quantile residuals.

`calculateQuantiles` *Calculate Quantiles for Distribution*

Description

Computes quantiles for the given probability levels using a bisection method for the first set of parameters in the matrix.

Usage

```
calculateQuantiles(probs, params, family = "gkw")
```

Arguments

`probs`	NumericVector of probabilities (values in (0,1)).
`params`	NumericMatrix with parameters (columns: alpha, beta, gamma, delta, lambda).
`family`	String specifying the distribution family (default: "gkw").

Value

NumericVector containing the calculated quantiles.

`calculateResponseResiduals`
Calculate Response Residuals

Description

Computes the raw response residuals as the difference between the observed and fitted values.

Usage

```
calculateResponseResiduals(y, fitted)
```

Arguments

`y`	NumericVector of observations.
`fitted`	NumericVector of fitted values.

Value

NumericVector of response residuals.

`calculateScoreResiduals`
Calculate Score Residuals

Description

Computes score residuals based on the numerical derivative (score) of the log-likelihood with respect to the observation.

Usage

```
calculateScoreResiduals(y, fitted, params, family = "gkw")
```

Arguments

`y`	NumericVector of observations.
`fitted`	NumericVector of fitted values (means).
`params`	NumericMatrix with parameters (columns: alpha, beta, gamma, delta, lambda).
`family`	String specifying the distribution family (default: "gkw").

Value

NumericVector of score residuals.

`coef.gkwfit` *Extract Model Coefficients from a gkwfit Object*

Description

Extracts the estimated coefficients for the parameters of a model fitted by `gkwfit`. This is an S3 method for the generic `coef` function.

Usage

```
## S3 method for class 'gkwfit'
coef(object, ...)
```

Arguments

`object`	An object of class `"gkwfit"`, typically the result of a call to `gkwfit`.
`...`	Additional arguments (currently ignored).

Value

A named numeric vector containing the estimated coefficients for the parameters of the specified GKw family distribution. The names correspond to the parameter names (e.g., `"alpha"`, `"beta"`, etc.).

Author(s)

Lopes, J. E.

See Also

`gkwfit`, `coef`, `vcov.gkwfit`, `logLik.gkwfit`

Examples

```
# Generate data and fit model
set.seed(2203)
y <- rgkw(50, alpha = 1.5, beta = 2.5, gamma = 1.2, delta = 0.3, lambda = 1.1)
fit <- gkwfit(data = y, family = "gkw", plot = FALSE)

# Extract all coefficients
params <- coef(fit)
print(params)

# Access specific parameters
alpha_est <- coef(fit)["alpha"]
lambda_est <- coef(fit)["lambda"]
cat("Estimated alpha:", alpha_est, "\n")
cat("Estimated lambda:", lambda_est, "\n")
```

coef.gkwreg *Extract Coefficients from a Fitted GKw Regression Model*

Description

Extracts the estimated regression coefficients from a fitted Generalized Kumaraswamy (GKw) regression model object of class "gkwreg". This is an S3 method for the generic coef function.

Usage

```
## S3 method for class 'gkwreg'
coef(object, ...)
```

Arguments

object	An object of class "gkwreg", typically the result of a call to gkwreg.
...	Additional arguments, currently ignored by this method.

Details

This function provides the standard way to access the estimated regression coefficients from a model fitted with gkwreg. It simply extracts the coefficients component from the fitted model object. The function coefficients is an alias for this function.

Value

A named numeric vector containing the estimated regression coefficients for all modeled parameters. The names indicate the parameter (e.g., alpha, beta) and the corresponding predictor variable (e.g., (Intercept), x1).

Author(s)

Lopes, J. E.

See Also

gkwreg, summary.gkwreg, coef, confint

confint.gkwfit *Compute Confidence Intervals for gkwfit Parameters*

Description

Computes confidence intervals for one or more parameters in a model fitted by gkwfit. It uses the Wald method based on the estimated coefficients and their standard errors derived from the variance-covariance matrix.

Usage

```
## S3 method for class 'gkwfit'
confint(object, parm, level = 0.95, ...)
```

Arguments

object	An object of class "gkwfit", typically the result of a call to gkwfit. The object must contain valid coefficient estimates and a corresponding variance-covariance matrix (usually requires fitting with `hessian = TRUE`).
parm	A specification of which parameters are to be given confidence intervals, either a vector of numbers (indices) or a vector of names. If missing, confidence intervals are computed for all parameters that have a valid standard error available. Parameter indices refer to the order of parameters for which standard errors could be calculated.
level	The confidence level required (default: 0.95).
...	Additional arguments (currently ignored).

Details

This function calculates confidence intervals using the Wald method: $Estimate \pm z \times SE$, where z is the appropriate quantile from the standard normal distribution for the given confidence `level`.

It relies on the results from `coef.gkwfit` and `vcov.gkwfit` (or directly accesses `object$coefficients` and `object$vcov` if those methods aren't defined). It checks for the validity of the variance-covariance matrix before proceeding.

Since all parameters of the GKw family distributions are constrained to be positive, the lower bound of the confidence interval is truncated at a small positive value (`.Machine$double.eps^0.5`) if the calculated lower bound is non-positive.

If `parm` is specified, it selects the parameters for which to compute intervals. Numeric indices in `parm` refer to the parameters that have calculable standard errors, not necessarily all parameters in the model (if some were fixed or had estimation issues).

Value

A matrix with columns giving lower and upper confidence limits for each parameter specified in `parm`. The columns are labeled with quantile percentages (e.g., "2.5 %" and "97.5 %" for `level = 0.95`). Row names are taken from the parameter names. Returns `NULL` or stops with an error if coefficients or a valid variance-covariance matrix cannot be extracted.

Author(s)

Lopes, J. E.

See Also

gkwfit, confint, coef.gkwfit, vcov.gkwfit

Examples

```
# Generate data and fit model
set.seed(2203)
y <- rkw(50, alpha = 2, beta = 3)
fit <- gkwfit(data = y, family = "kw", plot = FALSE, hessian = TRUE)

# Calculate confidence intervals for all parameters
ci <- confint(fit)
print(ci)

# 90% confidence interval
ci_90 <- confint(fit, level = 0.90)
print(ci_90)

# Confidence interval for specific parameter
ci_alpha <- confint(fit, parm = "alpha")
print(ci_alpha)
```

---

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")
```

---

extract_gof_stats *Extract Key Statistics from gkwgof Objects*

Description

Extracts the most important goodness-of-fit statistics from one or more gkwgof objects into a concise data frame format for easy comparison and reporting.

Usage

```
extract_gof_stats(..., statistics = "all")
```

Arguments

...	One or more objects of class "gkwgof", or a list of such objects.
statistics	Character vector specifying which statistics to include. Available options are: "all" (default), "information" (for AIC, BIC), "distance" (for KS, AD), "correlation" (for P-P, Q-Q correlations), or "prediction" (for RMSE, MAE).

Value

A data frame containing the requested statistics for each gkwgof object.

Examples

```
# Generate sample data
set.seed(123)
data <- rkw(n = 200, alpha = 2.5, beta = 1.8)

# Fit multiple models
fit_kw <- gkwfit(data, family = "kw")
fit_beta <- gkwfit(data, family = "beta")

# Calculate goodness-of-fit statistics for each model
gof_kw <- gkwgof(fit_kw, print_summary = FALSE)
gof_beta <- gkwgof(fit_beta, print_summary = FALSE)

# Extract key statistics
summary_stats <- extract_gof_stats(gof_kw, gof_beta)
print(summary_stats)

# Extract only information criteria
ic_stats <- extract_gof_stats(gof_kw, gof_beta, statistics = "information")
print(ic_stats)
```

`fitted.gkwreg` *Extract Fitted Values from a Generalized Kumaraswamy Regression Model*

Description

Extracts the fitted mean values (predicted expected values of the response) from a fitted Generalized Kumaraswamy (GKw) regression model object of class `"gkwreg"`. This is an S3 method for the generic `fitted.values` function.

Usage

```
## S3 method for class 'gkwreg'
fitted(object, family = NULL, ...)
```

Arguments

`object`	An object of class `"gkwreg"`, typically the result of a call to `gkwreg`.
`family`	Character string specifying the distribution family under which the fitted mean values should be calculated. If `NULL` (default), the family stored within the fitted `object` is used. Specifying a different family (e.g., `"beta"`) will trigger recalculation of the fitted means based on that family's mean structure, using the original model's estimated coefficients mapped to the relevant parameters. Available options match those in `gkwreg`: `"gkw"`, `"bkw"`, `"kkw"`, `"ekw"`, `"mc"`, `"kw"`, `"beta"`.
`...`	Additional arguments, currently ignored by this method.

Details

This function retrieves or calculates the fitted values, which represent the estimated conditional mean of the response variable given the covariates ($E(Y|X)$).

The function attempts to retrieve fitted values efficiently using the following priority:

1. Directly from the `fitted.values` component stored in the `object`, if available and complete. It includes logic to handle potentially incomplete stored values via interpolation (`approx`) for very large datasets where only a sample might be stored.
2. By recalculating the mean using stored parameter vectors for each observation (`object$parameter_vectors`) and an internal function (`calculateMeans`), if available.
3. From the `fitted` component within the TMB report (`object$tmb_object$report()`), if available, potentially using interpolation as above.
4. As a fallback, by calling `predict(object, type = "response", family = family)`.

Specifying a `family` different from the one used to fit the model will always force recalculation using the `predict` method (step 4).

Value

A numeric vector containing the fitted mean values. These values are typically bounded between 0 and 1, corresponding to the scale of the original response variable. The length of the vector corresponds to the number of observations used in the model fit (considering `subset` and `na.action`).

Author(s)

Lopes, J. E.

See Also

`gkwreg`, `predict.gkwreg`, `residuals.gkwreg`, `fitted.values`

Examples

```
# Assume 'mydata' exists with response 'y' and predictors 'x1', 'x2'
# and that rgkw() is available and data is appropriate (0 < y < 1).
set.seed(456)
n <- 100
x1 <- runif(n, -1, 1)
x2 <- rnorm(n)
alpha <- exp(0.5 + 0.2 * x1)
beta <- exp(0.8 - 0.3 * x1 + 0.1 * x2)
gamma <- exp(0.6)
delta <- plogis(0.0 + 0.2 * x1)
lambda <- exp(-0.2 + 0.1 * x2)
# Use stats::rbeta as placeholder if rgkw is not available
y <- stats::rbeta(n, shape1 = gamma * alpha, shape2 = delta * beta) # Approximation
y <- pmax(pmin(y, 1 - 1e-7), 1e-7)
mydata <- data.frame(y = y, x1 = x1, x2 = x2)

# Fit a GKw model
```

```
model <- gkwreg(y ~ x1 | x1 + x2 | 1 | x1 | x2, data = mydata, family = "gkw")

# Extract fitted values (using the original 'gkw' family)
fitted_vals_gkw <- fitted(model)

# Extract fitted values recalculated as if it were a Beta model
# (using the fitted gamma and delta coefficients)
fitted_vals_beta <- fitted(model, family = "beta")

# Plot observed vs. fitted (using original family)
response_y <- model$y # Get the response variable used in the fit
if (!is.null(response_y)) {
  plot(response_y, fitted_vals_gkw,
    xlab = "Observed Response", ylab = "Fitted Mean Value",
    main = "Observed vs Fitted Values (GKw Family)",
    pch = 1, col = "blue"
  )
  abline(0, 1, col = "red", lty = 2) # Line y = x
} else {
  print("Response variable not found in model object to create plot.")
}

# Compare fitted values under different family assumptions
head(data.frame(GKw_Fitted = fitted_vals_gkw, Beta_Fitted = fitted_vals_beta))
```

---

gkwfit — *Fit Generalized Kumaraswamy Distribution via Maximum Likelihood Estimation using TMB*

Description

Fits any distribution from the Generalized Kumaraswamy (GKw) family to data using maximum likelihood estimation through Template Model Builder (TMB). The function supports several optimization methods including R's nlminb and various optim algorithms.

Usage

```
gkwfit(
  data,
  family = "gkw",
  start = NULL,
  fixed = NULL,
  method = "nlminb",
  use_moments = FALSE,
  hessian = TRUE,
  profile = FALSE,
  npoints = 20,
```

```
    plot = TRUE,
    conf.level = 0.95,
    optimizer.control = list(),
    submodels = FALSE,
    silent = TRUE,
    ...
)
```

Arguments

data	A numeric vector with values strictly between 0 and 1. Values at the boundaries (0 or 1) may cause issues; consider slight adjustments if necessary (see Details).
family	A character string specifying the distribution family. One of: "gkw" (default), "bkw", "kkw", "ekw", "mc", "kw", or "beta". See Details for parameter specifications.
start	Optional list with initial parameter values (using natural parameter names like alpha, beta, etc.). If NULL, reasonable starting values will be determined, potentially using the method of moments if use_moments = TRUE.
fixed	Optional list of parameters to be held fixed at specific values during estimation (e.g., list(lambda = 1)).
method	Optimization method to use. One of: "nlminb" (default), "Nelder-Mead", "BFGS", "CG", "L-BFGS-B" or "SANN". If "nlminb" is selected, R's nlminb function is used; otherwise, R's optim function is used with the specified method.
use_moments	Logical; if TRUE and start = NULL, attempts to use method of moments estimates (via gkwgetstartvalues) as initial values. Default: FALSE.
hessian	Logical; if TRUE, attempts to compute the Hessian matrix at the MLE to estimate standard errors and the variance-covariance matrix using TMB's sdreport. Default: TRUE.
profile	Logical; if TRUE, computes likelihood profiles for parameters using TMB's profiling capabilities. Default: FALSE.
npoints	Integer; number of points to use in profile likelihood calculations (minimum 5). Only relevant if profile = TRUE. Default: 20.
plot	Logical; if TRUE, generates diagnostic plots (histogram with fitted density, QQ-plot) using ggplot2 and patchwork. Default: TRUE.
conf.level	Numeric, the confidence level for confidence intervals calculated from standard errors (requires hessian = TRUE). Default: 0.95.
optimizer.control	List of control parameters passed to the chosen optimizer. The valid parameters depend on the method chosen. See Details.
submodels	Logical; if TRUE, fits relevant nested submodels for comparison via likelihood ratio tests. Default: FALSE.
silent	Logical; if TRUE, suppresses messages during fitting. Default: FALSE.
...	Additional arguments (currently unused).

Details

The `gkwfit` function provides a unified interface for fitting the seven distributions in the Generalized Kumaraswamy family:

- **GKw**: 5 parameters ($\alpha, \beta, \gamma, \delta, \lambda$) - All positive.
- **BKw**: 4 parameters ($\alpha, \beta, \gamma, \delta$), $\lambda = 1$ fixed - All positive.
- **KKw**: 4 parameters ($\alpha, \beta, \delta, \lambda$), $\gamma = 1$ fixed - All positive.
- **EKw**: 3 parameters (α, β, λ), $\gamma = 1, \delta = 0$ fixed - All positive.
- **Mc** (McDonald / Beta Power): 3 parameters (γ, δ, λ), $\alpha = 1, \beta = 1$ fixed - All positive.
- **Kw** (Kumaraswamy): 2 parameters (α, β), $\gamma = 1, \delta = 0, \lambda = 1$ fixed - All positive.
- **Beta**: 2 parameters (γ, δ), $\alpha = 1, \beta = 1, \lambda = 1$ fixed - All positive. (γ, δ correspond to standard Beta shape1, shape2).

This function uses Template Model Builder (TMB) for parameter estimation, which provides accurate and efficient automatic differentiation.

Optimizer Method (`method` argument):

- `"nlminb"`: Uses R's built-in `stats::nlminb` optimizer. Good for problems with box constraints. Default option.
- `"Nelder-Mead"`: Uses R's `stats::optim` with the Nelder-Mead simplex algorithm, which doesn't require derivatives.
- `"BFGS"`: Uses R's `stats::optim` with the BFGS quasi-Newton method for unconstrained optimization.
- `"CG"`: Uses R's `stats::optim` with conjugate gradients method for unconstrained optimization.
- `"L-BFGS-B"`: Uses R's `stats::optim` with the limited-memory BFGS method with box constraints.
- `"SANN"`: Uses R's stats::optim with simulated annealing, a global optimization method useful for problems with multiple local minima.

Optimizer Control (`optimizer.control`): Pass a list with parameters specific to the chosen optimizer:

- For `method = "nlminb"`: Controls are passed to `stats::nlminb`. See `?nlminb` for options like `eval.max`, `iter.max`, `trace`, `rel.tol`, etc.
- For other methods: Controls are passed to `stats::optim`. See `?optim` for options like `maxit`, `trace`, `factr`, `pgtol`, etc.

If `optimizer.control` is empty, reasonable defaults are used for each method.

Data Preprocessing: The function includes basic validation to ensure data is numeric and within (0, 1). It attempts to adjust values exactly at 0 or 1 by a small epsilon (`.Machine$double.eps^0.5`) with a warning, but stops if more than 10% of data needs adjustment. It's generally recommended to handle boundary data appropriately *before* calling `gkwfit`.

Value

An object of class "gkwfit" (inheriting from "list") containing the fitted model results. Key components include:

`coefficients`	Named vector of estimated parameters (on their natural scale).
`std.errors`	Named vector of estimated standard errors (if `hessian = TRUE`).
`coef_summary`	Data frame summarizing estimates, SEs, z-values, and p-values.
`vcov`	Variance-covariance matrix of the estimates (if `hessian = TRUE`).
`loglik`	Log-likelihood value at the maximum.
`AIC`	Akaike Information Criterion.
`BIC`	Bayesian Information Criterion.
`AICc`	Corrected Akaike Information Criterion.
`data`	The input data vector used for fitting.
`nobs`	Number of observations used.
`df`	Number of estimated parameters.
`convergence`	Logical indicating successful convergence.
`message`	Convergence message from the optimizer.
`family`	The specified distribution family.
`method`	The specific optimization method used.
`conf.int`	Data frame with confidence intervals (if `hessian = TRUE`).
`conf.level`	The confidence level used.
`optimizer`	The raw output object from the optimizer function.
`obj`	The TMB object used for fitting (if available).
`fixed`	The list of fixed parameters used.
`profile`	A list containing likelihood profile results (if `profile = TRUE`).
`submodels`	A list of fitted submodels (if `submodels = TRUE`).
`lrt`	A list of likelihood ratio test results comparing nested models (if `submodels = TRUE`).
`gof`	Goodness-of-fit statistics (e.g., AD, CvM, KS).
`diagnostics`	Diagnostic information related to GOF tests.
`plots`	A list or `patchwork` object containing ggplot objects for diagnostics (if `plot = TRUE`).
`call`	The matched function call.

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. doi:10.1016/00221694(80)900360

Cordeiro, G. M., & de Castro, M. (2011). A new family of generalized distributions. *Journal of Statistical Computation and Simulation*, 81(7), 883-898. doi:10.1080/00949650903530745

Kristensen, K., Nielsen, A., Berg, C. W., Skaug, H., & Bell, B. M. (2016). TMB: Automatic Differentiation and Laplace Approximation. *Journal of Statistical Software*, 70(5), 1–21. doi:10.18637/jss.v070.i05

See Also

User-facing S3 methods: `summary.gkwfit`, `print.gkwfit`, `plot.gkwfit`, `coef.gkwfit`, `vcov.gkwfit`, `logLik.gkwfit`, `confint.gkwfit`. Density/distribution functions: `dgkw`, `pgkw`, `qgkw`, `rgkw`.

Examples

```
## Not run:
# Load required packages
library(ggplot2)
library(patchwork)
library(betareg)

# ============================================================================
# EXAMPLE 1: Basic Usage with Simulated Data for Different Distributions
# ============================================================================
# Set seed for reproducibility
set.seed(2203)
n <- 1000

# Generate random samples from various distributions in the GKw family
y_gkw <- rgkw(n, alpha = 2, beta = 3, gamma = 1.5, delta = 0.5, lambda = 1.2)
y_bkw <- rbkw(n, alpha = 2, beta = 3, gamma = 1.5, delta = 0.5) # BKw has lambda fixed at 1
y_kw <- rkw(n, alpha = 2, beta = 3) # Standard Kumaraswamy (gamma=1, delta=0, lambda=1)
y_beta <- rbeta_(n, gamma = 2, delta = 3) # Beta with shape1=2, shape2=3

# Calculate densities for the first 5 observations
head(dgkw(y_gkw[1:5], alpha = 2, beta = 3, gamma = 1.5, delta = 0.5, lambda = 1.2))

# Compare with beta density (note different parameterization)
head(dbeta_(y_beta[1:5], gamma = 2, delta = 3))

# Compute log-likelihood using parameter vector format
# Parameter order: alpha, beta, gamma, delta, lambda
par_gkw <- c(2, 3, 1.5, 0.5, 1.2)
ll_gkw <- llgkw(par_gkw, y_gkw)

par_kw <- c(2, 3) # Kumaraswamy parameters
ll_kw <- llkw(par_kw, y_kw)

cat("Log-likelihood GKw:", ll_gkw, "\nLog-likelihood Kw:", ll_kw, "\n")
```

```
# ============================================================================
# EXAMPLE 2: Visualization and Distribution Comparisons
# ============================================================================
# Generate data for plotting
x_vals <- seq(0.001, 0.999, length.out = 500)

# Calculate densities for different distributions
dens_gkw <- dgkw(x_vals, alpha = 2, beta = 3, gamma = 1.5, delta = 0.5, lambda = 1.2)
dens_bkw <- dbkw(x_vals, alpha = 2, beta = 3, gamma = 1.5, delta = 0.5)
dens_kw <- dkw(x_vals, alpha = 2, beta = 3)
dens_beta <- dbeta_(x_vals, gamma = 2, delta = 3)

# Create and display plot
df <- data.frame(
  x = rep(x_vals, 4),
  density = c(dens_gkw, dens_bkw, dens_kw, dens_beta),
  Distribution = rep(c("GKw", "BKw", "Kw", "Beta"), each = length(x_vals))
)

p <- ggplot(df, aes(x = x, y = density, color = Distribution)) +
  geom_line(linewidth = 1) +
  theme_minimal() +
  labs(
    title = "Density Comparison of GKw Distribution Family",
    x = "x", y = "Density"
  )

print(p)

# Examine quantile functions
# Calculate 0.25, 0.5, and 0.75 quantiles for each distribution
probs <- c(0.25, 0.5, 0.75)
qgkw_values <- qgkw(probs, alpha = 2, beta = 3, gamma = 1.5, delta = 0.5, lambda = 1.2)
qkw_values <- qkw(probs, alpha = 2, beta = 3)
qbeta_values <- qbeta_(probs, gamma = 2, delta = 3)

# Display the quantiles
quantile_df <- data.frame(
  Probability = probs,
  GKw = qgkw_values,
  Kw = qkw_values,
  Beta = qbeta_values
)
print(quantile_df)

# ============================================================================
# EXAMPLE 3: Model Fitting with Simulated Data
# ============================================================================
# Simulate data from GKw
set.seed(2203)
true_params <- list(alpha = 2.5, beta = 1.8, gamma = 1.3, delta = 0.4, lambda = 1.5)
y <- rgkw(n,
  alpha = true_params$alpha,
```

```
  beta = true_params$beta,
  gamma = true_params$gamma,
  delta = true_params$delta,
  lambda = true_params$lambda
)

# Fit full GKw model
fit_gkw <- gkwfit(data = y, family = "gkw")
summary(fit_gkw)

# Fit restricted models for comparison
fit_bkw <- gkwfit(data = y, family = "bkw")
fit_kkw <- gkwfit(data = y, family = "kkw")
fit_kw <- gkwfit(data = y, family = "kw")

# Compare models using AIC
models <- c("GKw", "BKw", "KKw", "Kw")
AIC_values <- c(fit_gkw$AIC, fit_bkw$AIC, fit_kkw$AIC, fit_kw$AIC)
model_comparison <- data.frame(Model = models, AIC = AIC_values)
model_comparison <- model_comparison[order(model_comparison$AIC), ]
print(model_comparison)

# ============================================================================
# EXAMPLE 4: Fixed Parameter Estimation and Likelihood Ratio Tests
# ============================================================================
# Simulate data where delta = 0
set.seed(2203)
true_params <- list(alpha = 1.8, beta = 2.2, gamma = 0.9, delta = 0, lambda = 1.2)
y <- rgkw(n,
  alpha = true_params$alpha,
  beta = true_params$beta,
  gamma = true_params$gamma,
  delta = true_params$delta,
  lambda = true_params$lambda
)

# Fit model with delta fixed at 0
fit_fixed <- gkwfit(data = y, family = "gkw", method = "L-BFGS-B", fixed = list(delta = 0.5))
summary(fit_fixed)

# Fit model without fixing delta (should estimate close to 0)
fit_free <- gkwfit(data = y, family = "gkw")
summary(fit_free)

# Compare models via likelihood ratio test
# H0: delta = 0 vs H1: delta != 0
LR_stat <- -2 * (fit_fixed$loglik - fit_free$loglik)
p_value <- 1 - pchisq(LR_stat, df = 1)
cat("LR test statistic:", LR_stat, "\np-value:", p_value, "\n")

# ============================================================================
# EXAMPLE 5: Profile Likelihood Analysis
# ============================================================================
```

```
set.seed(2203)
y <- rgkw(n, alpha = 2, beta = 3, gamma = 1.5, delta = 0, lambda = 1.2)

# Fit with profile likelihood
fit_profile <- gkwfit(
  data = y,
  family = "gkw",
  profile = TRUE,
  npoints = 15
)

# Examine profile objects
str(fit_profile$profile)
fit_profile$plots

# ============================================================================
# EXAMPLE 6: Real Data Application with Beta Regression Datasets
# ============================================================================
# Example 1: Reading Skills data
data("ReadingSkills", package = "betareg")
y <- ReadingSkills$accuracy

# Summary statistics of the response variable
summary(y)

# Fit different distributions to this data
fit_rs_beta <- gkwfit(data = y, family = "beta")
fit_rs_kw <- gkwfit(data = y, family = "kw")
fit_rs_gkw <- gkwfit(data = y, family = "gkw")

# Model comparison
rs_models <- c("Beta", "Kumaraswamy", "GKw")
rs_AICs <- c(fit_rs_beta$AIC, fit_rs_kw$AIC, fit_rs_gkw$AIC)
rs_BICs <- c(fit_rs_beta$BIC, fit_rs_kw$BIC, fit_rs_gkw$BIC)
rs_comparison <- data.frame(
  Model = rs_models,
  AIC = rs_AICs,
  BIC = rs_BICs,
  Parameters = c(
    length(coef(fit_rs_beta)),
    length(coef(fit_rs_kw)),
    length(coef(fit_rs_gkw))
  )
)
print(rs_comparison[order(rs_comparison$AIC), ])

# Example 2: Gasoline Yield data
data("GasolineYield", package = "betareg")
y <- GasolineYield$yield

# Check range and make adjustments if needed (data must be in (0,1))
if (min(y) <= 0 || max(y) >= 1) {
  # Apply common transformation for proportions that include 0 or 1
```

```
  n_obs <- length(y)
  y <- (y * (n_obs - 1) + 0.5) / n_obs
}

# Fit best model based on previous comparison
best_family <- rs_comparison$Model[1]
fit_gas <- gkwfit(data = y, family = tolower(best_family))
summary(fit_gas)

# Plot fitted density over histogram of data
# Get parameters from fitted model
params <- coef(fit_gas)

# Generate x values and calculate density
x_seq <- seq(min(y), max(y), length.out = 100)
fitted_density <- switch(tolower(best_family),
  "beta" = dbeta_(x_seq, gamma = params["gamma"], delta = params["delta"]),
  "kumaraswamy" = dkw(x_seq, alpha = params["alpha"], beta = params["beta"]),
  "gkw" = dgkw(x_seq,
    alpha = params["alpha"], beta = params["beta"],
    gamma = params["gamma"], delta = params["delta"],
    lambda = params["lambda"]
  )
)

# Create data frame for plotting
df_plot <- data.frame(x = x_seq, density = fitted_density)

# Create plot
p <- ggplot() +
  geom_histogram(
    data = data.frame(y = y), aes(x = y, y = after_stat(density)),
    bins = 30, fill = "lightblue", color = "black", alpha = 0.7
  ) +
  geom_line(data = df_plot, aes(x = x, y = density), color = "red", size = 1) +
  labs(
    title = paste("Fitted", best_family, "Distribution to Gasoline Yield Data"),
    x = "Yield",
    y = "Density"
  ) +
  theme_minimal()

print(p)

# ============================================================================
# EXAMPLE 7: Beta Distribution Variants and Special Functions
# ============================================================================
# Generate samples from beta distribution
set.seed(2203)
y_beta <- rbeta_(n, gamma = 2, delta = 3)

# Calculate density and log-likelihood
beta_dens <- dbeta_(y_beta[1:5], gamma = 2, delta = 3)
```

```
# Using parameter vector format for log-likelihood (gamma, delta)
par_beta <- c(2, 3)
beta_ll <- llbeta(par_beta, y_beta)
cat("Beta density (first 5):", beta_dens, "\n")
cat("Beta log-likelihood:", beta_ll, "\n")

# Gradient of log-likelihood with respect to parameters
beta_grad <- grbeta(par_beta, y_beta)
cat("Gradient of Beta log-likelihood:\n")
print(beta_grad)

# Hessian of log-likelihood with respect to parameters
beta_hess <- hsbeta(par_beta, y_beta)
cat("Hessian of Beta log-likelihood:\n")
print(beta_hess)

# ============================================================================
# EXAMPLE 8: Gradient and Hessian Functions for GKw Distribution
# ============================================================================
# Set seed and generate data
set.seed(2203)

# Define parameters
alpha <- 2
beta <- 1.5
gamma <- 1.2
delta <- 0.3
lambda <- 1.1
par_gkw <- c(alpha, beta, gamma, delta, lambda)

# Generate random sample
y <- rgkw(n, alpha, beta, gamma, delta, lambda)

# Calculate log-likelihood of the sample using parameter vector format
ll <- llgkw(par_gkw, y)
cat("GKw log-likelihood:", ll, "\n")

# Calculate gradient of log-likelihood
gr <- grgkw(par_gkw, y)
cat("GKw log-likelihood gradient:\n")
print(gr)

# Calculate Hessian matrix of log-likelihood
hs <- hsgkw(par_gkw, y)
cat("GKw log-likelihood Hessian:\n")
print(hs)

# ============================================================================
# EXAMPLE 9: Optimization with Custom Gradient and Hessian
# ============================================================================
# Manual optimization demonstration
set.seed(2203)
```

```
# Generate data from a known distribution
true_par <- c(alpha = 1.8, beta = 2.5, gamma = 1.3, delta = 0.2, lambda = 1.1)
y <- rgkw(n,
  alpha = true_par["alpha"],
  beta = true_par["beta"],
  gamma = true_par["gamma"],
  delta = true_par["delta"],
  lambda = true_par["lambda"]
)

# Define the negative log-likelihood function (for minimization)
nll <- function(log_par) {
  # Transform from log-scale to natural scale (ensures positivity)
  par <- exp(log_par)

  # Return negative log-likelihood
  -llgkw(par, y)
}

# Define the gradient function using analytical gradient
gr_func <- function(log_par) {
  # Transform parameters
  par <- exp(log_par)

  # Get the gradient with respect to the original parameters
  gradient <- grgkw(par, y)

  # Apply chain rule for the log transformation
  gradient <- gradient * par

  # Return negative gradient for minimization
  gradient
}

# Starting values (on log scale to ensure positivity)
start_log_par <- log(c(1, 1, 1, 0.1, 1))

# Optimize using L-BFGS-B method with analytic gradient
opt_result <- optim(
  par = start_log_par,
  fn = nll,
  gr = gr_func,
  method = "BFGS",
  control = list(trace = 1, maxit = 100)
)

# Transform parameters back to original scale
estimated_par <- exp(opt_result$par)
names(estimated_par) <- c("alpha", "beta", "gamma", "delta", "lambda")

# Compare with true parameters
params_comparison <- data.frame(
```

```
  True = true_par,
  Estimated = estimated_par,
  Absolute_Error = abs(true_par - estimated_par),
  Relative_Error = abs((true_par - estimated_par) / true_par)
)
print(params_comparison)

# ============================================================================
# EXAMPLE 10: Third Betareg Dataset (ImpreciseTask) and McDonald Distribution
# ============================================================================
data("ImpreciseTask", package = "betareg")
y <- ImpreciseTask$location

# Make sure data is within (0, 1)
if (min(y) <= 0 || max(y) >= 1) {
  # Apply common transformation for proportions
  n_obs <- length(y)
  y <- (y * (n_obs - 1) + 0.5) / n_obs
}

# Fit models from the GKw family
fit_beta <- gkwfit(data = y, family = "beta")
fit_kw <- gkwfit(data = y, family = "kw")
fit_mc <- gkwfit(data = y, family = "mc") # McDonald distribution

# Compare information criteria
ic_comparison <- data.frame(
  Model = c("Beta", "Kumaraswamy", "McDonald"),
  AIC = c(fit_beta$AIC, fit_kw$AIC, fit_mc$AIC),
  BIC = c(fit_beta$BIC, fit_kw$BIC, fit_mc$BIC),
  LogLik = c(fit_beta$loglik, fit_kw$loglik, fit_mc$loglik)
)
print(ic_comparison[order(ic_comparison$AIC), ])

# Get best model
best_model <- ic_comparison$Model[which.min(ic_comparison$AIC)]
best_fit <- switch(tolower(best_model),
  "beta" = fit_beta,
  "kumaraswamy" = fit_kw,
  "mcdonald" = fit_mc
)

# Goodness of fit tests
print(paste("Best model:", best_model))
print(best_fit$gof)

# Generate values from fitted McDonald distribution (if it's the best model)
if (best_model == "McDonald") {
  # McDonald's parameters: gamma, delta, lambda (alpha=1, beta=1 fixed)
  gamma_mc <- coef(fit_mc)["gamma"]
  delta_mc <- coef(fit_mc)["delta"]
  lambda_mc <- coef(fit_mc)["lambda"]
```

```
  # Generate new sample using fitted parameters
  set.seed(2203)
  y_mc_simulated <- rmc(n, gamma = gamma_mc, delta = delta_mc, lambda = lambda_mc)

  # Compare histograms of original and simulated data
  df_orig <- data.frame(value = y, type = "Original")
  df_sim <- data.frame(value = y_mc_simulated, type = "Simulated")
  df_combined <- rbind(df_orig, df_sim)

  # Create comparative histogram
  p <- ggplot(df_combined, aes(x = value, fill = type)) +
    geom_histogram(position = "identity", alpha = 0.5, bins = 30) +
    labs(
      title = "Comparison of Original Data and Simulated McDonald Distribution",
      x = "Value",
      y = "Count"
    ) +
    theme_minimal()

  print(p)
}

# ============================================================================
# EXAMPLE 11: Working with Alternative Distributions in the GKw Family
# ============================================================================
# Explore EKw - Exponential Kumaraswamy distribution (alpha, beta, lambda)
set.seed(2203)

# Generate EKw sample
y_ekw <- rekw(n, alpha = 2.5, beta = 1.5, lambda = 2.0)

# Calculate density and distribution function
ekw_density <- dekw(y_ekw[1:5], alpha = 2.5, beta = 1.5, lambda = 2.0)
ekw_cdf <- pekw(y_ekw[1:5], alpha = 2.5, beta = 1.5, lambda = 2.0)

# Calculate log-likelihood
par_ekw <- c(2.5, 1.5, 2.0) # alpha, beta, lambda for EKw
ll_ekw <- llekw(par_ekw, y_ekw)

cat("EKw density (first 5):", ekw_density, "\n")
cat("EKw CDF (first 5):", ekw_cdf, "\n")
cat("EKw log-likelihood:", ll_ekw, "\n")

# Calculate gradient and Hessian
gr_ekw <- grekw(par_ekw, y_ekw)
hs_ekw <- hsekw(par_ekw, y_ekw)

cat("EKw gradient:\n")
print(gr_ekw)

cat("EKw Hessian (first 2x2):\n")
print(hs_ekw)
```

```
# Fit EKw model to data
fit_ekw <- gkwfit(data = y_ekw, family = "ekw")
summary(fit_ekw)

# Compare with true parameters
cat("True parameters: alpha=2.5, beta=1.5, lambda=2.0\n")
cat("Estimated parameters:\n")
print(coef(fit_ekw))

# ============================================================================
# EXAMPLE 12: Comprehensive Parameter Recovery Simulation
# ============================================================================
# Function to simulate data and recover parameters
simulate_and_recover <- function(family, true_params, n = 1000) {
  set.seed(2203)

  # Generate data based on family
  y <- switch(family,
    "gkw" = rgkw(n,
      alpha = true_params[1], beta = true_params[2],
      gamma = true_params[3], delta = true_params[4],
      lambda = true_params[5]
    ),
    "bkw" = rbkw(n,
      alpha = true_params[1], beta = true_params[2],
      gamma = true_params[3], delta = true_params[4]
    ),
    "kw" = rkw(n, alpha = true_params[1], beta = true_params[2]),
    "beta" = rbeta_(n, gamma = true_params[1], delta = true_params[2])
  )

  # Fit model
  fit <- gkwfit(data = y, family = family, silent = TRUE)

  # Return comparison
  list(
    family = family,
    true = true_params,
    estimated = coef(fit),
    loglik = fit$loglik,
    AIC = fit$AIC,
    converged = fit$convergence
  )
}

# Define true parameters for each family
params_gkw <- c(alpha = 2.0, beta = 3.0, gamma = 1.5, delta = 0.5, lambda = 1.2)
params_bkw <- c(alpha = 2.5, beta = 1.8, gamma = 1.2, delta = 0.3)
params_kw <- c(alpha = 1.5, beta = 2.0)
params_beta <- c(gamma = 2.0, delta = 3.0)

# Run simulations
result_gkw <- simulate_and_recover("gkw", params_gkw)
```

```
result_bkw <- simulate_and_recover("bkw", params_bkw)
result_kw <- simulate_and_recover("kw", params_kw)
result_beta <- simulate_and_recover("beta", params_beta)

# Create summary table
create_comparison_df <- function(result) {
  param_names <- names(result$true)
  df <- data.frame(
    Parameter = param_names,
    True = result$true,
    Estimated = result$estimated[param_names],
    Abs_Error = abs(result$true - result$estimated[param_names]),
    Rel_Error = abs((result$true - result$estimated[param_names]) / result$true) * 100
  )
  return(df)
}

# Print results
cat("===== GKw Parameter Recovery =====\n")
print(create_comparison_df(result_gkw))
cat("\n===== BKw Parameter Recovery =====\n")
print(create_comparison_df(result_bkw))
cat("\n===== Kw Parameter Recovery =====\n")
print(create_comparison_df(result_kw))
cat("\n===== Beta Parameter Recovery =====\n")
print(create_comparison_df(result_beta))

## End(Not run)
```

---

gkwfitall *Fit All or Selected Generalized Kumaraswamy Family Distributions and Compare Them*

Description

Fits all seven or a user-specified subset of distributions from the Generalized Kumaraswamy (GKw) family to data using maximum likelihood estimation through Template Model Builder (TMB). It provides a comprehensive comparison of fit quality across the selected families through statistics and visualization.

Usage

```
gkwfitall(
  data,
  family = NULL,
  method = "nlminb",
  use_moments = FALSE,
  profile = TRUE,
```

```
  npoints = 20,
  plot = TRUE,
  optimizer.control = list(),
  gof_tests = c("ks", "ad", "cvm"),
  theme_fn = ggplot2::theme_minimal,
  export_report = FALSE,
  report_file = "gkw_comparison_report.html"
)
```

Arguments

`data` A numeric vector with values strictly between 0 and 1.

`family` A character vector specifying which families to fit. Options are "gkw", "bkw", "kkw", "ekw", "mc", "kw", and "beta". If `NULL` (default), all seven distributions will be fitted.

`method` Optimization method to use. One of: `"nlminb"` (default), `"Nelder-Mead"`, `"BFGS"`, `"CG"`, `"L-BFGS-B"` or `"SANN"`.

`use_moments` Logical; if `TRUE`, attempts to use method of moments estimates as initial values. Default: `FALSE`.

`profile` Logical; if `TRUE`, computes likelihood profiles for parameters. Default: `TRUE`.

`npoints` Integer; number of points to use in profile likelihood calculations. Default: 20.

`plot` Logical; if `TRUE`, generates comparison plots. Default: `TRUE`.

`optimizer.control` List of control parameters passed to the chosen optimizer. Default: `list()`.

`gof_tests` Character vector specifying which goodness-of-fit tests to perform. Options are "ks" (Kolmogorov-Smirnov), "ad" (Anderson-Darling), and "cvm" (Cramer-von Mises). Default: c("ks", "ad", "cvm").

`theme_fn` Function to apply a custom theme to plots. Default: `ggplot2::theme_minimal`.

`export_report` Logical; if `TRUE`, generates an R Markdown report summarizing results. Default: `FALSE`.

`report_file` Character; file path for the R Markdown report if `export_report = TRUE`. Default: "gkw_comparison_report.html".

Details

This function fits all seven distributions in the GKw family:

- **GKw**: 5 parameters $(\alpha, \beta, \gamma, \delta, \lambda)$ - All positive.
- **BKw**: 4 parameters $(\alpha, \beta, \gamma, \delta)$, $\lambda = 1$ fixed - All positive.
- **KKw**: 4 parameters $(\alpha, \beta, \delta, \lambda)$, $\gamma = 1$ fixed - All positive.
- **EKw**: 3 parameters (α, β, λ), $\gamma = 1, \delta = 0$ fixed - All positive.
- **Mc** (McDonald / Beta Power): 3 parameters $(\gamma, \delta, \lambda)$, $\alpha = 1, \beta = 1$ fixed - All positive.
- **Kw** (Kumaraswamy): 2 parameters (α, β), $\gamma = 1, \delta = 0, \lambda = 1$ fixed - All positive.
- **Beta**: 2 parameters (γ, δ), $\alpha = 1, \beta = 1, \lambda = 1$ fixed - All positive.

The function generates comparison statistics including AIC, BIC, AICc, log-likelihood values, and various goodness-of-fit measures. It also produces visualizations with all fitted distributions overlaid on diagnostic plots.

Value

A list containing:

`fits`	A list of `gkwfit` objects for all fitted distribution families.
`comparison`	A data frame with comparison statistics (AIC, BIC, log-likelihood, etc.) for all models.
`plots`	A ggplot2 object with diagnostic plots for all models if `plot = TRUE`.
`metrics`	A list with additional comparative metrics including RMSE and MAE.

Author(s)

Lopes, J. E.

Examples

```
# Generate a sample dataset (n = 1000)
set.seed(123)
n <- 1000

# Create predictors
x1 <- runif(n, -2, 2)
x2 <- rnorm(n)
x3 <- factor(rbinom(n, 1, 0.4))

# Simulate Kumaraswamy distributed data
# True parameters with specific relationships to predictors
true_alpha <- exp(0.7 + 0.3 * x1)
true_beta <- exp(1.2 - 0.2 * x2 + 0.4 * (x3 == "1"))

# Generate random responses (assuming rkw function is available)
# If not available, use the beta distribution as an approximation

y <- rkw(n, alpha = true_alpha, beta = true_beta)

# Create data frame
df <- data.frame(y = y, x1 = x1, x2 = x2, x3 = x3)

# Split into training and test sets
set.seed(456)
train_idx <- sample(n, 800)
train_data <- df[train_idx, ]
test_data <- df[-train_idx, ]

# ====================================================================
# Example 1: Basic usage - Fit a Kumaraswamy model and make predictions
# ====================================================================
```

```
# Fit the model
kw_model <- gkwreg(y ~ x1 | x2 + x3, data = train_data, family = "kw")

# Predict mean response for test data
pred_mean <- predict(kw_model, newdata = test_data, type = "response")

# Calculate prediction error
mse <- mean((test_data$y - pred_mean)^2)
cat("Mean Squared Error:", mse, "\n")

# ======================================================================
# Example 2: Different prediction types
# ======================================================================

# Create a grid of values for visualization
x1_grid <- seq(-2, 2, length.out = 100)
grid_data <- data.frame(x1 = x1_grid, x2 = 0, x3 = 0)

# Predict different quantities
pred_mean <- predict(kw_model, newdata = grid_data, type = "response")
pred_var <- predict(kw_model, newdata = grid_data, type = "variance")
pred_params <- predict(kw_model, newdata = grid_data, type = "parameter")
pred_alpha <- predict(kw_model, newdata = grid_data, type = "alpha")
pred_beta <- predict(kw_model, newdata = grid_data, type = "beta")

# Plot predicted mean and parameters against x1
plot(x1_grid, pred_mean,
  type = "l", col = "blue",
  xlab = "x1", ylab = "Predicted Mean", main = "Mean Response vs x1"
)
plot(x1_grid, pred_var,
  type = "l", col = "red",
  xlab = "x1", ylab = "Predicted Variance", main = "Response Variance vs x1"
)
plot(x1_grid, pred_alpha,
  type = "l", col = "purple",
  xlab = "x1", ylab = "Alpha", main = "Alpha Parameter vs x1"
)
plot(x1_grid, pred_beta,
  type = "l", col = "green",
  xlab = "x1", ylab = "Beta", main = "Beta Parameter vs x1"
)

# ======================================================================
# Example 3: Computing densities, CDFs, and quantiles
# ======================================================================

# Select a single observation
obs_data <- test_data[1, ]

# Create a sequence of y values for plotting
y_seq <- seq(0.01, 0.99, length.out = 100)
```

```
# Compute density at each y value
dens_values <- predict(kw_model,
  newdata = obs_data,
  type = "density", at = y_seq, elementwise = FALSE
)

# Compute CDF at each y value
cdf_values <- predict(kw_model,
  newdata = obs_data,
  type = "probability", at = y_seq, elementwise = FALSE
)

# Compute quantiles for a sequence of probabilities
prob_seq <- seq(0.1, 0.9, by = 0.1)
quant_values <- predict(kw_model,
  newdata = obs_data,
  type = "quantile", at = prob_seq, elementwise = FALSE
)

# Plot density and CDF
plot(y_seq, dens_values,
  type = "l", col = "blue",
  xlab = "y", ylab = "Density", main = "Predicted PDF"
)
plot(y_seq, cdf_values,
  type = "l", col = "red",
  xlab = "y", ylab = "Cumulative Probability", main = "Predicted CDF"
)

# ======================================================================
# Example 4: Prediction under different distributional assumptions
# ======================================================================

# Fit models with different families
beta_model <- gkwreg(y ~ x1 | x2 + x3, data = train_data, family = "beta")
gkw_model <- gkwreg(y ~ x1 | x2 + x3 | 1 | 1 | x3, data = train_data, family = "gkw")

# Predict means using different families
pred_kw <- predict(kw_model, newdata = test_data, type = "response")
pred_beta <- predict(beta_model, newdata = test_data, type = "response")
pred_gkw <- predict(gkw_model, newdata = test_data, type = "response")

# Calculate MSE for each family
mse_kw <- mean((test_data$y - pred_kw)^2)
mse_beta <- mean((test_data$y - pred_beta)^2)
mse_gkw <- mean((test_data$y - pred_gkw)^2)

cat("MSE by family:\n")
cat("Kumaraswamy:", mse_kw, "\n")
cat("Beta:", mse_beta, "\n")
cat("GKw:", mse_gkw, "\n")
```

```
# Compare predictions from different families visually
plot(test_data$y, pred_kw,
  col = "blue", pch = 16,
  xlab = "Observed", ylab = "Predicted", main = "Predicted vs Observed"
)
points(test_data$y, pred_beta, col = "red", pch = 17)
points(test_data$y, pred_gkw, col = "green", pch = 18)
abline(0, 1, lty = 2)
legend("topleft",
  legend = c("Kumaraswamy", "Beta", "GKw"),
  col = c("blue", "red", "green"), pch = c(16, 17, 18)
)

# Compare models using AIC
# Note: AIC is applied to each model individually, not as a list
aic_values <- c(
  KW = AIC(kw_model),
  Beta = AIC(beta_model),
  GKw = AIC(gkw_model)
)

# Compare models using BIC
bic_values <- c(
  KW = BIC(kw_model),
  Beta = BIC(beta_model),
  GKw = BIC(gkw_model)
)

# Display model comparison results
model_comparison <- data.frame(
  Family = c("KW", "Beta", "GKw"),
  AIC = aic_values,
  BIC = bic_values,
  MSE = c(mse_kw, mse_beta, mse_gkw)
)
print(model_comparison[order(model_comparison$AIC), ])

# ============================================================================
# Example 5: Working with linear predictors and link functions
# ============================================================================

# Extract linear predictors and parameter values
lp <- predict(kw_model, newdata = test_data, type = "link")
params <- predict(kw_model, newdata = test_data, type = "parameter")

# Verify that inverse link transformation works correctly
# For Kumaraswamy model, alpha and beta use log links by default
alpha_from_lp <- exp(lp$alpha)
beta_from_lp <- exp(lp$beta)

# Compare with direct parameter predictions
cat("Manual inverse link vs direct parameter prediction:\n")
cat("Alpha difference:", max(abs(alpha_from_lp - params$alpha)), "\n")
```

```
cat("Beta difference:", max(abs(beta_from_lp - params$beta)), "\n")

# ======================================================================
# Example 6: Elementwise calculations
# ======================================================================

# Generate probabilities specific to each observation
probs <- runif(nrow(test_data), 0.1, 0.9)

# Calculate quantiles for each observation at its own probability level
quant_elementwise <- predict(kw_model,
  newdata = test_data,
  type = "quantile", at = probs, elementwise = TRUE
)

# Calculate probabilities at each observation's actual value
prob_at_y <- predict(kw_model,
  newdata = test_data,
  type = "probability", at = test_data$y, elementwise = TRUE
)

# Create Q-Q plot
plot(sort(prob_at_y), seq(0, 1, length.out = length(prob_at_y)),
  xlab = "Empirical Probability", ylab = "Theoretical Probability",
  main = "P-P Plot", type = "l"
)
abline(0, 1, lty = 2, col = "red")

# ======================================================================
# Example 7: Predicting for the original data
# ======================================================================

# Fit a model with original data
full_model <- gkwreg(y ~ x1 + x2 + x3 | x1 + x2 + x3, data = df, family = "kw")

# Get fitted values using predict and compare with model's fitted.values
fitted_from_predict <- predict(full_model, type = "response")
fitted_from_model <- full_model$fitted.values

# Compare results
cat(
  "Max difference between predict() and fitted.values:",
  max(abs(fitted_from_predict - fitted_from_model)), "\n"
)

# ======================================================================
# Example 8: Handling missing data
# ======================================================================

# Create test data with some missing values
test_missing <- test_data
test_missing$x1[1:5] <- NA
test_missing$x2[6:10] <- NA
```

```
# Predict with different na.action options
pred_na_pass <- try(
  predict(kw_model, newdata = test_missing, na.action = na.pass),
  silent = TRUE
)

if (!inherits(pred_na_pass, "try-error")) {
  # Show which positions have NAs
  cat("Rows with missing predictors:", which(is.na(pred_na_pass)), "\n")
} else {
 cat("na.pass resulted in an error. This is expected if the model requires complete cases.\n")
}

# Try with na.omit
pred_na_omit <- try(
  predict(kw_model, newdata = test_missing, na.action = na.omit),
  silent = TRUE
)

if (!inherits(pred_na_omit, "try-error")) {
  cat("Length after na.omit:", length(pred_na_omit), "\n")
  cat("(Original data had", nrow(test_missing), "rows)\n")
} else {
  cat("na.omit resulted in an error. Check model implementation.\n")
}

# ======================================================================
# Example 9: Simulating different distribution families
# ======================================================================

# Simulate data from different distributions
# Beta data
set.seed(123)
gamma_param <- 2
delta_param <- 5
y_beta <- rbeta_(1000, gamma_param, delta_param)

# Kumaraswamy data
set.seed(123)
alpha_param <- 1.5
beta_param <- 3.0
y_kw <- rkw(1000, alpha = alpha_param, beta = beta_param)

# Basic GKw data (using one approach if rgkw not available)
set.seed(123)
y_gkw <- rgkw(1000, alpha = 1.2, beta = 2.5, gamma = 1.8, delta = 0.6, lambda = 1.2)

# Create data frames with just the response
df_beta <- data.frame(y = y_beta)
df_kw <- data.frame(y = y_kw)
df_gkw <- data.frame(y = y_gkw)
```

```
# Fit models to each dataset
model_beta <- gkwreg(y ~ 1, data = df_beta, family = "beta")
model_kw <- gkwreg(y ~ 1, data = df_kw, family = "kw")
model_gkw <- gkwreg(y ~ 1, data = df_gkw, family = "gkw")

# Use predict to get densities at a grid of points
eval_points <- seq(0.01, 0.99, length.out = 100)

# Get densities
dens_beta <- predict(model_beta, type = "density", at = eval_points)
dens_kw <- predict(model_kw, type = "density", at = eval_points)
dens_gkw <- predict(model_gkw, type = "density", at = eval_points)

# Plot density comparisons
plot(eval_points, as.numeric(unique(dens_beta)),
  type = "l", col = "red",
  xlab = "y", ylab = "Density",
  main = "Fitted Density Functions for Different Distributions"
)
lines(eval_points, as.numeric(unique(dens_kw)), col = "blue", lty = 2)
lines(eval_points, as.numeric(unique(dens_gkw)), col = "green", lty = 3)
legend("topright",
  legend = c("Beta", "Kumaraswamy", "GKw"),
  col = c("red", "blue", "green"), lty = 1:3
)

# ======================================================================
# Example 10: Nested model comparison with different families
# ======================================================================

# Simulate data from GKw distribution
set.seed(123)
n <- 1000
x1 <- runif(n, -1, 1)

# First, simulate from a model with covariate effects
alpha <- exp(0.5 + 0.2 * x1)
beta <- exp(0.8 - 0.3 * x1)
gamma <- rep(1, n) # fixed (for Kw and KKw)
delta <- rep(0, n) # fixed (for Kw and EKw)
lambda <- rep(1, n) # fixed (for Kw and BKw)

# Generate Kumaraswamy data directly (since we fixed gamma=1, delta=0, lambda=1)
y <- rkw(n, alpha = alpha, beta = beta)

# Create data frame
sim_df <- data.frame(y = y, x1 = x1)

# Fit different family models
kw_model <- gkwreg(y ~ x1 | x1, data = sim_df, family = "kw")
beta_model <- gkwreg(y ~ x1 | x1, data = sim_df, family = "beta")
gkw_model <- gkwreg(y ~ x1 | x1 | 1 | 1 | 1, data = sim_df, family = "gkw")
```

```
# Compare AIC of the models (individually, not as a list)
aic_kw <- AIC(kw_model)
aic_beta <- AIC(beta_model)
aic_gkw <- AIC(gkw_model)

# Compare BIC of the models (individually, not as a list)
bic_kw <- BIC(kw_model)
bic_beta <- BIC(beta_model)
bic_gkw <- BIC(gkw_model)

# Create comparison table
model_comparison <- data.frame(
  Family = c("KW", "Beta", "GKw"),
  LogLik = c(logLik(kw_model), logLik(beta_model), logLik(gkw_model)),
  AIC = c(aic_kw, aic_beta, aic_gkw),
  BIC = c(bic_kw, bic_beta, bic_gkw)
)

# Display comparison sorted by AIC
print(model_comparison[order(model_comparison$AIC), ])

# Perform likelihood ratio test for nested models
# Function to perform LRT
lr_test <- function(full_model, reduced_model, df_diff) {
  lr_stat <- 2 * (as.numeric(logLik(full_model)) - as.numeric(logLik(reduced_model)))
  p_value <- 1 - pchisq(lr_stat, df = df_diff)
  return(c(statistic = lr_stat, p_value = p_value))
}

# Test if GKw is significantly better than Kw
# GKw has 3 more parameters: gamma, delta, lambda
lrt_gkw_kw <- lr_test(gkw_model, kw_model, df_diff = 3)

# Display LRT results
cat("\nLikelihood Ratio Test: GKw vs Kw\n")
cat("LR statistic:", round(lrt_gkw_kw["statistic"], 4), "\n")
cat("p-value:", format.pval(lrt_gkw_kw["p_value"]), "\n")
cat("Conclusion:", ifelse(lrt_gkw_kw["p_value"] < 0.05,
  "Reject H0: Full model (GKw) is better",
  "Fail to reject H0: Simpler model (Kw) is adequate"
), "\n")
```

`gkwgetstartvalues` *Main function to estimate GKw distribution parameters using the method of moments. This implementation is optimized for numerical stability and computational efficiency.*

Description

Main function to estimate GKw distribution parameters using the method of moments. This implementation is optimized for numerical stability and computational efficiency.

Usage

```
gkwgetstartvalues(x, n_starts = 5L)
```

Arguments

x	Data vector (must be in (0,1))
n_starts	Number of starting points for optimization

Value

Vector of estimated parameters $\alpha, \beta, \gamma, \delta, \lambda$

---

gkwgof *Comprehensive Goodness-of-Fit Analysis for GKw Family Distributions*

Description

Computes and displays a comprehensive set of goodness-of-fit statistics for distributions from the Generalized Kumaraswamy (GKw) family fitted using `gkwfit`. This function provides various measures including distance-based tests, information criteria, moment comparisons, probability plot metrics, and additional visualization tools for model adequacy assessment.

Usage

```
gkwgof(
  object,
  simulate_p_values = FALSE,
  n_bootstrap = 1000,
  plot = TRUE,
  print_summary = TRUE,
  verbose = FALSE,
  theme = ggplot2::theme_bw(),
  ncols = 4,
  title = NULL,
  ...
)
```

Arguments

`object` An object of class "gkwfit", typically the result of a call to `gkwfit`.

`simulate_p_values` Logical; if TRUE, uses parametric bootstrap to compute approximate p-values for distance-based tests. Default is FALSE.

`n_bootstrap` Number of bootstrap replicates for p-value simulation. Only used if `simulate_p_values = TRUE`. Default is 1000.

`plot` Logical; if TRUE, generates additional diagnostic plots beyond those already in the gkwfit object. Default is TRUE.

`print_summary` Logical; if TRUE, prints a formatted summary of the goodness-of-fit statistics. Default is TRUE.

`verbose` Logical; if TRUE, provides additional details and explanations about the test statistics. Default is FALSE.

`theme` A ggplot theme for all plots. default ggplot2::theme_bw()

`ncols` Number of columns to draw plots in graphics window.

`title` Plot title.

`...` Additional arguments to be passed to plotting functions.

Details

This function calculates the following goodness-of-fit statistics:

Distance-based tests:

- Kolmogorov-Smirnov (KS) statistic: Measures the maximum absolute difference between the empirical and theoretical CDFs.
- Cramer-von Mises (CvM) statistic: Measures the integrated squared difference between the empirical and theoretical CDFs.
- Anderson-Darling (AD) statistic: Similar to CvM but places more weight on the tails of the distribution.
- Watson (W^2) statistic: A modification of CvM that is location-invariant on the circle.

Information criteria:

- Akaike Information Criterion (AIC): $-2\log(L) + 2k$
- Bayesian Information Criterion (BIC): $-2\log(L) + k\log(n)$
- Corrected AIC (AICc): $AIC + \frac{2k(k+1)}{n-k-1}$
- Consistent AIC (CAIC): $-2\log(L) + k(\log(n) + 1)$
- Hannan-Quinn IC (HQIC): $-2\log(L) + 2k\log(\log(n))$

Moment-based comparisons:

- Theoretical vs. sample mean, variance, skewness, and kurtosis
- Standardized moment differences (relative to sample standard deviation)
- Root mean squared error of moments (RMSE)

Probability plot metrics:

- Correlation coefficient from P-P plot (closer to 1 indicates better fit)
- Area between P-P curve and diagonal line
- Mean absolute error in Q-Q plot

Likelihood statistics:

- Log-likelihood
- Log-likelihood per observation
- Pseudo-R^2 measure for bounded distributions

Prediction accuracy metrics:

- Mean Absolute Error (MAE) between empirical and theoretical CDF
- Root Mean Squared Error (RMSE) between empirical and theoretical CDF
- Continuous Ranked Probability Score (CRPS)

For model selection, lower values of information criteria (AIC, BIC, etc.) indicate better fit, while higher values of correlation coefficients and pseudo-R^2 indicate better fit. The distance-based tests are primarily used for hypothesis testing rather than model selection.

When `simulate_p_values = TRUE`, the function performs parametric bootstrap to compute approximate p-values for the distance-based tests, which accounts for parameter estimation uncertainty.

Value

An object of class "gkwgof" (inheriting from "list") containing the following components:

- `family`: The fitted distribution family
- `coefficients`: The estimated model parameters
- `sample_size`: The number of observations used in fitting
- `distance_tests`: Results from KS, CvM, AD, and Watson tests
- `information_criteria`: AIC, BIC, AICc, CAIC, and HQIC values
- `moments`: Theoretical moments, sample moments, and their differences
- `probability_plots`: Metrics from P-P and Q-Q plots
- `likelihood`: Log-likelihood statistics and pseudo-R^2 measure
- `prediction`: Prediction accuracy metrics
- `plots`: Additional diagnostic plots (if `plot = TRUE`)
- `p_values`: Simulated p-values (if `simulate_p_values = TRUE`)
- `bootstrap_stats`: Bootstrap distribution of test statistics (if `simulate_p_values = TRUE`)
- `call`: The matched call
- `gkwfit_object`: The original gkwfit object used for analysis

References

Anderson, T. W., & Darling, D. A. (1952). Asymptotic theory of certain "goodness of fit" criteria based on stochastic processes. Annals of Mathematical Statistics, 23(2), 193-212.

Burnham, K. P., & Anderson, D. R. (2002). Model Selection and Multimodel Inference: A Practical Information-Theoretic Approach (2nd ed.). Springer.

D'Agostino, R. B., & Stephens, M. A. (1986). Goodness-of-fit techniques. Marcel Dekker, Inc.

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

See Also

gkwfit, summary.gkwfit, print.gkwgof, plot.gkwgof, plotcompare

Examples

```
# Example 1: Simulate and analyze data from a Kumaraswamy (Kw) distribution
set.seed(2203) # Set seed for reproducibility
# Simulate 1000 observations from Kumaraswamy distribution with parameters alpha=2.5, beta=1.8
data_kw <- rkw(n = 1000, alpha = 2.5, beta = 1.8)
# Fit the Kumaraswamy distribution to the data
fit_kw <- gkwfit(data_kw, family = "kw")
# Basic goodness-of-fit analysis
gof_kw <- gkwgof(fit_kw)
# Analysis with bootstrap simulated p-values
gof_kw_bootstrap <- gkwgof(fit_kw, simulate_p_values = TRUE, n_bootstrap = 500)
# Detailed summary with additional explanations
gof_kw_verbose <- gkwgof(fit_kw, verbose = TRUE)

# Example 2: Comparing different distributions from the GKw family
# Simulate data from the Generalized Kumaraswamy (GKw) distribution
data_gkw <- rgkw(n = 1000, alpha = 2.0, beta = 1.5, gamma = 3.0, delta = 2.0, lambda = 1.2)
# Fit different models to the same data
fit_kw <- gkwfit(data_gkw, family = "kw") # Kumaraswamy model (simplified)
fit_bkw <- gkwfit(data_gkw, family = "bkw") # Beta-Kumaraswamy model
fit_ekw <- gkwfit(data_gkw, family = "ekw") # Exponentiated Kumaraswamy model
fit_gkw <- gkwfit(data_gkw, family = "gkw") # Full Generalized Kumaraswamy model
fit_beta <- gkwfit(data_gkw, family = "beta") # Standard Beta model
# Goodness-of-fit analysis for each model (without printing summaries)
gof_kw <- gkwgof(fit_kw, print_summary = FALSE)
gof_bkw <- gkwgof(fit_bkw, print_summary = FALSE)
gof_ekw <- gkwgof(fit_ekw, print_summary = FALSE)
gof_gkw <- gkwgof(fit_gkw, print_summary = FALSE)
gof_beta <- gkwgof(fit_beta, print_summary = FALSE)
# Information criteria comparison for model selection
ic_comparison <- data.frame(
  family = c("kw", "bkw", "ekw", "gkw", "beta"),
  n_params = c(
    length(gof_kw$coefficients),
    length(gof_bkw$coefficients),
    length(gof_ekw$coefficients),
```

```
    length(gof_gkw$coefficients),
    length(gof_beta$coefficients)
  ),
  logLik = c(
    gof_kw$likelihood$loglik,
    gof_bkw$likelihood$loglik,
    gof_ekw$likelihood$loglik,
    gof_gkw$likelihood$loglik,
    gof_beta$likelihood$loglik
  ),
  AIC = c(
    gof_kw$information_criteria$AIC,
    gof_bkw$information_criteria$AIC,
    gof_ekw$information_criteria$AIC,
    gof_gkw$information_criteria$AIC,
    gof_beta$information_criteria$AIC
  ),
  BIC = c(
    gof_kw$information_criteria$BIC,
    gof_bkw$information_criteria$BIC,
    gof_ekw$information_criteria$BIC,
    gof_gkw$information_criteria$BIC,
    gof_beta$information_criteria$BIC
  ),
  AICc = c(
    gof_kw$information_criteria$AICc,
    gof_bkw$information_criteria$AICc,
    gof_ekw$information_criteria$AICc,
    gof_gkw$information_criteria$AICc,
    gof_beta$information_criteria$AICc
  )
)
# Sort by AIC (lower is better)
ic_comparison <- ic_comparison[order(ic_comparison$AIC), ]
print(ic_comparison)

# Example 3: Comparative visualization
# Generate data from Beta distribution to demonstrate another case
set.seed(2203)
data_beta <- rbeta_(n = 1000, gamma = 2.5, delta = 1.5)
# Fit different distributions
fit_beta_true <- gkwfit(data_beta, family = "beta")
fit_kw_misspec <- gkwfit(data_beta, family = "kw")
fit_gkw_complex <- gkwfit(data_beta, family = "gkw")
# Goodness-of-fit analysis
gof_beta_true <- gkwgof(fit_beta_true, print_summary = FALSE, plot = FALSE)
gof_kw_misspec <- gkwgof(fit_kw_misspec, print_summary = FALSE, plot = FALSE)
gof_gkw_complex <- gkwgof(fit_gkw_complex, print_summary = FALSE, plot = FALSE)
# Comparative goodness-of-fit plot

plotcompare(
  list(
    "Beta (correct)" = gof_beta_true,
```

```
    "Kw (underspecified)" = gof_kw_misspec,
    "GKw (overspecified)" = gof_gkw_complex
  ),
  title = "Comparison of fits for Beta data"
)

# Example 5: Likelihood ratio tests for nested models
# Testing if the GKw distribution is significantly better than BKw (lambda = 1)
# Null hypothesis: lambda = 1 (BKw is adequate)
# Alternative hypothesis: lambda != 1 (GKw is necessary)
# Fitting nested models to data
nested_data <- rgkw(n = 1000, alpha = 2.0, beta = 1.5, gamma = 3.0, delta = 2.0, lambda = 1.0)
nested_fit_bkw <- gkwfit(nested_data, family = "bkw") # Restricted model (lambda = 1)
nested_fit_gkw <- gkwfit(nested_data, family = "gkw") # Unrestricted model

# Extracting log-likelihoods
ll_bkw <- nested_fit_bkw$loglik
ll_gkw <- nested_fit_gkw$loglik
# Calculating likelihood ratio test statistic
lr_stat <- 2 * (ll_gkw - ll_bkw)
# Calculating p-value (chi-square with 1 degree of freedom)
lr_pvalue <- 1 - pchisq(lr_stat, df = 1)
# Displaying test results
cat("Likelihood ratio test:\n")
cat("Test statistic:", round(lr_stat, 4), "\n")
cat("P-value:", format.pval(lr_pvalue), "\n")
cat("Conclusion:", ifelse(lr_pvalue < 0.05,
  "Reject H0 - GKw is necessary",
  "Fail to reject H0 - BKw is adequate"
), "\n")

# Example 6: Power simulation
# Checking the power of goodness-of-fit tests in detecting misspecifications
# Function to simulate power
simulate_power <- function(n_sim = 1000, n_obs = 1000, alpha_level = 0.05) {
  # Counters for rejections
  ks_rejections <- 0
  ad_rejections <- 0

  for (i in 1:n_sim) {
    # Simulating data from GKw, but fitting a Kw model (incorrect)
    sim_data <- rgkw(
      n = n_obs, alpha = 2.0, beta = 1.5,
      gamma = 3.0, delta = 2.0, lambda = 1.5
    )

    # Fitting incorrect model
    sim_fit_kw <- gkwfit(sim_data, family = "kw")

    # Calculating test statistics
    sim_gof <- gkwgof(sim_fit_kw,
      simulate_p_values = TRUE,
      n_bootstrap = 200, print_summary = FALSE, plot = FALSE
```

```
    )

    # Checking rejections in tests
    if (sim_gof$p_values$ks < alpha_level) ks_rejections <- ks_rejections + 1
    if (sim_gof$p_values$ad < alpha_level) ad_rejections <- ad_rejections + 1
  }

  # Calculating power
  power_ks <- ks_rejections / n_sim
  power_ad <- ad_rejections / n_sim

  return(list(
    power_ks = power_ks,
    power_ad = power_ad
  ))
}

# Run power simulation with reduced number of repetitions for example
power_results <- simulate_power(n_sim = 100)
cat("Estimated power (KS):", round(power_results$power_ks, 2), "\n")
cat("Estimated power (AD):", round(power_results$power_ad, 2), "\n")
```

---

gkwreg *Fit Generalized Kumaraswamy Regression Models*

Description

Fits regression models using the Generalized Kumaraswamy (GKw) family of distributions for response variables strictly bounded in the interval (0, 1). The function allows modeling parameters from all seven submodels of the GKw family as functions of predictors using appropriate link functions. Estimation is performed using Maximum Likelihood via the TMB (Template Model Builder) package. Requires the `Formula` and `TMB` packages.

Usage

```
gkwreg(
  formula,
  data,
  family = c("gkw", "bkw", "kkw", "ekw", "mc", "kw", "beta"),
  link = NULL,
  start = NULL,
  fixed = NULL,
  method = c("nlminb", "BFGS", "Nelder-Mead", "CG", "SANN", "L-BFGS-B"),
  hessian = TRUE,
  plot = TRUE,
  conf.level = 0.95,
  optimizer.control = list(),
```

```
  subset = NULL,
  weights = NULL,
  offset = NULL,
  na.action = getOption("na.action"),
  contrasts = NULL,
  x = FALSE,
  y = TRUE,
  model = TRUE,
  silent = TRUE,
  ...
)
```

Arguments

`formula` An object of class `Formula` (or one that can be coerced to that class). It should be structured as `y ~ model_alpha | model_beta | model_gamma | model_delta | model_lambda`, where `y` is the response variable and each `model_*` part specifies the linear predictor for the corresponding parameter (α, β, γ, δ, λ). If a part is omitted or specified as `~ 1` or `.`, an intercept-only model is used for that parameter. See Details for parameter correspondence in subfamilies.

`data` A data frame containing the variables specified in the `formula`.

`family` A character string specifying the desired distribution family. Defaults to `"gkw"`. Supported families are:

- `"gkw"`: Generalized Kumaraswamy (5 parameters: $\alpha, \beta, \gamma, \delta, \lambda$)
- `"bkw"`: Beta-Kumaraswamy (4 parameters: $\alpha, \beta, \gamma, \delta$; $\lambda = 1$ fixed)
- `"kkw"`: Kumaraswamy-Kumaraswamy (4 parameters: $\alpha, \beta, \delta, \lambda$; $\gamma = 1$ fixed)
- `"ekw"`: Exponentiated Kumaraswamy (3 parameters: α, β, λ; $\gamma = 1, \delta = 0$ fixed)
- `"mc"`: McDonald / Beta Power (3 parameters: γ, δ, λ; $\alpha = 1, \beta = 1$ fixed)
- `"kw"`: Kumaraswamy (2 parameters: α, β; $\gamma = 1, \delta = 0, \lambda = 1$ fixed)
- `"beta"`: Beta distribution (2 parameters: γ, δ; $\alpha = 1, \beta = 1, \lambda = 1$ fixed)

`link` Either a single character string specifying the same link function for all relevant parameters, or a named list specifying the link function for each modeled parameter (e.g., `list(alpha = "log", beta = "log", delta = "logit")`). Defaults are `"log"` for $\alpha, \beta, \gamma, \lambda$ (parameters > 0) and `"logit"` for δ (parameter in (0, 1)). Supported link functions are:

- `"log"`
- `"logit"`
- `"identity"`
- `"inverse"` (i.e., 1/mu)
- `"sqrt"`
- `"probit"`
- `"cloglog"` (complementary log-log)

start
: An optional named list providing initial values for the regression coefficients. Parameter names should match the distribution parameters (alpha, beta, etc.), and values should be vectors corresponding to the coefficients in the respective linear predictors (including intercept). If `NULL` (default), suitable starting values are automatically determined based on global parameter estimates.

fixed
: An optional named list specifying parameters or coefficients to be held fixed at specific values during estimation. Currently not fully implemented.

method
: Character string specifying the optimization algorithm to use. Options are `"nlminb"` (default, using `nlminb`), `"BFGS"`, `"Nelder-Mead"`, `"CG"`, `"SANN"`, or `"L-BFGS-B"`. If `"nlminb"` is selected, R's `nlminb` function is used; otherwise, R's `optim` function is used with the specified method.

hessian
: Logical. If `TRUE` (default), the Hessian matrix is computed via `sdreport` to obtain standard errors and the covariance matrix of the estimated coefficients. Setting to `FALSE` speeds up fitting but prevents calculation of standard errors and confidence intervals.

plot
: Logical. If `TRUE` (default), enables the generation of diagnostic plots when calling the generic `plot()` function on the fitted object. Actual plotting is handled by the `plot.gkwreg` method.

conf.level
: Numeric. The confidence level (1 - alpha) for constructing confidence intervals for the parameters. Default is 0.95. Used only if `hessian = TRUE`.

optimizer.control
: A list of control parameters passed directly to the chosen optimizer (`nlminb` or `optim`). See their respective documentation for details.

subset
: An optional vector specifying a subset of observations from `data` to be used in the fitting process.

weights
: An optional vector of prior weights (e.g., frequency weights) to be used in the fitting process. Should be `NULL` or a numeric vector of non-negative values.

offset
: An optional numeric vector or matrix specifying an a priori known component to be included *on the scale of the linear predictor* for each parameter. If a vector, it's applied to the predictor of the first parameter in the standard order (α). If a matrix, columns must correspond to parameters in the order $\alpha, \beta, \gamma, \delta, \lambda$.

na.action
: A function which indicates what should happen when the data contain `NA`s. The default (`na.fail`) stops if `NA`s are present. Other options include `na.omit` or `na.exclude`.

contrasts
: An optional list specifying the contrasts to be used for factor variables in the model. See the `contrasts.arg` of `model.matrix.default`.

x
: Logical. If `TRUE`, the list of model matrices (one for each modeled parameter) is returned as component `x` of the fitted object. Default `FALSE`.

y
: Logical. If `TRUE` (default), the response variable (after processing by `na.action`, `subset`) is returned as component `y`.

model
: Logical. If `TRUE` (default), the model frame (containing all variables used from `data`) is returned as component `model`.

silent
: Logical. If `TRUE` (default), suppresses progress messages from TMB compilation and optimization. Set to `FALSE` for verbose output.

...
: Additional arguments, currently unused or passed down to internal methods (potentially).

Details

The gkwreg function provides a regression framework for the Generalized Kumaraswamy (GKw) family and its submodels, extending density estimation to include covariates. The response variable must be strictly bounded in the (0, 1) interval.

Model Specification: The extended Formula syntax is crucial for specifying potentially different linear predictors for each relevant distribution parameter. The parameters $(\alpha, \beta, \gamma, \delta, \lambda)$ correspond sequentially to the parts of the formula separated by `|`. For subfamilies where some parameters are fixed by definition (see `family` argument), the corresponding parts of the formula are automatically ignored. For example, in a `family = "kw"` model, only the first two parts (for α and β) are relevant.

Parameter Constraints and Link Functions: The parameters $\alpha, \beta, \gamma, \lambda$ are constrained to be positive, while δ is constrained to the interval (0, 1). The default link functions (`"log"` for positive parameters, `"logit"` for δ) ensure these constraints during estimation. Users can specify alternative link functions suitable for the parameter's domain via the `link` argument.

Families and Parameters: The function automatically handles parameter fixing based on the chosen `family`:

- **GKw**: All 5 parameters $(\alpha, \beta, \gamma, \delta, \lambda)$ modeled.
- **BKw**: Models $\alpha, \beta, \gamma, \delta$; fixes $\lambda = 1$.
- **KKw**: Models $\alpha, \beta, \delta, \lambda$; fixes $\gamma = 1$.
- **EKw**: Models α, β, λ; fixes $\gamma = 1, \delta = 0$.
- **Mc** (McDonald): Models γ, δ, λ; fixes $\alpha = 1, \beta = 1$.
- **Kw** (Kumaraswamy): Models α, β; fixes $\gamma = 1, \delta = 0, \lambda = 1$.
- **Beta**: Models γ, δ; fixes $\alpha = 1, \beta = 1, \lambda = 1$. This parameterization corresponds to the standard Beta distribution with shape1 = γ and shape2 = δ.

Estimation Engine: Maximum Likelihood Estimation (MLE) is performed using C++ templates via the `TMB` package, which provides automatic differentiation and efficient optimization capabilities. The specific TMB template used depends on the chosen `family`.

Optimizer Method (`method` argument):

- `"nlminb"`: Uses R's built-in `stats::nlminb` optimizer. Good for problems with box constraints. Default option.
- `"Nelder-Mead"`: Uses R's `stats::optim` with the Nelder-Mead simplex algorithm, which doesn't require derivatives.
- `"BFGS"`: Uses R's `stats::optim` with the BFGS quasi-Newton method for unconstrained optimization.
- `"CG"`: Uses R's `stats::optim` with conjugate gradients method for unconstrained optimization.
- `"SANN"`: Uses R's `stats::optim` with simulated annealing, a global optimization method useful for problems with multiple local minima.
- `"L-BFGS-B"`: Uses R's `stats::optim` with the limited-memory BFGS method with box constraints.

Value

An object of class gkwreg. This is a list containing the following components:

- `call` The matched function call.
- `family` The specified distribution family string.
- `formula` The Formula object used.
- `coefficients` A named vector of estimated regression coefficients.
- `fitted.values` Vector of estimated means (expected values) of the response.
- `residuals` Vector of response residuals (observed - fitted mean).
- `fitted_parameters` List containing the estimated mean value for each distribution parameter $(\alpha, \beta, \gamma, \delta, \lambda)$.
- `parameter_vectors` List containing vectors of the estimated parameters $(\alpha, \beta, \gamma, \delta, \lambda)$ for each observation, evaluated on the link scale.
- `link` List of link functions used for each parameter.
- `param_names` Character vector of names of the parameters modeled by the family.
- `fixed_params` Named list indicating which parameters are fixed by the family definition.
- `loglik` The maximized log-likelihood value.
- `aic` Akaike Information Criterion.
- `bic` Bayesian Information Criterion.
- `deviance` The deviance (-2 * loglik).
- `df.residual` Residual degrees of freedom (nobs - npar).
- `nobs` Number of observations used in the fit.
- `npar` Total number of estimated parameters (coefficients).
- `vcov` The variance-covariance matrix of the coefficients (if `hessian = TRUE`).
- `se` Standard errors of the coefficients (if `hessian = TRUE`).
- `convergence` Convergence code from the optimizer (0 typically indicates success).
- `message` Convergence message from the optimizer.
- `iterations` Number of iterations used by the optimizer.
- `rmse` Root Mean Squared Error of response residuals.
- `efron_r2` Efron's pseudo R-squared.
- `mean_absolute_error` Mean Absolute Error of response residuals.
- `x` List of model matrices (if `x = TRUE`).
- `y` The response vector (if `y = TRUE`).
- `model` The model frame (if `model = TRUE`).
- `tmb_object` The raw object returned by MakeADFun.

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.

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

Ferrari, S. L. P., & Cribari-Neto, F. (2004). Beta regression for modelling rates and proportions. *Journal of Applied Statistics*, **31**(7), 799-815.

Kristensen, K., Nielsen, A., Berg, C. W., Skaug, H., & Bell, B. M. (2016). TMB: Automatic Differentiation and Laplace Approximation. *Journal of Statistical Software*, **70**(5), 1-21. (Underlying TMB package)

Zeileis, A., Kleiber, C., Jackman, S. (2008). Regression Models for Count Data in R. *Journal of Statistical Software*, **27**(8), 1-25.

Smithson, M., & Verkuilen, J. (2006). A Better Lemon Squeezer? Maximum-Likelihood Regression with Beta-Distributed Dependent Variables. *Psychological Methods*, **11**(1), 54–71.

See Also

summary.gkwreg, predict.gkwreg, plot.gkwreg, coef.gkwreg, vcov.gkwreg, logLik, AIC, Formula, MakeADFun, sdreport

Examples

```
## Example 1: Simple Kumaraswamy regression model ----
set.seed(123)
n <- 1000
x1 <- runif(n, -2, 2)
x2 <- rnorm(n)

# True regression coefficients
alpha_coef <- c(0.8, 0.3, -0.2) # Intercept, x1, x2
beta_coef <- c(1.2, -0.4, 0.1) # Intercept, x1, x2

# Generate linear predictors and transform to parameters using inverse link (exp)
eta_alpha <- alpha_coef[1] + alpha_coef[2] * x1 + alpha_coef[3] * x2
eta_beta <- beta_coef[1] + beta_coef[2] * x1 + beta_coef[3] * x2
alpha_true <- exp(eta_alpha)
beta_true <- exp(eta_beta)

# Generate responses from Kumaraswamy distribution (assuming rkw is available)
y <- rkw(n, alpha = alpha_true, beta = beta_true)
# Create data frame
df1 <- data.frame(y = y, x1 = x1, x2 = x2)

# Fit Kumaraswamy regression model using extended formula syntax
# Model alpha ~ x1 + x2 and beta ~ x1 + x2
kw_reg <- gkwreg(y ~ x1 + x2 | x1 + x2, data = df1, family = "kw", silent = TRUE)

# Display summary
summary(kw_reg)
```

```
## Example 2: Generalized Kumaraswamy regression ----
set.seed(456)
x1 <- runif(n, -1, 1)
x2 <- rnorm(n)
x3 <- factor(rbinom(n, 1, 0.5), labels = c("A", "B")) # Factor variable

# True regression coefficients
alpha_coef <- c(0.5, 0.2) # Intercept, x1
beta_coef <- c(0.8, -0.3, 0.1) # Intercept, x1, x2
gamma_coef <- c(0.6, 0.4) # Intercept, x3B
delta_coef <- c(0.0, 0.2) # Intercept, x3B (logit scale)
lambda_coef <- c(-0.2, 0.1) # Intercept, x2

# Design matrices
X_alpha <- model.matrix(~x1, data = data.frame(x1 = x1))
X_beta <- model.matrix(~ x1 + x2, data = data.frame(x1 = x1, x2 = x2))
X_gamma <- model.matrix(~x3, data = data.frame(x3 = x3))
X_delta <- model.matrix(~x3, data = data.frame(x3 = x3))
X_lambda <- model.matrix(~x2, data = data.frame(x2 = x2))

# Generate linear predictors and transform to parameters
alpha <- exp(X_alpha %*% alpha_coef)
beta <- exp(X_beta %*% beta_coef)
gamma <- exp(X_gamma %*% gamma_coef)
delta <- plogis(X_delta %*% delta_coef) # logit link for delta
lambda <- exp(X_lambda %*% lambda_coef)

# Generate response from GKw distribution (assuming rgkw is available)
y <- rgkw(n, alpha = alpha, beta = beta, gamma = gamma, delta = delta, lambda = lambda)

# Create data frame
df2 <- data.frame(y = y, x1 = x1, x2 = x2, x3 = x3)

# Fit GKw regression with parameter-specific formulas
# alpha ~ x1, beta ~ x1 + x2, gamma ~ x3, delta ~ x3, lambda ~ x2
gkw_reg <- gkwreg(y ~ x1 | x1 + x2 | x3 | x3 | x2, data = df2, family = "gkw")

# Compare true vs. estimated coefficients
print("Estimated Coefficients (GKw):")
print(coef(gkw_reg))
print("True Coefficients (approx):")
print(list(
  alpha = alpha_coef, beta = beta_coef, gamma = gamma_coef,
  delta = delta_coef, lambda = lambda_coef
))

## Example 3: Beta regression for comparison ----
set.seed(789)
x1 <- runif(n, -1, 1)

# True coefficients for Beta parameters (gamma = shape1, delta = shape2)
gamma_coef <- c(1.0, 0.5) # Intercept, x1 (log scale for shape1)
```

```
delta_coef <- c(1.5, -0.7) # Intercept, x1 (log scale for shape2)

# Generate linear predictors and transform (default link is log for Beta params here)
X_beta_eg <- model.matrix(~x1, data.frame(x1 = x1))
gamma_true <- exp(X_beta_eg %*% gamma_coef)
delta_true <- exp(X_beta_eg %*% delta_coef)

# Generate response from Beta distribution
y <- rbeta_(n, gamma_true, delta_true)

# Create data frame
df_beta <- data.frame(y = y, x1 = x1)

# Fit Beta regression model using gkwreg
# Formula maps to gamma and delta: y ~  x1 | x1
beta_reg <- gkwreg(y ~ x1 | x1,
  data = df_beta, family = "beta",
  link = list(gamma = "log", delta = "log")
) # Specify links if non-default

## Example 4: Model comparison using AIC/BIC ----
# Fit an alternative model, e.g., Kumaraswamy, to the same beta-generated data
kw_reg2 <- try(gkwreg(y ~ x1 | x1, data = df_beta, family = "kw"))

print("AIC Comparison (Beta vs Kw):")
c(AIC(beta_reg), AIC(kw_reg2))
print("BIC Comparison (Beta vs Kw):")
c(BIC(beta_reg), BIC(kw_reg2))

## Example 5: Predicting with a fitted model

# Use the Beta regression model from Example 3
newdata <- data.frame(x1 = seq(-1, 1, length.out = 20))

# Predict expected response (mean of the Beta distribution)
pred_response <- predict(beta_reg, newdata = newdata, type = "response")

# Predict parameters (gamma and delta) on the scale of the link function
pred_link <- predict(beta_reg, newdata = newdata, type = "link")

# Predict parameters on the original scale (shape1, shape2)
pred_params <- predict(beta_reg, newdata = newdata, type = "parameter")

# Plot original data and predicted mean response curve
plot(df_beta$x1, df_beta$y,
  pch = 20, col = "grey", xlab = "x1", ylab = "y",
  main = "Beta Regression Fit (using gkwreg)"
)
lines(newdata$x1, pred_response, col = "red", lwd = 2)
legend("topright", legend = "Predicted Mean", col = "red", lty = 1, lwd = 2)
```

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

```
# Assuming existence of rbeta_, llbeta, grbeta, hsbeta functions

# Generate sample data from a Beta(2, 4) distribution
# (gamma=2, delta=3 in this parameterization)
set.seed(123)
true_par_beta <- c(gamma = 2, delta = 3)
sample_data_beta <- rbeta_(100, gamma = true_par_beta[1], delta = true_par_beta[2])
hist(sample_data_beta, breaks = 20, main = "Beta(2, 4) Sample")

# --- Find MLE estimates ---
start_par_beta <- c(1.5, 2.5)
mle_result_beta <- stats::optim(par = start_par_beta,
                               fn = llbeta,
                               gr = grbeta, # Use analytical gradient
                               method = "L-BFGS-B",
                               lower = c(1e-6, 1e-6), # Bounds: gamma>0, delta>=0
                               hessian = TRUE,
                               data = sample_data_beta)

# --- Compare analytical gradient to numerical gradient ---
if (mle_result_beta$convergence == 0 &&
    requireNamespace("numDeriv", quietly = TRUE)) {

  mle_par_beta <- mle_result_beta$par
  cat("\nComparing Gradients for Beta at MLE estimates:\n")

  # Numerical gradient of llbeta
 num_grad_beta <- numDeriv::grad(func = llbeta, x = mle_par_beta, data = sample_data_beta)

  # Analytical gradient from grbeta
  ana_grad_beta <- grbeta(par = mle_par_beta, data = sample_data_beta)
```

```
  cat("Numerical Gradient (Beta):\n")
  print(num_grad_beta)
  cat("Analytical Gradient (Beta):\n")
  print(ana_grad_beta)

  # Check differences
  cat("Max absolute difference between Beta gradients:\n")
  print(max(abs(num_grad_beta - ana_grad_beta)))

} else {
  cat("\nSkipping Beta gradient comparison.\n")
}

# Example with Hessian comparison (if hsbeta exists)
if (mle_result_beta$convergence == 0 &&
    requireNamespace("numDeriv", quietly = TRUE) && exists("hsbeta")) {

 num_hess_beta <- numDeriv::hessian(func = llbeta, x = mle_par_beta, data = sample_data_beta)
  ana_hess_beta <- hsbeta(par = mle_par_beta, data = sample_data_beta)
  cat("\nMax absolute difference between Beta Hessians:\n")
  print(max(abs(num_hess_beta - ana_hess_beta)))

}
```

---

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.

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.

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).

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).

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.

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

```
# Assuming existence of rbkw, llbkw, grbkw, hsbkw functions for BKw

# Generate sample data
set.seed(123)
true_par_bkw <- c(alpha = 2, beta = 3, gamma = 1, delta = 0.5)
if (exists("rbkw")) {
  sample_data_bkw <- rbkw(100, alpha = true_par_bkw[1], beta = true_par_bkw[2],
                         gamma = true_par_bkw[3], delta = true_par_bkw[4])
} else {
  sample_data_bkw <- rgkw(100, alpha = true_par_bkw[1], beta = true_par_bkw[2],
                         gamma = true_par_bkw[3], delta = true_par_bkw[4], lambda = 1)
}
hist(sample_data_bkw, breaks = 20, main = "BKw(2, 3, 1, 0.5) Sample")

# --- Find MLE estimates ---
start_par_bkw <- c(1.5, 2.5, 0.8, 0.3)
mle_result_bkw <- stats::optim(par = start_par_bkw,
                               fn = llbkw,
                               gr = grbkw, # Use analytical gradient for BKw
                               method = "BFGS",
                               hessian = TRUE,
                               data = sample_data_bkw)

# --- Compare analytical gradient to numerical gradient ---
if (mle_result_bkw$convergence == 0 &&
    requireNamespace("numDeriv", quietly = TRUE)) {

  mle_par_bkw <- mle_result_bkw$par
  cat("\nComparing Gradients for BKw at MLE estimates:\n")

  # Numerical gradient of llbkw
  num_grad_bkw <- numDeriv::grad(func = llbkw, x = mle_par_bkw, data = sample_data_bkw)

  # Analytical gradient from grbkw
  ana_grad_bkw <- grbkw(par = mle_par_bkw, data = sample_data_bkw)

  cat("Numerical Gradient (BKw):\n")
  print(num_grad_bkw)
  cat("Analytical Gradient (BKw):\n")
  print(ana_grad_bkw)

  # Check differences
  cat("Max absolute difference between BKw gradients:\n")
  print(max(abs(num_grad_bkw - ana_grad_bkw)))

} else {
  cat("\nSkipping BKw gradient comparison.\n")
}

# --- Optional: Compare with relevant components of GKw gradient ---
# Requires grgkw function
```

```
if (mle_result_bkw$convergence == 0 && exists("grgkw")) {
  # Create 5-param vector for grgkw (insert lambda=1)
  mle_par_gkw_equiv <- c(mle_par_bkw[1:4], lambda = 1.0)
  ana_grad_gkw <- grgkw(par = mle_par_gkw_equiv, data = sample_data_bkw)
  # Extract components corresponding to alpha, beta, gamma, delta
  ana_grad_gkw_subset <- ana_grad_gkw[c(1, 2, 3, 4)]

  cat("\nComparison with relevant components of GKw gradient:\n")
  cat("Max absolute difference:\n")
  print(max(abs(ana_grad_bkw - ana_grad_gkw_subset))) # Should be very small
}
```

```
# Assuming existence of rbkw, llbkw, grbkw, hsbkw functions for BKw

# Generate sample data
set.seed(123)
true_par_bkw <- c(alpha = 2, beta = 3, gamma = 1, delta = 0.5)
if (exists("rbkw")) {
  sample_data_bkw <- rbkw(100, alpha = true_par_bkw[1], beta = true_par_bkw[2],
                         gamma = true_par_bkw[3], delta = true_par_bkw[4])
} else {
  sample_data_bkw <- rgkw(100, alpha = true_par_bkw[1], beta = true_par_bkw[2],
                         gamma = true_par_bkw[3], delta = true_par_bkw[4], lambda = 1)
}
hist(sample_data_bkw, breaks = 20, main = "BKw(2, 3, 1, 0.5) Sample")

# --- Find MLE estimates ---
start_par_bkw <- c(1.5, 2.5, 0.8, 0.3)
mle_result_bkw <- stats::optim(par = start_par_bkw,
                               fn = llbkw,
                               gr = grbkw, # Use analytical gradient for BKw
                               method = "BFGS",
                               hessian = TRUE,
                               data = sample_data_bkw)

# --- Compare analytical gradient to numerical gradient ---
if (mle_result_bkw$convergence == 0 &&
    requireNamespace("numDeriv", quietly = TRUE)) {

  mle_par_bkw <- mle_result_bkw$par
  cat("\nComparing Gradients for BKw at MLE estimates:\n")

  # Numerical gradient of llbkw
  num_grad_bkw <- numDeriv::grad(func = llbkw, x = mle_par_bkw, data = sample_data_bkw)

  # Analytical gradient from grbkw
  ana_grad_bkw <- grbkw(par = mle_par_bkw, data = sample_data_bkw)

  cat("Numerical Gradient (BKw):\n")
  print(num_grad_bkw)
```

```
  cat("Analytical Gradient (BKw):\n")
  print(ana_grad_bkw)

  # Check differences
  cat("Max absolute difference between BKw gradients:\n")
  print(max(abs(num_grad_bkw - ana_grad_bkw)))

} else {
  cat("\nSkipping BKw gradient comparison.\n")
}

# --- Optional: Compare with relevant components of GKw gradient ---
# Requires grgkw function
if (mle_result_bkw$convergence == 0 && exists("grgkw")) {
  # Create 5-param vector for grgkw (insert lambda=1)
  mle_par_gkw_equiv <- c(mle_par_bkw[1:4], lambda = 1.0)
  ana_grad_gkw <- grgkw(par = mle_par_gkw_equiv, data = sample_data_bkw)
  # Extract components corresponding to alpha, beta, gamma, delta
  ana_grad_gkw_subset <- ana_grad_gkw[c(1, 2, 3, 4)]

  cat("\nComparison with relevant components of GKw gradient:\n")
  cat("Max absolute difference:\n")
  print(max(abs(ana_grad_bkw - ana_grad_gkw_subset))) # Should be very small
}
```

---

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

```
# Assuming existence of rekw, llekw, grekw, hsekw functions for EKw

# Generate sample data
set.seed(123)
true_par_ekw <- c(alpha = 2, beta = 3, lambda = 0.5)
if (exists("rekw")) {
  sample_data_ekw <- rekw(100, alpha = true_par_ekw[1], beta = true_par_ekw[2],
                          lambda = true_par_ekw[3])
} else {
  sample_data_ekw <- rgkw(100, alpha = true_par_ekw[1], beta = true_par_ekw[2],
                          gamma = 1, delta = 0, lambda = true_par_ekw[3])
}
hist(sample_data_ekw, breaks = 20, main = "EKw(2, 3, 0.5) Sample")

# --- Find MLE estimates ---
start_par_ekw <- c(1.5, 2.5, 0.8)
mle_result_ekw <- stats::optim(par = start_par_ekw,
                               fn = llekw,
                               gr = grekw, # Use analytical gradient for EKw
                               method = "BFGS",
                               hessian = TRUE,
                               data = sample_data_ekw)

# --- Compare analytical gradient to numerical gradient ---
if (mle_result_ekw$convergence == 0 &&
    requireNamespace("numDeriv", quietly = TRUE)) {

  mle_par_ekw <- mle_result_ekw$par
  cat("\nComparing Gradients for EKw at MLE estimates:\n")

  # Numerical gradient of llekw
  num_grad_ekw <- numDeriv::grad(func = llekw, x = mle_par_ekw, data = sample_data_ekw)

  # Analytical gradient from grekw
  ana_grad_ekw <- grekw(par = mle_par_ekw, data = sample_data_ekw)

  cat("Numerical Gradient (EKw):\n")
  print(num_grad_ekw)
  cat("Analytical Gradient (EKw):\n")
  print(ana_grad_ekw)

  # Check differences
  cat("Max absolute difference between EKw gradients:\n")
  print(max(abs(num_grad_ekw - ana_grad_ekw)))

} else {
  cat("\nSkipping EKw gradient comparison.\n")
}

# Example with Hessian comparison (if hsekw exists)
if (mle_result_ekw$convergence == 0 &&
```

```
    requireNamespace("numDeriv", quietly = TRUE) && exists("hsekw")) {

  num_hess_ekw <- numDeriv::hessian(func = llekw, x = mle_par_ekw, data = sample_data_ekw)
  ana_hess_ekw <- hsekw(par = mle_par_ekw, data = sample_data_ekw)
  cat("\nMax absolute difference between EKw Hessians:\n")
  print(max(abs(num_hess_ekw - ana_hess_ekw)))

}
```

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

```
# Generate sample data from a known GKw distribution
set.seed(123)
true_par <- c(alpha = 2, beta = 3, gamma = 1.0, delta = 0.5, lambda = 0.5)
sample_data <- rgkw(100, alpha = true_par[1], beta = true_par[2],
                    gamma = true_par[3], delta = true_par[4], lambda = true_par[5])

# --- Use in Optimization (e.g., with optim using analytical gradient) ---
start_par <- c(1.5, 2.5, 1.2, 0.3, 0.6) # Initial guess

# Optimization using analytical gradient
mle_result_gr <- stats::optim(par = start_par,
```

```
                              fn = llgkw,  # Objective function (Negative LL)
                              gr = grgkw,  # Gradient function
                              method = "BFGS", # Method using gradient
                              hessian = TRUE,
                              data = sample_data)

if (mle_result_gr$convergence == 0) {
  print("Optimization with analytical gradient converged.")
  mle_par_gr <- mle_result_gr$par
  print("Estimated parameters:")
  print(mle_par_gr)
} else {
  warning("Optimization with analytical gradient failed!")
}

# --- Compare analytical gradient to numerical gradient ---
# Requires the 'numDeriv' package
if (requireNamespace("numDeriv", quietly = TRUE) && mle_result_gr$convergence == 0) {

  cat("\nComparing Gradients at MLE estimates:\n")

  # Numerical gradient of the negative log-likelihood function
  num_grad <- numDeriv::grad(func = llgkw, x = mle_par_gr, data = sample_data)

  # Analytical gradient (output of grgkw)
  ana_grad <- grgkw(par = mle_par_gr, data = sample_data)

  cat("Numerical Gradient:\n")
  print(num_grad)
  cat("Analytical Gradient:\n")
  print(ana_grad)

  # Check differences (should be small)
  cat("Max absolute difference between gradients:\n")
  print(max(abs(num_grad - ana_grad)))

} else {
  cat("\nSkipping gradient comparison (requires 'numDeriv' package or convergence).\n")
}
```

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

```
# Assuming existence of rkkw, llkkw, grkkw, hskkw functions for kkw

# Generate sample data
set.seed(123)
true_par_kkw <- c(alpha = 2, beta = 3, delta = 1.5, lambda = 0.5)
if (exists("rkkw")) {
  sample_data_kkw <- rkkw(100, alpha = true_par_kkw[1], beta = true_par_kkw[2],
                         delta = true_par_kkw[3], lambda = true_par_kkw[4])
} else {
  sample_data_kkw <- rgkw(100, alpha = true_par_kkw[1], beta = true_par_kkw[2],
                         gamma = 1, delta = true_par_kkw[3], lambda = true_par_kkw[4])
}

# --- Find MLE estimates ---
start_par_kkw <- c(1.5, 2.5, 1.0, 0.6)
mle_result_kkw <- stats::optim(par = start_par_kkw,
                               fn = llkkw,
                               gr = grkkw, # Use analytical gradient for kkw
                               method = "BFGS",
                               hessian = TRUE,
                               data = sample_data_kkw)

# --- Compare analytical gradient to numerical gradient ---
if (mle_result_kkw$convergence == 0 &&
    requireNamespace("numDeriv", quietly = TRUE)) {

  mle_par_kkw <- mle_result_kkw$par
  cat("\nComparing Gradients for kkw at MLE estimates:\n")

  # Numerical gradient of llkkw
  num_grad_kkw <- numDeriv::grad(func = llkkw, x = mle_par_kkw, data = sample_data_kkw)

  # Analytical gradient from grkkw
  ana_grad_kkw <- grkkw(par = mle_par_kkw, data = sample_data_kkw)
```

```
  cat("Numerical Gradient (kkw):\n")
  print(num_grad_kkw)
  cat("Analytical Gradient (kkw):\n")
  print(ana_grad_kkw)

  # Check differences
  cat("Max absolute difference between kkw gradients:\n")
  print(max(abs(num_grad_kkw - ana_grad_kkw)))

} else {
  cat("\nSkipping kkw gradient comparison.\n")
}

# --- Optional: Compare with relevant components of GKw gradient ---
# Requires grgkw function
if (mle_result_kkw$convergence == 0 && exists("grgkw")) {
  # Create 5-param vector for grgkw (insert gamma=1)
  mle_par_gkw_equiv <- c(mle_par_kkw[1:2], gamma = 1.0, mle_par_kkw[3:4])
  ana_grad_gkw <- grgkw(par = mle_par_gkw_equiv, data = sample_data_kkw)
  # Extract components corresponding to alpha, beta, delta, lambda
  ana_grad_gkw_subset <- ana_grad_gkw[c(1, 2, 4, 5)]

  cat("\nComparison with relevant components of GKw gradient:\n")
  cat("Max absolute difference:\n")
  print(max(abs(ana_grad_kkw - ana_grad_gkw_subset))) # Should be very small
}
```

---

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

```
# Assuming existence of rkw, llkw, grkw, hskw functions for Kw

# Generate sample data
set.seed(123)
true_par_kw <- c(alpha = 2, beta = 3)
```

```
sample_data_kw <- rkw(100, alpha = true_par_kw[1], beta = true_par_kw[2])
hist(sample_data_kw, breaks = 20, main = "Kw(2, 3) Sample")

# --- Find MLE estimates ---
start_par_kw <- c(1.5, 2.5)
mle_result_kw <- stats::optim(par = start_par_kw,
                              fn = llkw,
                              gr = grkw, # Use analytical gradient for Kw
                              method = "L-BFGS-B", # Recommended for bounds
                              lower = c(1e-6, 1e-6),
                              hessian = TRUE,
                              data = sample_data_kw)

# --- Compare analytical gradient to numerical gradient ---
if (mle_result_kw$convergence == 0 &&
    requireNamespace("numDeriv", quietly = TRUE)) {

  mle_par_kw <- mle_result_kw$par
  cat("\nComparing Gradients for Kw at MLE estimates:\n")

  # Numerical gradient of llkw
  num_grad_kw <- numDeriv::grad(func = llkw, x = mle_par_kw, data = sample_data_kw)

  # Analytical gradient from grkw
  ana_grad_kw <- grkw(par = mle_par_kw, data = sample_data_kw)

  cat("Numerical Gradient (Kw):\n")
  print(num_grad_kw)
  cat("Analytical Gradient (Kw):\n")
  print(ana_grad_kw)

  # Check differences
  cat("Max absolute difference between Kw gradients:\n")
  print(max(abs(num_grad_kw - ana_grad_kw)))

} else {
  cat("\nSkipping Kw gradient comparison.\n")
}

# Example with Hessian comparison (if hskw exists)
if (mle_result_kw$convergence == 0 &&
    requireNamespace("numDeriv", quietly = TRUE) && exists("hskw")) {

  num_hess_kw <- numDeriv::hessian(func = llkw, x = mle_par_kw, data = sample_data_kw)
  ana_hess_kw <- hskw(par = mle_par_kw, data = sample_data_kw)
  cat("\nMax absolute difference between Kw Hessians:\n")
  print(max(abs(num_hess_kw - ana_hess_kw)))

}
```

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

```
# Assuming existence of rmc, llmc, grmc, hsmc functions for Mc distribution

# Generate sample data
set.seed(123)
true_par_mc <- c(gamma = 2, delta = 3, lambda = 0.5)
sample_data_mc <- rmc(100, gamma = true_par_mc[1], delta = true_par_mc[2],
                      lambda = true_par_mc[3])
hist(sample_data_mc, breaks = 20, main = "Mc(2, 3, 0.5) Sample")

# --- Find MLE estimates ---
start_par_mc <- c(1.5, 2.5, 0.8)
mle_result_mc <- stats::optim(par = start_par_mc,
                              fn = llmc,
                              gr = grmc, # Use analytical gradient for Mc
                              method = "BFGS",
                              hessian = TRUE,
                              data = sample_data_mc)

# --- Compare analytical gradient to numerical gradient ---
if (mle_result_mc$convergence == 0 &&
    requireNamespace("numDeriv", quietly = TRUE)) {

  mle_par_mc <- mle_result_mc$par
  cat("\nComparing Gradients for Mc at MLE estimates:\n")

  # Numerical gradient of llmc
  num_grad_mc <- numDeriv::grad(func = llmc, x = mle_par_mc, data = sample_data_mc)

  # Analytical gradient from grmc
  ana_grad_mc <- grmc(par = mle_par_mc, data = sample_data_mc)

  cat("Numerical Gradient (Mc):\n")
  print(num_grad_mc)
```

```
  cat("Analytical Gradient (Mc):\n")
  print(ana_grad_mc)

  # Check differences
  cat("Max absolute difference between Mc gradients:\n")
  print(max(abs(num_grad_mc - ana_grad_mc)))

} else {
  cat("\nSkipping Mc gradient comparison.\n")
}

# Example with Hessian comparison (if hsmc exists)
if (mle_result_mc$convergence == 0 &&
    requireNamespace("numDeriv", quietly = TRUE) && exists("hsmc")) {

  num_hess_mc <- numDeriv::hessian(func = llmc, x = mle_par_mc, data = sample_data_mc)
  ana_hess_mc <- hsmc(par = mle_par_mc, data = sample_data_mc)
  cat("\nMax absolute difference between Mc Hessians:\n")
  print(max(abs(num_hess_mc - ana_hess_mc)))

}
```

---

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

```
# Assuming existence of rbeta_, llbeta, grbeta, hsbeta functions

# Generate sample data from a Beta(2, 4) distribution
# (gamma=2, delta=3 in this parameterization)
set.seed(123)
true_par_beta <- c(gamma = 2, delta = 3)
sample_data_beta <- rbeta_(100, gamma = true_par_beta[1], delta = true_par_beta[2])
hist(sample_data_beta, breaks = 20, main = "Beta(2, 4) Sample")

# --- Find MLE estimates ---
start_par_beta <- c(1.5, 2.5)
mle_result_beta <- stats::optim(par = start_par_beta,
                               fn = llbeta,
                               gr = if (exists("grbeta")) grbeta else NULL,
                               method = "L-BFGS-B",
                               lower = c(1e-6, 1e-6), # Bounds: gamma>0, delta>=0
                               hessian = TRUE, # Ask optim for numerical Hessian
                               data = sample_data_beta)

# --- Compare analytical Hessian to numerical Hessian ---
if (mle_result_beta$convergence == 0 &&
    requireNamespace("numDeriv", quietly = TRUE) &&
    exists("hsbeta")) {

  mle_par_beta <- mle_result_beta$par
  cat("\nComparing Hessians for Beta at MLE estimates:\n")

  # Numerical Hessian of llbeta
  num_hess_beta <- numDeriv::hessian(func = llbeta, x = mle_par_beta, data = sample_data_beta)

  # Analytical Hessian from hsbeta
  ana_hess_beta <- hsbeta(par = mle_par_beta, data = sample_data_beta)

  cat("Numerical Hessian (Beta):\n")
  print(round(num_hess_beta, 4))
  cat("Analytical Hessian (Beta):\n")
  print(round(ana_hess_beta, 4))

  # Check differences
  cat("Max absolute difference between Beta Hessians:\n")
  print(max(abs(num_hess_beta - ana_hess_beta)))

  # Optional: Use analytical Hessian for Standard Errors
  # tryCatch({
  #   cov_matrix_beta <- solve(ana_hess_beta) # ana_hess_beta is already Hessian of negative LL
  #   std_errors_beta <- sqrt(diag(cov_matrix_beta))
  #   cat("Std. Errors from Analytical Beta Hessian:\n")
  #   print(std_errors_beta)
  # }, error = function(e) {
  #   warning("Could not invert analytical Beta Hessian: ", e$message)
  # })
```

```
} else {
  cat("\nSkipping Beta Hessian comparison.\n")
  cat("Requires convergence, 'numDeriv' package, and function 'hsbeta'.\n")
}
```

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` ($\delta \ge 0$).
`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

```
# Assuming existence of rbkw, llbkw, grbkw, hsbkw functions for BKw

# Generate sample data
set.seed(123)
true_par_bkw <- c(alpha = 2, beta = 3, gamma = 1, delta = 0.5)
if (exists("rbkw")) {
  sample_data_bkw <- rbkw(100, alpha = true_par_bkw[1], beta = true_par_bkw[2],
                         gamma = true_par_bkw[3], delta = true_par_bkw[4])
} else {
  sample_data_bkw <- rgkw(100, alpha = true_par_bkw[1], beta = true_par_bkw[2],
                         gamma = true_par_bkw[3], delta = true_par_bkw[4], lambda = 1)
}
hist(sample_data_bkw, breaks = 20, main = "BKw(2, 3, 1, 0.5) Sample")

# --- Find MLE estimates ---
start_par_bkw <- c(1.5, 2.5, 0.8, 0.3)
```

```
mle_result_bkw <- stats::optim(par = start_par_bkw,
                               fn = llbkw,
                               gr = if (exists("grbkw")) grbkw else NULL,
                               method = "BFGS",
                               hessian = TRUE, # Ask optim for numerical Hessian
                               data = sample_data_bkw)

# --- Compare analytical Hessian to numerical Hessian ---
if (mle_result_bkw$convergence == 0 &&
    requireNamespace("numDeriv", quietly = TRUE) &&
    exists("hsbkw")) {

  mle_par_bkw <- mle_result_bkw$par
  cat("\nComparing Hessians for BKw at MLE estimates:\n")

  # Numerical Hessian of llbkw
 num_hess_bkw <- numDeriv::hessian(func = llbkw, x = mle_par_bkw, data = sample_data_bkw)

  # Analytical Hessian from hsbkw
  ana_hess_bkw <- hsbkw(par = mle_par_bkw, data = sample_data_bkw)

  cat("Numerical Hessian (BKw):\n")
  print(round(num_hess_bkw, 4))
  cat("Analytical Hessian (BKw):\n")
  print(round(ana_hess_bkw, 4))

  # Check differences
  cat("Max absolute difference between BKw Hessians:\n")
  print(max(abs(num_hess_bkw - ana_hess_bkw)))

  # Optional: Use analytical Hessian for Standard Errors
  # tryCatch({
  #   cov_matrix_bkw <- solve(ana_hess_bkw)
  #   std_errors_bkw <- sqrt(diag(cov_matrix_bkw))
  #   cat("Std. Errors from Analytical BKw Hessian:\n")
  #   print(std_errors_bkw)
  # }, error = function(e) {
  #   warning("Could not invert analytical BKw Hessian: ", e$message)
  # })

} else {
  cat("\nSkipping BKw Hessian comparison.\n")
  cat("Requires convergence, 'numDeriv' package, and function 'hsbkw'.\n")
}
```

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

```
# Assuming existence of rekw, llekw, grekw, hsekw functions for EKw

# Generate sample data
set.seed(123)
true_par_ekw <- c(alpha = 2, beta = 3, lambda = 0.5)
if (exists("rekw")) {
  sample_data_ekw <- rekw(100, alpha = true_par_ekw[1], beta = true_par_ekw[2],
                          lambda = true_par_ekw[3])
} else {
  sample_data_ekw <- rgkw(100, alpha = true_par_ekw[1], beta = true_par_ekw[2],
                         gamma = 1, delta = 0, lambda = true_par_ekw[3])
}
hist(sample_data_ekw, breaks = 20, main = "EKw(2, 3, 0.5) Sample")

# --- Find MLE estimates ---
start_par_ekw <- c(1.5, 2.5, 0.8)
mle_result_ekw <- stats::optim(par = start_par_ekw,
                               fn = llekw,
                               gr = if (exists("grekw")) grekw else NULL,
                               method = "BFGS",
                               hessian = TRUE, # Ask optim for numerical Hessian
                               data = sample_data_ekw)

# --- Compare analytical Hessian to numerical Hessian ---
if (mle_result_ekw$convergence == 0 &&
    requireNamespace("numDeriv", quietly = TRUE) &&
    exists("hsekw")) {

  mle_par_ekw <- mle_result_ekw$par
  cat("\nComparing Hessians for EKw at MLE estimates:\n")
```

```
  # Numerical Hessian of llekw
  num_hess_ekw <- numDeriv::hessian(func = llekw, x = mle_par_ekw, data = sample_data_ekw)

  # Analytical Hessian from hsekw
  ana_hess_ekw <- hsekw(par = mle_par_ekw, data = sample_data_ekw)

  cat("Numerical Hessian (EKw):\n")
  print(round(num_hess_ekw, 4))
  cat("Analytical Hessian (EKw):\n")
  print(round(ana_hess_ekw, 4))

  # Check differences
  cat("Max absolute difference between EKw Hessians:\n")
  print(max(abs(num_hess_ekw - ana_hess_ekw)))

  # Optional: Use analytical Hessian for Standard Errors
  # tryCatch({
  #   cov_matrix_ekw <- solve(ana_hess_ekw)
  #   std_errors_ekw <- sqrt(diag(cov_matrix_ekw))
  #   cat("Std. Errors from Analytical EKw Hessian:\n")
  #   print(std_errors_ekw)
  # }, error = function(e) {
  #   warning("Could not invert analytical EKw Hessian: ", e$message)
  # })

} else {
  cat("\nSkipping EKw Hessian comparison.\n")
  cat("Requires convergence, 'numDeriv' package, and function 'hsekw'.\n")
}
```

`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), gkwreg (if provides regression fitting), optim, hessian (for numerical Hessian comparison).

Examples

```
# Generate sample data from a known GKw distribution
set.seed(123)
true_par <- c(alpha = 2, beta = 3, gamma = 1.0, delta = 0.5, lambda = 0.5)
sample_data <- rgkw(100, alpha = true_par[1], beta = true_par[2],
                   gamma = true_par[3], delta = true_par[4], lambda = true_par[5])

# --- Find MLE estimates (e.g., using optim) ---
start_par <- c(1.5, 2.5, 1.2, 0.3, 0.6) # Initial guess
mle_result <- stats::optim(par = start_par,
                           fn = llgkw,
                           method = "BFGS",
                           hessian = TRUE, # Ask optim for numerical Hessian
                           data = sample_data)

if (mle_result$convergence == 0) {
  mle_par <- mle_result$par
  print("MLE parameters found:")
  print(mle_par)

  # --- Compare analytical Hessian to numerical Hessian ---
  # Requires the 'numDeriv' package
  if (requireNamespace("numDeriv", quietly = TRUE)) {

    cat("\nComparing Hessians at MLE estimates:\n")

    # Numerical Hessian from numDeriv applied to negative LL function
    num_hess <- numDeriv::hessian(func = llgkw, x = mle_par, data = sample_data)

    # Analytical Hessian (output of hsgkw)
    ana_hess <- hsgkw(par = mle_par, data = sample_data)

    cat("Numerical Hessian (from numDeriv):\n")
    print(round(num_hess, 4))
    cat("Analytical Hessian (from hsgkw):\n")
    print(round(ana_hess, 4))

    # Check differences (should be small)
    cat("Max absolute difference between Hessians:\n")
    print(max(abs(num_hess - ana_hess)))

    # Optional: Use analytical Hessian to estimate standard errors
    # Ensure Hessian is positive definite before inverting
    # fisher_info_analytic <- ana_hess # Hessian of negative LL
    # tryCatch({
    #   cov_matrix <- solve(fisher_info_analytic)
    #   std_errors <- sqrt(diag(cov_matrix))
```

```
    #   cat("Std. Errors from Analytical Hessian:\n")
    #   print(std_errors)
    # }, error = function(e) {
    #   warning("Could not invert analytical Hessian to get standard errors: ", e$message)
    # })

  } else {
    cat("\nSkipping Hessian comparison (requires 'numDeriv' package).\n")
  }
} else {
  warning("Optimization did not converge. Hessian comparison skipped.")
}
```

---

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

```
# Assuming existence of rkkw, llkkw, grkkw, hskkw functions for kkw

# Generate sample data
set.seed(123)
true_par_kkw <- c(alpha = 2, beta = 3, delta = 1.5, lambda = 0.5)
if (exists("rkkw")) {
  sample_data_kkw <- rkkw(100, alpha = true_par_kkw[1], beta = true_par_kkw[2],
                          delta = true_par_kkw[3], lambda = true_par_kkw[4])
} else {
```

```
  sample_data_kkw <- rgkw(100, alpha = true_par_kkw[1], beta = true_par_kkw[2],
                         gamma = 1, delta = true_par_kkw[3], lambda = true_par_kkw[4])
}

# --- Find MLE estimates ---
start_par_kkw <- c(1.5, 2.5, 1.0, 0.6)
mle_result_kkw <- stats::optim(par = start_par_kkw,
                               fn = llkkw,
                               gr = if (exists("grkkw")) grkkw else NULL,
                               method = "BFGS",
                               hessian = TRUE,
                               data = sample_data_kkw)

# --- Compare analytical Hessian to numerical Hessian ---
if (mle_result_kkw$convergence == 0 &&
    requireNamespace("numDeriv", quietly = TRUE) &&
    exists("hskkw")) {

  mle_par_kkw <- mle_result_kkw$par
  cat("\nComparing Hessians for kkw at MLE estimates:\n")

  # Numerical Hessian of llkkw
 num_hess_kkw <- numDeriv::hessian(func = llkkw, x = mle_par_kkw, data = sample_data_kkw)

  # Analytical Hessian from hskkw
  ana_hess_kkw <- hskkw(par = mle_par_kkw, data = sample_data_kkw)

  cat("Numerical Hessian (kkw):\n")
  print(round(num_hess_kkw, 4))
  cat("Analytical Hessian (kkw):\n")
  print(round(ana_hess_kkw, 4))

  # Check differences
  cat("Max absolute difference between kkw Hessians:\n")
  print(max(abs(num_hess_kkw - ana_hess_kkw)))

  # Optional: Use analytical Hessian for Standard Errors
  # tryCatch({
  #   cov_matrix_kkw <- solve(ana_hess_kkw)
  #   std_errors_kkw <- sqrt(diag(cov_matrix_kkw))
  #   cat("Std. Errors from Analytical kkw Hessian:\n")
  #   print(std_errors_kkw)
  # }, error = function(e) {
  #   warning("Could not invert analytical kkw Hessian: ", e$message)
  # })

} else {
  cat("\nSkipping kkw Hessian comparison.\n")
  cat("Requires convergence, 'numDeriv' package, and function 'hskkw'.\n")
}
```

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

```
# Assuming existence of rkw, llkw, grkw, hskw functions for Kw

# Generate sample data
set.seed(123)
true_par_kw <- c(alpha = 2, beta = 3)
sample_data_kw <- rkw(100, alpha = true_par_kw[1], beta = true_par_kw[2])
hist(sample_data_kw, breaks = 20, main = "Kw(2, 3) Sample")

# --- Find MLE estimates ---
start_par_kw <- c(1.5, 2.5)
mle_result_kw <- stats::optim(par = start_par_kw,
                              fn = llkw,
                              gr = if (exists("grkw")) grkw else NULL,
                              method = "L-BFGS-B",
                              lower = c(1e-6, 1e-6),
                              hessian = TRUE, # Ask optim for numerical Hessian
                              data = sample_data_kw)

# --- Compare analytical Hessian to numerical Hessian ---
if (mle_result_kw$convergence == 0 &&
    requireNamespace("numDeriv", quietly = TRUE) &&
    exists("hskw")) {

  mle_par_kw <- mle_result_kw$par
  cat("\nComparing Hessians for Kw at MLE estimates:\n")
```

```
  # Numerical Hessian of llkw
  num_hess_kw <- numDeriv::hessian(func = llkw, x = mle_par_kw, data = sample_data_kw)

  # Analytical Hessian from hskw
  ana_hess_kw <- hskw(par = mle_par_kw, data = sample_data_kw)

  cat("Numerical Hessian (Kw):\n")
  print(round(num_hess_kw, 4))
  cat("Analytical Hessian (Kw):\n")
  print(round(ana_hess_kw, 4))

  # Check differences
  cat("Max absolute difference between Kw Hessians:\n")
  print(max(abs(num_hess_kw - ana_hess_kw)))

  # Optional: Use analytical Hessian for Standard Errors
  # tryCatch({
  #   cov_matrix_kw <- solve(ana_hess_kw) # ana_hess_kw is already Hessian of negative LL
  #   std_errors_kw <- sqrt(diag(cov_matrix_kw))
  #   cat("Std. Errors from Analytical Kw Hessian:\n")
  #   print(std_errors_kw)
  # }, error = function(e) {
  #   warning("Could not invert analytical Kw Hessian: ", e$message)
  # })

} else {
  cat("\nSkipping Kw Hessian comparison.\n")
  cat("Requires convergence, 'numDeriv' package, and function 'hskw'.\n")
}
```

---

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

```
# Assuming existence of rmc, llmc, grmc, hsmc functions for Mc distribution

# Generate sample data
set.seed(123)
true_par_mc <- c(gamma = 2, delta = 3, lambda = 0.5)
sample_data_mc <- rmc(100, gamma = true_par_mc[1], delta = true_par_mc[2],
                      lambda = true_par_mc[3])
hist(sample_data_mc, breaks = 20, main = "Mc(2, 3, 0.5) Sample")

# --- Find MLE estimates ---
start_par_mc <- c(1.5, 2.5, 0.8)
mle_result_mc <- stats::optim(par = start_par_mc,
                              fn = llmc,
                              gr = if (exists("grmc")) grmc else NULL,
                              method = "BFGS",
                              hessian = TRUE, # Ask optim for numerical Hessian
                              data = sample_data_mc)

# --- Compare analytical Hessian to numerical Hessian ---
if (mle_result_mc$convergence == 0 &&
    requireNamespace("numDeriv", quietly = TRUE) &&
    exists("hsmc")) {

  mle_par_mc <- mle_result_mc$par
  cat("\nComparing Hessians for Mc at MLE estimates:\n")

  # Numerical Hessian of llmc
  num_hess_mc <- numDeriv::hessian(func = llmc, x = mle_par_mc, data = sample_data_mc)

  # Analytical Hessian from hsmc
  ana_hess_mc <- hsmc(par = mle_par_mc, data = sample_data_mc)

  cat("Numerical Hessian (Mc):\n")
  print(round(num_hess_mc, 4))
  cat("Analytical Hessian (Mc):\n")
  print(round(ana_hess_mc, 4))
```

```
  # Check differences (monitor sign consistency)
  cat("Max absolute difference between Mc Hessians:\n")
  print(max(abs(num_hess_mc - ana_hess_mc)))

  # Optional: Use analytical Hessian for Standard Errors
  # tryCatch({
  #   cov_matrix_mc <- solve(ana_hess_mc) # ana_hess_mc is already Hessian of negative LL
  #   std_errors_mc <- sqrt(diag(cov_matrix_mc))
  #   cat("Std. Errors from Analytical Mc Hessian:\n")
  #   print(std_errors_mc)
  # }, error = function(e) {
  #   warning("Could not invert analytical Mc Hessian: ", e$message)
  # })

} else {
  cat("\nSkipping Mc Hessian comparison.\n")
  cat("Requires convergence, 'numDeriv' package, and function 'hsmc'.\n")
}
```

`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

```
# Assuming existence of rbeta_, llbeta, grbeta, hsbeta functions

# Generate sample data from a Beta(2, 4) distribution
# (gamma=2, delta=3 in this parameterization)
set.seed(123)
true_par_beta <- c(gamma = 2, delta = 3)
sample_data_beta <- rbeta_(100, gamma = true_par_beta[1], delta = true_par_beta[2])
hist(sample_data_beta, breaks = 20, main = "Beta(2, 4) Sample")

# --- Maximum Likelihood Estimation using optim ---
# Initial parameter guess
start_par_beta <- c(1.5, 2.5)

# Perform optimization (minimizing negative log-likelihood)
# Use method="L-BFGS-B" for box constraints (params > 0 / >= 0)
```

```
mle_result_beta <- stats::optim(par = start_par_beta,
                               fn = llbeta, # Use the custom Beta neg-log-likelihood
                               method = "L-BFGS-B",
                               lower = c(1e-6, 1e-6), # Bounds: gamma>0, delta>=0
                               hessian = TRUE,
                               data = sample_data_beta)

# Check convergence and results
if (mle_result_beta$convergence == 0) {
  print("Optimization converged successfully.")
  mle_par_beta <- mle_result_beta$par
  print("Estimated Beta parameters (gamma, delta):")
  print(mle_par_beta)
  print("True Beta parameters (gamma, delta):")
  print(true_par_beta)
  cat(sprintf("MLE corresponds approx to Beta(%.2f, %.2f)\n",
      mle_par_beta[1], mle_par_beta[2] + 1))

} else {
  warning("Optimization did not converge!")
  print(mle_result_beta$message)
}

# --- Compare numerical and analytical derivatives (if available) ---
# Requires 'numDeriv' package and analytical functions 'grbeta', 'hsbeta'
if (mle_result_beta$convergence == 0 &&
    requireNamespace("numDeriv", quietly = TRUE) &&
    exists("grbeta") && exists("hsbeta")) {

  cat("\nComparing Derivatives at Beta MLE estimates:\n")

  # Numerical derivatives of llbeta
  num_grad_beta <- numDeriv::grad(func = llbeta, x = mle_par_beta, data = sample_data_beta)
  num_hess_beta <- numDeriv::hessian(func = llbeta, x = mle_par_beta, data = sample_data_beta)

  # Analytical derivatives (assuming they return derivatives of negative LL)
  ana_grad_beta <- grbeta(par = mle_par_beta, data = sample_data_beta)
  ana_hess_beta <- hsbeta(par = mle_par_beta, data = sample_data_beta)

  # Check differences
  cat("Max absolute difference between gradients:\n")
  print(max(abs(num_grad_beta - ana_grad_beta)))
  cat("Max absolute difference between Hessians:\n")
  print(max(abs(num_hess_beta - ana_hess_beta)))

} else {
   cat("\nSkipping derivative comparison for Beta.\n")
   cat("Requires convergence, 'numDeriv' pkg & functions 'grbeta', 'hsbeta'.\n")
}
```

`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

```
# Generate sample data from a known BKw distribution
set.seed(2203)
true_par_bkw <- c(alpha = 2.0, beta = 1.5, gamma = 1.5, delta = 0.5)
sample_data_bkw <- rbkw(1000, alpha = true_par_bkw[1], beta = true_par_bkw[2],
                         gamma = true_par_bkw[3], delta = true_par_bkw[4])
hist(sample_data_bkw, breaks = 20, main = "BKw(2, 1.5, 1.5, 0.5) Sample")

# --- Maximum Likelihood Estimation using optim ---
# Initial parameter guess
start_par_bkw <- c(1.8, 1.2, 1.1, 0.3)

# Perform optimization (minimizing negative log-likelihood)
mle_result_bkw <- stats::optim(par = start_par_bkw,
                               fn = llbkw, # Use the BKw neg-log-likelihood
                              method = "BFGS", # Needs parameters > 0, consider L-BFGS-B
                               hessian = TRUE,
                               data = sample_data_bkw)

# Check convergence and results
if (mle_result_bkw$convergence == 0) {
  print("Optimization converged successfully.")
  mle_par_bkw <- mle_result_bkw$par
  print("Estimated BKw parameters:")
  print(mle_par_bkw)
  print("True BKw parameters:")
  print(true_par_bkw)
} else {
  warning("Optimization did not converge!")
  print(mle_result_bkw$message)
}

# --- Compare numerical and analytical derivatives (if available) ---
# Requires 'numDeriv' package and analytical functions 'grbkw', 'hsbkw'
```

```
if (mle_result_bkw$convergence == 0 &&
    requireNamespace("numDeriv", quietly = TRUE) &&
    exists("grbkw") && exists("hsbkw")) {

  cat("\nComparing Derivatives at BKw MLE estimates:\n")

  # Numerical derivatives of llbkw
  num_grad_bkw <- numDeriv::grad(func = llbkw, x = mle_par_bkw, data = sample_data_bkw)
 num_hess_bkw <- numDeriv::hessian(func = llbkw, x = mle_par_bkw, data = sample_data_bkw)

  # Analytical derivatives (assuming they return derivatives of negative LL)
  ana_grad_bkw <- grbkw(par = mle_par_bkw, data = sample_data_bkw)
  ana_hess_bkw <- hsbkw(par = mle_par_bkw, data = sample_data_bkw)

  # Check differences
  cat("Max absolute difference between gradients:\n")
  print(max(abs(num_grad_bkw - ana_grad_bkw)))
  cat("Max absolute difference between Hessians:\n")
  print(max(abs(num_hess_bkw - ana_hess_bkw)))

} else {
   cat("\nSkipping derivative comparison for BKw.\n")
   cat("Requires convergence, 'numDeriv' package and functions 'grbkw', 'hsbkw'.\n")
}
```

---

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 $\mathbf{x} = (x_1, \ldots, x_n)$ 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

```
# Assuming existence of rekw, grekw, hsekw functions for EKw distribution

# Generate sample data from a known EKw distribution
set.seed(123)
true_par_ekw <- c(alpha = 2, beta = 3, lambda = 0.5)
# Use rekw if it exists, otherwise use rgkw with gamma=1, delta=0
if (exists("rekw")) {
  sample_data_ekw <- rekw(100, alpha = true_par_ekw[1], beta = true_par_ekw[2],
                          lambda = true_par_ekw[3])
} else {
  sample_data_ekw <- rgkw(100, alpha = true_par_ekw[1], beta = true_par_ekw[2],
                         gamma = 1, delta = 0, lambda = true_par_ekw[3])
}
hist(sample_data_ekw, breaks = 20, main = "EKw(2, 3, 0.5) Sample")

# --- Maximum Likelihood Estimation using optim ---
# Initial parameter guess
start_par_ekw <- c(1.5, 2.5, 0.8)

# Perform optimization (minimizing negative log-likelihood)
# Use method="L-BFGS-B" for box constraints if needed (all params > 0)
mle_result_ekw <- stats::optim(par = start_par_ekw,
                               fn = llekw, # Use the EKw neg-log-likelihood
                               method = "BFGS", # Or "L-BFGS-B" with lower=1e-6
                               hessian = TRUE,
                               data = sample_data_ekw)

# Check convergence and results
if (mle_result_ekw$convergence == 0) {
  print("Optimization converged successfully.")
  mle_par_ekw <- mle_result_ekw$par
  print("Estimated EKw parameters:")
  print(mle_par_ekw)
  print("True EKw parameters:")
  print(true_par_ekw)
} else {
  warning("Optimization did not converge!")
  print(mle_result_ekw$message)
}

# --- Compare numerical and analytical derivatives (if available) ---
# Requires 'numDeriv' package and analytical functions 'grekw', 'hsekw'
if (mle_result_ekw$convergence == 0 &&
    requireNamespace("numDeriv", quietly = TRUE) &&
    exists("grekw") && exists("hsekw")) {

  cat("\nComparing Derivatives at EKw MLE estimates:\n")

  # Numerical derivatives of llekw
  num_grad_ekw <- numDeriv::grad(func = llekw, x = mle_par_ekw, data = sample_data_ekw)
  num_hess_ekw <- numDeriv::hessian(func = llekw, x = mle_par_ekw, data = sample_data_ekw)
```

```
  # Analytical derivatives (assuming they return derivatives of negative LL)
  ana_grad_ekw <- grekw(par = mle_par_ekw, data = sample_data_ekw)
  ana_hess_ekw <- hsekw(par = mle_par_ekw, data = sample_data_ekw)

  # Check differences
  cat("Max absolute difference between gradients:\n")
  print(max(abs(num_grad_ekw - ana_grad_ekw)))
  cat("Max absolute difference between Hessians:\n")
  print(max(abs(num_hess_ekw - ana_hess_ekw)))

} else {
   cat("\nSkipping derivative comparison for EKw.\n")
   cat("Requires convergence, 'numDeriv' package and functions 'grekw', 'hsekw'.\n")
}
```

`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 $\mathbf{x} = (x_1, \ldots, x_n)$ 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

```
# Generate sample data from a known GKw distribution
set.seed(123)
true_par <- c(alpha = 2, beta = 3, gamma = 1.0, delta = 0.5, lambda = 0.5)
sample_data <- rgkw(100, alpha = true_par[1], beta = true_par[2],
                   gamma = true_par[3], delta = true_par[4], lambda = true_par[5])
hist(sample_data, breaks = 20, main = "Sample GKw Data")

# --- Maximum Likelihood Estimation using optim ---
# Initial parameter guess (can be crucial)
start_par <- c(1.5, 2.5, 1.2, 0.3, 0.6)

# Perform optimization (minimizing negative log-likelihood)
# Ensure data is passed correctly to llgkw
```

```
mle_result <- stats::optim(par = start_par,
                           fn = llgkw,
                           method = "BFGS", # Method supporting bounds might be safer
                           hessian = TRUE,
                           data = sample_data)

# Check convergence and results
if (mle_result$convergence == 0) {
  print("Optimization converged successfully.")
  mle_par <- mle_result$par
  print("Estimated parameters:")
  print(mle_par)
  print("True parameters:")
  print(true_par)

  # Standard errors from Hessian (optional)
  # fisher_info <- solve(mle_result$hessian) # Need positive definite Hessian
  # std_errors <- sqrt(diag(fisher_info))
  # print("Approximate Standard Errors:")
  # print(std_errors)

} else {
  warning("Optimization did not converge!")
  print(mle_result$message)
}

# --- Compare numerical and analytical derivatives (if available) ---
# Requires the 'numDeriv' package and analytical functions 'grgkw', 'hsgkw'
if (requireNamespace("numDeriv", quietly = TRUE) &&
    exists("grgkw") && exists("hsgkw") && mle_result$convergence == 0) {

  cat("\nComparing Derivatives at MLE estimates:\n")

  # Numerical derivatives
  num_grad <- numDeriv::grad(func = llgkw, x = mle_par, data = sample_data)
  num_hess <- numDeriv::hessian(func = llgkw, x = mle_par, data = sample_data)

  # Analytical derivatives (assuming they exist)
  # Note: grgkw/hsgkw might compute derivatives of log-likelihood,
  # while llgkw is negative log-likelihood. Adjust signs if needed.
  # Assuming grgkw/hsgkw compute derivatives of NEGATIVE log-likelihood here:
  ana_grad <- grgkw(par = mle_par, data = sample_data)
  ana_hess <- hsgkw(par = mle_par, data = sample_data)

  # Check differences (should be small if analytical functions are correct)
  cat("Difference between numerical and analytical gradient:\n")
  print(summary(abs(num_grad - ana_grad)))

  cat("Difference between numerical and analytical Hessian:\n")
  print(summary(abs(num_hess - ana_hess)))

} else {
   cat("\nSkipping derivative comparison.\n")
```

```
    cat("Requires 'numDeriv' package and functions 'grgkw', 'hsgkw'.\n")
}
```

---

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 $\mathbf{x} = (x_1, \ldots, x_n)$ 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

```
# Assuming existence of rkkw, grkkw, hskkw functions for kkw distribution

# Generate sample data from a known kkw distribution
set.seed(123)
true_par_kkw <- c(alpha = 2, beta = 3, delta = 1.5, lambda = 0.5)
# Use rkkw if it exists, otherwise use rgkw with gamma=1
if (exists("rkkw")) {
  sample_data_kkw <- rkkw(100, alpha = true_par_kkw[1], beta = true_par_kkw[2],
                         delta = true_par_kkw[3], lambda = true_par_kkw[4])
} else {
  sample_data_kkw <- rgkw(100, alpha = true_par_kkw[1], beta = true_par_kkw[2],
                         gamma = 1, delta = true_par_kkw[3], lambda = true_par_kkw[4])
}
hist(sample_data_kkw, breaks = 20, main = "kkw(2, 3, 1.5, 0.5) Sample")

# --- Maximum Likelihood Estimation using optim ---
# Initial parameter guess
start_par_kkw <- c(1.5, 2.5, 1.0, 0.6)

# Perform optimization (minimizing negative log-likelihood)
mle_result_kkw <- stats::optim(par = start_par_kkw,
                               fn = llkkw, # Use the kkw neg-log-likelihood
                               method = "BFGS",
                               hessian = TRUE,
                               data = sample_data_kkw)

# Check convergence and results
if (mle_result_kkw$convergence == 0) {
```

```
  print("Optimization converged successfully.")
  mle_par_kkw <- mle_result_kkw$par
  print("Estimated kkw parameters:")
  print(mle_par_kkw)
  print("True kkw parameters:")
  print(true_par_kkw)
} else {
  warning("Optimization did not converge!")
  print(mle_result_kkw$message)
}

# --- Compare numerical and analytical derivatives (if available) ---
# Requires 'numDeriv' package and analytical functions 'grkkw', 'hskkw'
if (mle_result_kkw$convergence == 0 &&
    requireNamespace("numDeriv", quietly = TRUE) &&
    exists("grkkw") && exists("hskkw")) {

  cat("\nComparing Derivatives at kkw MLE estimates:\n")

  # Numerical derivatives of llkkw
  num_grad_kkw <- numDeriv::grad(func = llkkw, x = mle_par_kkw, data = sample_data_kkw)
  num_hess_kkw <- numDeriv::hessian(func = llkkw, x = mle_par_kkw, data = sample_data_kkw)

  # Analytical derivatives (assuming they return derivatives of negative LL)
  ana_grad_kkw <- grkkw(par = mle_par_kkw, data = sample_data_kkw)
  ana_hess_kkw <- hskkw(par = mle_par_kkw, data = sample_data_kkw)

  # Check differences
  cat("Max absolute difference between gradients:\n")
  print(max(abs(num_grad_kkw - ana_grad_kkw)))
  cat("Max absolute difference between Hessians:\n")
  print(max(abs(num_hess_kkw - ana_hess_kkw)))

} else {
   cat("\nSkipping derivative comparison for kkw.\n")
   cat("Requires convergence, 'numDeriv' package and functions 'grkkw', 'hskkw'.\n")
}
```

`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 $\mathbf{x} = (x_1, \ldots, x_n)$ 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

```
# Assuming existence of rkw, grkw, hskw functions for Kw distribution

# Generate sample data from a known Kw distribution
set.seed(123)
true_par_kw <- c(alpha = 2, beta = 3)
sample_data_kw <- rkw(100, alpha = true_par_kw[1], beta = true_par_kw[2])
hist(sample_data_kw, breaks = 20, main = "Kw(2, 3) Sample")

# --- Maximum Likelihood Estimation using optim ---
# Initial parameter guess
start_par_kw <- c(1.5, 2.5)

# Perform optimization (minimizing negative log-likelihood)
# Use method="L-BFGS-B" for box constraints (params > 0)
mle_result_kw <- stats::optim(par = start_par_kw,
                              fn = llkw, # Use the Kw neg-log-likelihood
                              method = "L-BFGS-B",
                              lower = c(1e-6, 1e-6), # Lower bounds > 0
                              hessian = TRUE,
                              data = sample_data_kw)

# Check convergence and results
if (mle_result_kw$convergence == 0) {
  print("Optimization converged successfully.")
  mle_par_kw <- mle_result_kw$par
  print("Estimated Kw parameters:")
  print(mle_par_kw)
  print("True Kw parameters:")
  print(true_par_kw)
} else {
  warning("Optimization did not converge!")
  print(mle_result_kw$message)
}

# --- Compare numerical and analytical derivatives (if available) ---
# Requires 'numDeriv' package and analytical functions 'grkw', 'hskw'
if (mle_result_kw$convergence == 0 &&
    requireNamespace("numDeriv", quietly = TRUE) &&
    exists("grkw") && exists("hskw")) {

  cat("\nComparing Derivatives at Kw MLE estimates:\n")

  # Numerical derivatives of llkw
  num_grad_kw <- numDeriv::grad(func = llkw, x = mle_par_kw, data = sample_data_kw)
  num_hess_kw <- numDeriv::hessian(func = llkw, x = mle_par_kw, data = sample_data_kw)

  # Analytical derivatives (assuming they return derivatives of negative LL)
  ana_grad_kw <- grkw(par = mle_par_kw, data = sample_data_kw)
  ana_hess_kw <- hskw(par = mle_par_kw, data = sample_data_kw)

  # Check differences
```

```
  cat("Max absolute difference between gradients:\n")
  print(max(abs(num_grad_kw - ana_grad_kw)))
  cat("Max absolute difference between Hessians:\n")
  print(max(abs(num_hess_kw - ana_hess_kw)))

} else {
   cat("\nSkipping derivative comparison for Kw.\n")
   cat("Requires convergence, 'numDeriv' package and functions 'grkw', 'hskw'.\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

```
# Assuming existence of rmc, grmc, hsmc functions for Mc distribution

# Generate sample data from a known Mc distribution
set.seed(123)
true_par_mc <- c(gamma = 2, delta = 3, lambda = 0.5)
# Use rmc for data generation
sample_data_mc <- rmc(100, gamma = true_par_mc[1], delta = true_par_mc[2],
                      lambda = true_par_mc[3])
hist(sample_data_mc, breaks = 20, main = "Mc(2, 3, 0.5) Sample")

# --- Maximum Likelihood Estimation using optim ---
# Initial parameter guess
start_par_mc <- c(1.5, 2.5, 0.8)

# Perform optimization (minimizing negative log-likelihood)
mle_result_mc <- stats::optim(par = start_par_mc,
                              fn = llmc, # Use the Mc neg-log-likelihood
                              method = "BFGS", # Or "L-BFGS-B" with lower=1e-6
                              hessian = TRUE,
                              data = sample_data_mc)

# Check convergence and results
if (mle_result_mc$convergence == 0) {
  print("Optimization converged successfully.")
  mle_par_mc <- mle_result_mc$par
```

```
  print("Estimated Mc parameters:")
  print(mle_par_mc)
  print("True Mc parameters:")
  print(true_par_mc)
} else {
  warning("Optimization did not converge!")
  print(mle_result_mc$message)
}

# --- Compare numerical and analytical derivatives (if available) ---
# Requires 'numDeriv' package and analytical functions 'grmc', 'hsmc'
if (mle_result_mc$convergence == 0 &&
    requireNamespace("numDeriv", quietly = TRUE) &&
    exists("grmc") && exists("hsmc")) {

  cat("\nComparing Derivatives at Mc MLE estimates:\n")

  # Numerical derivatives of llmc
  num_grad_mc <- numDeriv::grad(func = llmc, x = mle_par_mc, data = sample_data_mc)
  num_hess_mc <- numDeriv::hessian(func = llmc, x = mle_par_mc, data = sample_data_mc)

  # Analytical derivatives (assuming they return derivatives of negative LL)
  ana_grad_mc <- grmc(par = mle_par_mc, data = sample_data_mc)
  ana_hess_mc <- hsmc(par = mle_par_mc, data = sample_data_mc)

  # Check differences
  cat("Max absolute difference between gradients:\n")
  print(max(abs(num_grad_mc - ana_grad_mc)))
  cat("Max absolute difference between Hessians:\n")
  print(max(abs(num_hess_mc - ana_hess_mc)))

} else {
   cat("\nSkipping derivative comparison for Mc.\n")
   cat("Requires convergence, 'numDeriv' package and functions 'grmc', 'hsmc'.\n")
}
```

`logLik.gkwfit` *Extract Log-Likelihood from a gkwfit Object*

Description

Extracts the maximized log-likelihood value from a model fitted by `gkwfit`. It returns an object of class `"logLik"`, which includes attributes for the degrees of freedom (`"df"`) and the number of observations (`"nobs"`) used in the fit.

Usage

```
## S3 method for class 'gkwfit'
logLik(object, ...)
```

Arguments

`object`	An object of class `"gkwfit"`, typically the result of a call to `gkwfit`.
`...`	Additional arguments (currently ignored).

Details

This method provides compatibility with standard R functions that operate on log-likelihood values, such as `AIC`, `BIC`, and likelihood ratio tests. It retrieves the log-likelihood stored during the model fitting process (in `object$loglik`) and attaches the required attributes (`object$df` for the number of estimated parameters and `object$nobs` for the number of observations).

Value

An object of class `"logLik"`. This is the numeric log-likelihood value with the following attributes:

`df`	The number of estimated parameters in the model (integer).
`nobs`	The number of observations used for fitting the model (integer).

Author(s)

Lopes, J. E.

See Also

`gkwfit`, `AIC`, `BIC`, `logLik`

Examples

```
# Generate data and fit two models
set.seed(2203)
y <- rgkw(50, alpha = 2, beta = 3, gamma = 1, delta = 0, lambda = 1.5)

fit1 <- gkwfit(data = y, family = "kkw", plot = FALSE) # KKw model
fit2 <- gkwfit(data = y, family = "ekw", plot = FALSE) # EKw model

# Extract log-likelihood values
ll1 <- logLik(fit1)
ll2 <- logLik(fit2)

print(ll1)
print(ll2)

# Use for likelihood ratio test
lr_stat <- -2 * (as.numeric(ll1) - as.numeric(ll2))
df_diff <- attr(ll1, "df") - attr(ll2, "df")
```

```
p_value <- pchisq(lr_stat, df = abs(df_diff), lower.tail = FALSE)

cat("LR statistic:", lr_stat, "\n")
cat("df:", df_diff, "\n")
cat("p-value:", p_value, "\n")
```

---

logLik.gkwreg — *Extract Log-Likelihood from a Generalized Kumaraswamy Regression Model*

Description

This function extracts the maximized log-likelihood value from a fitted Generalized Kumaraswamy (GKw) regression model object (class "gkwreg"). The result is returned as an object of class "logLik", which includes attributes for degrees of freedom and number of observations, suitable for use with model selection criteria like AIC and BIC.

Usage

```
## S3 method for class 'gkwreg'
logLik(object, ...)
```

Arguments

object	An object of class "gkwreg", typically the result of a call to gkwreg.
...	Additional arguments, currently ignored by this method.

Details

The log-likelihood value is typically computed during the model fitting process (e.g., by gkwreg) and stored within the resulting object. This method retrieves this stored value. If the value is not directly available, it attempts to calculate it from the stored deviance ($logLik = -deviance/2$).

The log-likelihood for a GKw family model with parameters θ is generally defined as the sum of the log-density contributions for each observation:

$$l(\theta|y) = \sum_{i=1}^{n} \log f(y_i; \alpha_i, \beta_i, \gamma_i, \delta_i, \lambda_i)$$

where $f(y; \dots)$ is the probability density function (PDF) of the specific distribution from the GKw family used in the model (determined by the `family` argument in `gkwreg`), and parameters $(\alpha_i, \dots, \lambda_i)$ may depend on covariates.

The function also extracts the number of estimated parameters (`df`) and the number of observations (`nobs`) used in the fit, storing them as attributes of the returned "logLik" object, which is essential for functions like AIC and BIC. It attempts to find `df` and `nobs` from various components within the `object` if they are not directly stored as `npar` and `nobs`.

Value

An object of class "logLik" representing the maximized log-likelihood value. It has the following attributes:

- df: (numeric) The number of estimated parameters in the model (coefficients).
- nobs: (numeric) The number of observations used for fitting the model.

Author(s)

Lopes, J. E.

See Also

gkwreg, AIC.gkwreg, BIC.gkwreg, logLik, AIC, BIC

Examples

```
# Assume 'df' exists with response 'y' and predictors 'x1', 'x2', 'x3'
# and that rkw() is available and data is appropriate (0 < y < 1).
set.seed(123)
n <- 100
x1 <- runif(n)
x2 <- rnorm(n)
x3 <- factor(rbinom(n, 1, 0.4))
alpha <- exp(0.5 + 0.2 * x1)
beta <- exp(1.0 - 0.1 * x2 + 0.3 * (x3 == "1"))
y <- rkw(n, alpha = alpha, beta = beta) # Placeholder if rkw not available
y <- pmax(pmin(y, 1 - 1e-7), 1e-7)
df <- data.frame(y = y, x1 = x1, x2 = x2, x3 = x3)

# Fit a Kumaraswamy regression model
kw_reg <- gkwreg(y ~ x1 | x2 + x3, data = df, family = "kw")

# Extract log-likelihood object
ll <- logLik(kw_reg)

# Print the log-likelihood value (with attributes)
print(ll)

# Access the value directly
ll_value <- as.numeric(ll)
print(ll_value)

# Get the number of parameters (degrees of freedom)
df_model <- attr(ll, "df")
print(paste("Number of parameters:", df_model))

# Get the number of observations
nobs_model <- attr(ll, "nobs")
print(paste("Number of observations:", nobs_model))

# Use with AIC/BIC
```

```
AIC(kw_reg)
BIC(kw_reg)
```

nrgkw *Enhanced Newton-Raphson Optimization for GKw Family Distributions*

Description

An industrial-strength implementation of maximum likelihood estimation (MLE) for the parameters of any distribution in the Generalized Kumaraswamy (GKw) family. This function incorporates multiple advanced numerical techniques including trust region methods, eigenvalue-based regularization, adaptive scaling, and sophisticated line search to ensure robust convergence even for challenging datasets or extreme parameter values.

Usage

```
nrgkw(
  start = NULL,
  data = as.numeric(c()),
  family = "gkw",
  tol = 1e-06,
  max_iter = 100L,
  verbose = FALSE,
  optimization_method = "trust-region",
  enforce_bounds = TRUE,
  min_param_val = 1e-05,
  max_param_val = 1e+05,
  adaptive_scaling = TRUE,
  use_stochastic_perturbation = TRUE,
  get_num_hess = FALSE,
  multi_start_attempts = 3L,
  eigenvalue_hessian_reg = TRUE,
  max_backtrack = 20L,
  initial_trust_radius = 1
)
```

Arguments

`start` A numeric vector containing initial values for the parameters. If NULL, automatic initialization is used based on the dataset characteristics. The length and order must correspond to the selected `family` (e.g., `c(alpha, beta, gamma, delta, lambda)` for "gkw"; `c(alpha, beta)` for "kw"; `c(gamma, delta)` for "beta").

`data` A numeric vector containing the observed data. All values must be strictly between 0 and 1.

family — A character string specifying the distribution family. One of "gkw", "bkw", "kkw", "ekw", "mc", "kw", or "beta". Default: "gkw".

tol — Convergence tolerance. The algorithm stops when the Euclidean norm of the gradient is below this value, or if relative changes in parameters or the negative log-likelihood are below this threshold across consecutive iterations. Default: 1e-6.

max_iter — Maximum number of iterations allowed. Default: 100.

verbose — Logical; if TRUE, prints detailed progress information during optimization, including iteration number, negative log-likelihood, gradient norm, and step adjustment details. Default: FALSE.

optimization_method — Character string specifying the optimization method: "trust-region" (default), "newton-raphson", or "hybrid" (starts with trust-region, switches to newton-raphson near convergence).

enforce_bounds — Logical; if TRUE, parameter values are constrained to stay within min_param_val, max_param_val (and $\delta \geq 0$) during optimization. Default: TRUE.

min_param_val — Minimum allowed value for parameters constrained to be strictly positive ($\alpha, \beta, \gamma, \lambda$). Default: 1e-5.

max_param_val — Maximum allowed value for all parameters. Default: 1e5.

adaptive_scaling — Logical; if TRUE, parameters are automatically scaled to improve numerical stability. Default: TRUE.

use_stochastic_perturbation — Logical; if TRUE, applies random perturbations when optimization stalls. Default: TRUE.

get_num_hess — Logical; if TRUE, computes and returns a numerical approximation of the Hessian at the solution. Default: FALSE.

multi_start_attempts — Integer specifying the number of different starting points to try if initial optimization fails to converge. Default: 3.

eigenvalue_hessian_reg — Logical; if TRUE, uses eigenvalue-based regularization for the Hessian matrix. Default: TRUE.

max_backtrack — Integer specifying the maximum number of backtracking steps in line search. Default: 20.

initial_trust_radius — Initial radius for trust region method. Default: 1.0.

Details

This enhanced algorithm provides robust parameter estimation for the Generalized Kumaraswamy distribution and its subfamilies. The function implements several advanced numerical optimization techniques to maximize the likelihood function reliably even in difficult cases.

The GKw Family of Distributions: The Generalized Kumaraswamy (GKw) distribution, introduced by Carrasco, Ferrari, and Cordeiro (2010), is a flexible five-parameter continuous distribution defined on the standard unit interval (0,1). Its probability density function is given by:

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

where $\alpha, \beta, \gamma, \lambda > 0$ and $\delta \geq 0$ are the model parameters, and $B(\gamma, \delta+1)$ is the beta function. The GKw distribution encompasses several important special cases:

- **GKw** (5 parameters): $\alpha, \beta, \gamma, \delta, \lambda$
- **BKw** (4 parameters): $\alpha, \beta, \gamma, \delta$ (with $\lambda = 1$)
- **KKw** (4 parameters): $\alpha, \beta, \delta, \lambda$ (with $\gamma = 1$)
- **EKw** (3 parameters): α, β, λ (with $\gamma = 1, \delta = 0$)
- **Mc** (3 parameters): γ, δ, λ (with $\alpha = 1, \beta = 1$)
- **Kw** (2 parameters): α, β (with $\gamma = 1, \delta = 0, \lambda = 1$)
- **Beta**(2 parameters): γ, δ (with $\alpha = 1, \beta = 1, \lambda = 1$)

Trust Region Method with Levenberg-Marquardt Algorithm: The trust region approach restricts parameter updates to a region where the quadratic approximation of the objective function is trusted to be accurate. This algorithm implements the Levenberg-Marquardt variant, which solves the subproblem:

$$\min_p m_k(p) = -\nabla\ell(\theta_k)^T p + \frac{1}{2}p^T H_k p$$
$$\text{subject to } \|p\| \leq \Delta_k$$

where $\nabla\ell(\theta_k)$ is the gradient, H_k is the Hessian, and Δ_k is the trust region radius at iteration k. The Levenberg-Marquardt approach adds a regularization parameter λ to the Hessian, solving:

$$(H_k + \lambda I)p = -\nabla\ell(\theta_k)$$

The parameter λ controls the step size and direction:

- When λ is large, the step approaches a scaled steepest descent direction.
- When λ is small, the step approaches the Newton direction.

The algorithm dynamically adjusts λ based on the agreement between the quadratic model and the actual function:

$$\rho_k = \frac{f(\theta_k) - f(\theta_k + p_k)}{m_k(0) - m_k(p_k)}$$

The trust region radius is updated according to:

- If $\rho_k < 0.25$, reduce the radius: $\Delta_{k+1} = 0.5\Delta_k$
- If $\rho_k > 0.75$ and $\|p_k\| \approx \Delta_k$, increase the radius: $\Delta_{k+1} = 2\Delta_k$
- Otherwise, leave the radius unchanged: $\Delta_{k+1} = \Delta_k$

The step is accepted if $\rho_k > \eta$ (typically $\eta = 0.1$).

Eigenvalue-Based Hessian Regularization: For the Newton-Raphson method to converge, the Hessian matrix must be positive definite. This algorithm uses eigendecomposition to create a positive definite approximation that preserves the Hessian's structure:

$$H = Q\Lambda Q^T$$

where Q contains the eigenvectors and Λ is a diagonal matrix of eigenvalues.

The regularized Hessian is constructed by:

$$\tilde{H} = Q\tilde{\Lambda}Q^T$$

where $\tilde{\Lambda}$ contains modified eigenvalues:

$$\tilde{\lambda}_i = \max(\lambda_i, \epsilon)$$

with ϵ being a small positive threshold (typically 10^{-6}).

This approach is superior to diagonal loading ($H + \lambda I$) as it:

- Preserves the eigenvector structure of the original Hessian
- Minimally modifies the eigenvalues needed to ensure positive definiteness
- Better maintains the directional information in the Hessian

Parameter Scaling for Numerical Stability: When parameters have widely different magnitudes, optimization can become numerically unstable. The adaptive scaling system transforms the parameter space to improve conditioning:

$$\theta_i^{scaled} = s_i \theta_i$$

where scaling factors s_i are determined by:

- For large parameters ($|\theta_i| > 100$): $s_i = 100/|\theta_i|$
- For small parameters ($0 < |\theta_i| < 0.01$): $s_i = 0.01/|\theta_i|$
- Otherwise: $s_i = 1$

The optimization is performed in the scaled space, with appropriate transformations for the gradient and Hessian:

$$\nabla\ell(\theta^{scaled})_i = \frac{1}{s_i}\nabla\ell(\theta)_i$$

$$H(\theta^{scaled})_{ij} = \frac{1}{s_i s_j}H(\theta)_{ij}$$

The final results are back-transformed to the original parameter space before being returned.

Line Search with Wolfe Conditions: The line search procedure ensures sufficient decrease in the objective function when taking a step in the search direction. The implementation uses Wolfe conditions which include both:

1. Sufficient decrease (Armijo) condition:

$$f(\theta_k + \alpha p_k) \leq f(\theta_k) + c_1 \alpha \nabla f(\theta_k)^T p_k$$

2. Curvature condition:

$$|\nabla f(\theta_k + \alpha p_k)^T p_k| \leq c_2 |\nabla f(\theta_k)^T p_k|$$

where $0 < c_1 < c_2 < 1$, typically $c_1 = 10^{-4}$ and $c_2 = 0.9$.

The step length α is determined using polynomial interpolation:

- First iteration: quadratic interpolation
- Subsequent iterations: cubic interpolation using function and derivative values

The cubic polynomial model has the form:

$$a\alpha^3 + b\alpha^2 + c\alpha + d$$

The algorithm computes coefficients from values at two points, then finds the minimizer of this polynomial subject to bounds to ensure convergence.

Adaptive Numerical Differentiation: When analytical derivatives are unreliable, the algorithm uses numerical differentiation with adaptive step sizes based on parameter magnitudes:

$$h_i = \max(h_{min}, \min(h_{base}, h_{base} \cdot |\theta_i|))$$

where h_{min} is a minimum step size (typically 10^{-8}), h_{base} is a base step size (typically 10^{-5}), and θ_i is the parameter value.

For computing diagonal Hessian elements, the central difference formula is used:

$$\frac{\partial^2 f}{\partial \theta_i^2} \approx \frac{f(\theta + h_i e_i) - 2f(\theta) + f(\theta - h_i e_i)}{h_i^2}$$

For mixed partial derivatives:

$$\frac{\partial^2 f}{\partial \theta_i \partial \theta_j} \approx \frac{f(\theta + h_i e_i + h_j e_j) - f(\theta + h_i e_i - h_j e_j) - f(\theta - h_i e_i + h_j e_j) + f(\theta - h_i e_i - h_j e_j)}{4h_i h_j}$$

The algorithm validates numerical differentiation by comparing with existing gradients and adaptively adjusts step sizes when discrepancies are detected.

Stochastic Perturbation: To escape flat regions or local minima, the algorithm implements controlled stochastic perturbation when progress stalls (detected by monitoring gradient norm changes):

$$\theta_i^{new} = \theta_i + \Delta\theta_i$$

where the perturbation $\Delta\theta_i$ combines:

- A directed component opposite to the gradient: $-\text{sign}(\nabla \ell_i) \cdot 0.05 \cdot |\theta_i|$
- A random noise component: $U(-0.05|\theta_i|, 0.05|\theta_i|)$

The perturbation is applied when:

- The relative change in gradient norm is below a threshold for several consecutive iterations
- The algorithm appears to be stuck in a non-optimal region

The perturbation is accepted only if it improves the objective function value.

Multi-Start Strategy: For particularly challenging optimization landscapes, the algorithm can employ multiple starting points:

- Initial values are generated using moment-based estimation and domain knowledge about each distribution family
- Each initial point is randomly perturbed to explore different regions of the parameter space
- Independent optimization runs are performed from each starting point
- The best result (based on likelihood value and convergence status) is returned

This approach increases the probability of finding the global optimum or a high-quality local optimum, particularly for complex models with many parameters.

Advanced Parameter Initialization: Intelligent starting values are critical for convergence in complex models. The algorithm uses data-driven initialization based on:

- Method of moments estimators for beta parameters:

$$\alpha + \beta = \frac{\bar{x}(1-\bar{x})}{s^2} - 1$$

$$\alpha = (\alpha + \beta)\bar{x}$$

- Transformations to Kumaraswamy parameters:

$$a_{Kw} = \sqrt{\alpha_{Beta}}$$

$$b_{Kw} = \sqrt{\beta_{Beta}}$$

- Adjustments based on data skewness (detected via mean relative to 0.5)
- Corrections based on data dispersion (range relative to (0,1) interval)

The transformations between beta and Kumaraswamy parameters leverage the similarities between these distributions while accounting for their parametric differences.

Hybrid Optimization Strategy: The algorithm can dynamically switch between trust region and Newton-Raphson methods based on optimization progress:

- Early iterations: trust region method for stability and global convergence properties
- Later iterations (when close to optimum): Newton-Raphson with line search for quadratic convergence rate

The switching criteria are based on iteration count and gradient norm, with additional logic to handle cases where one method consistently fails.

Value

A list object of class `gkw_fit` containing the following components:

`parameters`	A named numeric vector with the estimated parameters.
`loglik`	The maximized value of the log-likelihood function.
`iterations`	Number of iterations performed.
`converged`	Logical flag indicating whether the algorithm converged successfully.
`param_history`	A matrix where rows represent iterations and columns represent parameter values.

`loglik_history` A vector of log-likelihood values at each iteration.

`gradient` The gradient vector at the final parameter estimates.

`hessian` The analytical Hessian matrix at the final parameter estimates.

`std_errors` A named numeric vector of approximate standard errors for the parameters.

`aic` Akaike Information Criterion.

`bic` Bayesian Information Criterion.

`aicc` AIC with small sample correction.

`n` The sample size.

`status` A character string indicating the termination status.

`z_values` A named numeric vector of Z-statistics for parameter significance testing.

`p_values` A named numeric vector of two-sided p-values corresponding to the Z-statistics.

`param_names` A character vector of the parameter names.

`family` The distribution family used.

`optimization_method` The optimization method used.

`numeric_hessian` The numerically approximated Hessian at the solution (if requested).

`condition_number` The condition number of the final Hessian, a measure of parameter identifiability.

`scaling_factors` The scaling factors used for parameters (if adaptive scaling was enabled).

Warning

Although this implementation is highly robust, fitting complex distributions can still be challenging. For best results:

- Try multiple starting values if results seem suboptimal
- Examine diagnostic information carefully, especially condition numbers and standard errors
- Be cautious of parameter estimates at or very near boundaries
- Consider model simplification if convergence is consistently problematic
- For the full GKw model with 5 parameters, convergence may be sensitive to starting values
- High condition numbers (>1e6) may indicate parameter redundancy or weak identifiability

Author(s)

Enhanced by Lopes, J. E.

References

Carrasco, J. M. F., Ferrari, S. L. P., & Cordeiro, G. M. (2010). A new generalized Kumaraswamy distribution. arXiv preprint arXiv:1004.0911.

Nocedal, J., & Wright, S. J. (2006). Numerical Optimization (2nd ed.). Springer.

Conn, A. R., Gould, N. I. M., & Toint, P. L. (2000). Trust Region Methods. MPS-SIAM Series on Optimization.

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

Examples

```
# Generate sample data from a Beta(2,5) distribution for testing
set.seed(123)
sample_data <- rbeta_(200, 2, 5)

# Automatic initialization (recommended for beginners)
result_auto <- nrgkw(NULL, sample_data, family = "beta", verbose = FALSE)
print(result_auto$parameters)
print(result_auto$loglik)

# Compare different optimization methods
methods <- c("trust-region", "newton-raphson", "hybrid")
results <- list()

for (method in methods) {
  results[[method]] <- nrgkw(NULL, sample_data, family = "beta",
                              optimization_method = method)
  cat(sprintf("Method: %s, AIC: %.4f\n", method, results[[method]]$aic))
}

# Fit the full GKw model with diagnostic information
gkw_result <- nrgkw(NULL, sample_data, family = "gkw",
                      verbose = FALSE, get_num_hess = TRUE)

# Examine parameter identifiability through condition number
cat(sprintf("Condition number: %.2e\n", gkw_result$condition_number))
print(gkw_result)

# Compare with simpler models using information criteria
cat("Information criteria comparison:\n")
cat(sprintf("GKw: AIC=%.4f, BIC=%.4f\n", gkw_result$aic, gkw_result$bic))
cat(sprintf("Beta: AIC=%.4f, BIC=%.4f\n",
           results[["trust-region"]]$aic, results[["trust-region"]]$bic))
```

---

pbeta_ *CDF of the Beta Distribution (gamma, delta+1 Parameterization)*

Description

Computes the cumulative distribution function (CDF), $F(q) = P(X \le q)$, 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 $\delta \ge 0$ 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 $P(X \le q)$, 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 $P(X \leq q)$.

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 $P(X \leq q)$, 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 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 $P(X \le q)$, 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), $P(X \le q)$, 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 $P(X \le q)$, 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 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)
```

`plot.gkwfit` *Plot Diagnostics for a gkwfit Object*

Description

Creates a panel of diagnostic plots for assessing the fit of a model estimated by `gkwfit`. It displays a histogram of the data overlaid with the fitted density, a Probability-Probability (P-P) plot, a Quantile-Quantile (Q-Q) plot, and profile likelihood plots for each parameter if they were computed during the fit (i.e., if `profile = TRUE` was used in `gkwfit`).

Usage

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

Arguments

x	An object of class "gkwfit", typically the result of a call to gkwfit.
...	Additional arguments (currently ignored).

Details

This function utilizes ggplot2 for creating the plots and patchwork for arranging them into a single figure. All plots use ggplot2::theme_minimal().

If the plots were already generated during the original gkwfit call (because plot = TRUE), they are retrieved from the fitted object. Otherwise, this function will attempt to generate the plots on the fly, which requires the ggplot2 package and the necessary distribution functions (like dgkw, pgkw, qgkw, etc.) for the specific family to be available.

The arrangement of plots is handled automatically by patchwork::wrap_plots. No user interaction (like menu selection) is required.

Value

Invisibly returns the original input object x. This function is called for its side effect of producing a plot.

Author(s)

Lopes, J. E.

See Also

gkwfit, summary.gkwfit

Examples

```
# Load required package
library(ggplot2)

# Generate data and fit model
set.seed(2203)
y <- rbeta_(50, gamma = 2, delta = 3)
fit <- gkwfit(data = y, family = "beta", plot = FALSE)

# Generate standard diagnostic plots
plot(fit)

# Generate data and fit model with profile = TRUE
fit <- gkwfit(data = y, family = "gkw", profile = TRUE, npoints = 15)

# Standard diagnostic plots
```

```
plot(fit)
```

---

plot.gkwfitall *Plot method for gkwfitall objects*

Description

Plot method for gkwfitall objects

Usage

```
## S3 method for class 'gkwfitall'
plot(x, which = "all", theme_fn = ggplot2::theme_minimal, ...)
```

Arguments

x	An object of class "gkwfitall"
which	Character vector specifying which plots to show. Options are "all" (default), "density", "pp", "qq", "residuals", "aic", or "parameters".
theme_fn	Function to apply a custom theme to plots. Default: `ggplot2::theme_minimal`.
...	Additional arguments passed to plotting functions

Value

Invisibly returns the input object

Author(s)

Lopes, J. E.

---

plot.gkwgof *Plot Method for gkwgof Objects*

Description

Creates a panel of diagnostic plots for assessing the goodness-of-fit of a model from the GKw family of distributions.

Usage

```
## S3 method for class 'gkwgof'
plot(x, title = NULL, ncols = 4, which = NULL, ...)
```

Arguments

- `x` — An object of class "gkwgof", typically the result of a call to `gkwgof`.
- `title` — Plot title
- `ncols` — Number of columns to draw plots in graphics window
- `which` — A numeric vector specifying which plots to display. If NULL (default), all available plots will be displayed.
- `...` — Additional arguments to be passed to plotting functions.

Value

The input object `x` is returned invisibly.

---

plot.gkwreg — *Diagnostic Plots for Generalized Kumaraswamy Regression Models*

Description

Produces a set of diagnostic plots for assessing the adequacy of a fitted Generalized Kumaraswamy (GKw) regression model (objects of class "gkwreg"). Options allow selection of specific plots, choice of residual type, and plotting using either base R graphics or `ggplot2`.

Usage

```
## S3 method for class 'gkwreg'
plot(
  x,
  which = 1:6,
  caption = c("Residuals vs. Observation Indices", "Cook's Distance Plot",
    "Generalized Leverage vs. Fitted Values", "Residuals vs. Linear Predictor",
    "Half-Normal Plot of Residuals", "Predicted vs. Observed Values"),
  sub.caption = paste(deparse(x$call), collapse = "\n"),
  main = "",
  ask = prod(par("mfcol")) < length(which) && dev.interactive(),
  ...,
  type = c("quantile", "pearson", "deviance"),
  family = NULL,
  nsim = 100,
  level = 0.9,
  use_ggplot = FALSE,
  arrange_plots = FALSE,
  sample_size = NULL,
  theme_fn = ggplot2::theme_minimal,
  save_diagnostics = FALSE
)
```

Arguments

- `x` — An object of class "gkwreg", typically the result of a call to gkwreg.
- `which` — Integer vector specifying which diagnostic plots to produce. If a subset of the plots is required, specify a subset of the numbers 1:6. Defaults to 1:6. The plots correspond to:
  1. Residuals vs. Observation Indices: Checks for time trends or patterns.
  2. Cook's Distance Plot: Helps identify influential observations.
  3. Generalized Leverage vs. Fitted Values: Identifies points with high leverage.
  4. Residuals vs. Linear Predictor: Checks for non-linearity and heteroscedasticity.
  5. Half-Normal Plot of Residuals (with simulated envelope): Assesses normality of residuals, comparing against simulated quantiles.
  6. Predicted vs. Observed Values: Checks overall model prediction accuracy.
- `caption` — Character vector providing captions (titles) for the plots. Its length must be at least max(which). Defaults are provided for plots 1-6.
- `sub.caption` — Character string used as a common subtitle positioned above all plots (especially when multiple plots are arranged). Defaults to the deparsed model call.
- `main` — An optional character string to be prepended to the individual plot captions (from the caption argument).
- `ask` — Logical. If TRUE (and using base R graphics with multiple plots on an interactive device), the user is prompted before displaying each plot. Defaults to TRUE if more plots are requested than fit on the current screen layout.
- `...` — Additional arguments passed to the underlying plotting functions (e.g., graphical parameters like col, pch, cex for base R plots).
- `type` — Character string indicating the type of residuals to be used for plotting. Defaults to "quantile". Valid options are:
  - "quantile": Randomized quantile residuals (Dunn & Smyth, 1996). Recommended for bounded responses as they should be approximately N(0,1) if the model is correctly specified.
  - "pearson": Pearson residuals (response residual standardized by estimated standard deviation). Useful for checking the variance function.
  - "deviance": Deviance residuals. Related to the log-likelihood contribution of each observation.
- `family` — Character string specifying the distribution family assumptions to use when calculating residuals and other diagnostics. If NULL (default), the family stored within the fitted object is used. Specifying a different family can be useful for diagnostic comparisons. Available options match those in gkwreg: "gkw", "bkw", "kkw", "ekw", "mc", "kw", "beta".
- `nsim` — Integer. Number of simulations used to generate the envelope in the half-normal plot (which = 5). Defaults to 100. Must be positive.
- `level` — Numeric. The confidence level (between 0 and 1) for the simulated envelope in the half-normal plot (which = 5). Defaults to 0.90.

use_ggplot
: Logical. If TRUE, plots are generated using the ggplot2 package. If FALSE (default), base R graphics are used. Requires the ggplot2 package to be installed if set to TRUE.

arrange_plots
: Logical. Only relevant if use_ggplot = TRUE and multiple plots are requested (length(which) > 1). If TRUE, attempts to arrange the generated ggplot objects into a grid using either the gridExtra or ggpubr package (requires one of them to be installed). Defaults to FALSE.

sample_size
: Integer or NULL. If specified as an integer less than the total number of observations (x$nobs), a random sample of this size is used for calculating diagnostics and plotting. This can be useful for speeding up plots with very large datasets. Defaults to NULL (use all observations).

theme_fn
: A function. Only relevant if use_ggplot = TRUE. Specifies a ggplot2 theme function to apply to the plots (e.g., theme_bw, theme_classic). Defaults to ggplot2::theme_minimal.

save_diagnostics
: Logical. If TRUE, the function invisibly returns a list containing the calculated diagnostic measures (residuals, leverage, Cook's distance, etc.) instead of the model object. If FALSE (default), the function invisibly returns the original model object x.

Details

Diagnostic plots are essential for evaluating the assumptions and adequacy of fitted regression models. This function provides several standard plots adapted for gkwreg objects.

The choice of residual type (type) is important. For models with bounded responses like the GKw family, quantile residuals (type = "quantile") are generally preferred as they are constructed to be approximately normally distributed under a correctly specified model, making standard diagnostic tools like QQ-plots more directly interpretable.

The plots help to assess:

- Plot 1 (Residuals vs. Index): Potential patterns or autocorrelation over time/index.
- Plot 2 (Cook's Distance): Observations with disproportionately large influence on the estimated coefficients.
- Plot 3 (Leverage vs. Fitted): Observations with unusual predictor combinations (high leverage) that might influence the fit.
- Plot 4 (Residuals vs. Linear Predictor): Non-linearity in the predictor-response relationship or non-constant variance (heteroscedasticity).
- Plot 5 (Half-Normal Plot): Deviations from the assumed residual distribution (ideally normal for quantile residuals). Points outside the simulated envelope are potentially problematic.
- Plot 6 (Predicted vs. Observed): Overall goodness-of-fit and potential systematic over- or under-prediction.

The function relies on internal helper functions to calculate the necessary diagnostic quantities and generate the plots using either base R or ggplot2.

Value

Invisibly returns either the original fitted model object x (if save_diagnostics = FALSE) or a list containing the calculated diagnostic measures used for plotting (if save_diagnostics = TRUE). Primarily called for its side effect of generating plots.

Author(s)

Lopes, J. E.

See Also

gkwreg, residuals.gkwreg, summary.gkwreg, plot.lm, ggplot, grid.arrange, ggarrange

Examples

```
# Assume 'mydata' exists with response 'y' and predictors 'x1', 'x2'
# and that rgkw() is available and data is appropriate (0 < y < 1).
set.seed(456)
n <- 150
x1 <- runif(n, -1, 1)
x2 <- rnorm(n)
alpha <- exp(0.5 + 0.2 * x1)
beta <- exp(0.8 - 0.3 * x1 + 0.1 * x2)
gamma <- exp(0.6)
delta <- plogis(0.0 + 0.2 * x1)
lambda <- exp(-0.2 + 0.1 * x2)
# Use stats::rbeta as placeholder if rgkw is not available
y <- stats::rbeta(n, shape1 = gamma * alpha, shape2 = delta * beta) # Approximation
y <- pmax(pmin(y, 1 - 1e-7), 1e-7)
mydata <- data.frame(y = y, x1 = x1, x2 = x2)

# Fit a GKw model
model <- gkwreg(y ~ x1 | x1 + x2 | 1 | x1 | x2, data = mydata, family = "gkw")

# --- Generate default base R plots (prompts for each plot) ---
plot(model)

# --- Generate specific plots using base R ---
plot(model, which = c(1, 5), type = "quantile") # Residuals vs Index, Half-Normal

# --- Generate plots using ggplot2 (requires ggplot2 package) ---
# Ensure ggplot2 is installed: install.packages("ggplot2")
plot(model, which = c(4, 6), use_ggplot = TRUE) # Res vs Lin Pred, Pred vs Obs

# --- Generate all ggplot2 plots and arrange them (requires gridExtra or ggpubr) ---
# Ensure gridExtra is installed: install.packages("gridExtra")
# plot(model, use_ggplot = TRUE, arrange_plots = TRUE, ask = FALSE)

# --- Generate plots using Pearson residuals ---
plot(model, which = 4, type = "pearson") # Res vs Lin Pred using Pearson residuals

# --- Save diagnostic measures instead of plotting ---
```

```
diagnostics <- plot(model, save_diagnostics = TRUE)
head(diagnostics$residuals)
head(diagnostics$cooks_distance)
```

---

plotcompare *Compare Goodness-of-Fit Results Across Multiple Models*

Description

Creates a comparison of goodness-of-fit statistics and plots across multiple models from the GKw family of distributions.

Usage

```
plotcompare(gof_list, criteria = "all", plot = TRUE, plot_type = "all", ...)
```

Arguments

gof_list	A named list of gkwgof objects, where names are used as model identifiers.
criteria	Character vector specifying which criteria to compare. Available options are: "information" (for AIC, BIC, etc.), "distance" (for KS, CvM, AD, etc.), "prediction" (for MAE, RMSE, etc.), "probability" (for P-P, Q-Q correlations), or "all" for all criteria. Default is "all".
plot	Logical; if TRUE, creates comparison plots. Default is TRUE.
plot_type	Character string specifying the type of plot to create. Available options are: "radar" for a radar chart (requires the fmsb package), "bar" for bar charts, "table" for a formatted table, or "all" for all plot types. Default is "all".
...	Additional arguments to be passed to plotting functions.

Value

A list containing the comparison results and plots.

Examples

```
# Generate sample data
set.seed(123)
data <- rkw(n = 200, alpha = 2.5, beta = 1.8)

# Fit multiple models
fit_kw <- gkwfit(data, family = "kw")
fit_beta <- gkwfit(data, family = "beta")
fit_gkw <- gkwfit(data, family = "gkw")

# Calculate goodness-of-fit statistics for each model
gof_kw <- gkwgof(fit_kw, print_summary = FALSE)
```

```
gof_beta <- gkwgof(fit_beta, print_summary = FALSE)
gof_gkw <- gkwgof(fit_gkw, print_summary = FALSE)

# Compare the models
comparison <- plotcompare(
  list(KW = gof_kw, Beta = gof_beta, GKW = gof_gkw),
  plot_type = "all"
)
```

---

pmc *CDF of the McDonald (Mc)/Beta Power Distribution*

Description

Computes the cumulative distribution function (CDF), $F(q) = P(X \le q)$, 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 $P(X \le q)$, 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")
```

---

predict.gkwreg *Predictions from a Fitted Generalized Kumaraswamy Regression Model*

Description

Computes predictions and related quantities from a fitted Generalized Kumaraswamy (GKw) regression model object. This method can extract fitted values, predicted means, linear predictors, parameter values, variances, densities, probabilities, and quantiles based on the estimated model. Predictions can be made for new data or for the original data used to fit the model.

Usage

```
## S3 method for class 'gkwreg'
predict(
  object,
  newdata = NULL,
  type = "response",
  na.action = stats::na.pass,
  at = 0.5,
  elementwise = NULL,
  family = NULL,
  ...
)
```

Arguments

`object` An object of class "gkwreg", typically the result of a call to `gkwreg`.

newdata
: An optional data frame containing the variables needed for prediction. If omitted, predictions are made for the data used to fit the model.

type
: A character string specifying the type of prediction. Options are:

  "response" **or** "mean" Predicted mean response (default).

  "link" Linear predictors for each parameter before applying inverse link functions.

  "parameter" Parameter values on their original scale (after applying inverse link functions).

  "alpha"**,** "beta"**,** "gamma"**,** "delta"**,** "lambda" Values for a specific distribution parameter.

  "variance" Predicted variance of the response.

  "density" **or** "pdf" Density function values at points specified by at.

  "probability" **or** "cdf" Cumulative distribution function values at points specified by at.

  "quantile" Quantiles corresponding to probabilities specified by at.

na.action
: Function determining how to handle missing values in newdata. Default is stats::na.pass, which returns NA for cases with missing predictors.

at
: Numeric vector of values at which to evaluate densities, probabilities, or for which to compute quantiles, depending on type. Required for type = "density", type = "probability", or type = "quantile". Defaults to 0.5.

elementwise
: Logical. If TRUE and at has the same length as the number of observations, applies each value in at to the corresponding observation. If FALSE (default), applies all values in at to each observation, returning a matrix.

family
: Character string specifying the distribution family to use for calculations. If NULL (default), uses the family from the fitted model. Options match those in gkwreg: "gkw", "bkw", "kkw", "ekw", "mc", "kw", "beta".

...
: Additional arguments (currently not used).

Details

The predict.gkwreg function provides a flexible framework for obtaining predictions and inference from fitted Generalized Kumaraswamy regression models. It handles all subfamilies of GKw distributions and respects the parametrization and link functions specified in the original model.

Prediction Types: The function supports several types of predictions:

- **Response/Mean**: Computes the expected value of the response variable based on the model parameters. For most GKw family distributions, this requires numerical integration or special formulas.
- **Link**: Returns the linear predictors for each parameter without applying inverse link functions. These are the values $\eta_j = X\beta_j$ for each parameter j.
- **Parameter**: Computes the distribution parameter values on their original scale by applying the appropriate inverse link functions to the linear predictors. For example, if alpha uses a log link, then $\alpha = \exp(X\beta_\alpha)$.
- **Individual Parameters**: Extract specific parameter values (alpha, beta, gamma, delta, lambda) on their original scale.

- **Variance**: Estimates the variance of the response based on the model parameters. For some distributions, analytical formulas are used; for others, numerical approximations are employed.
- **Density/PDF**: Evaluates the probability density function at specified points given the model parameters.
- **Probability/CDF**: Computes the cumulative distribution function at specified points given the model parameters.
- **Quantile**: Calculates quantiles corresponding to specified probabilities given the model parameters.

Link Functions: The function respects the link functions specified in the original model for each parameter. The supported link functions are:

- `"log"`: $g(\mu) = \log(\mu)$, $g^{-1}(\eta) = \exp(\eta)$
- `"logit"`: $g(\mu) = \log(\mu/(1-\mu))$, $g^{-1}(\eta) = 1/(1+\exp(-\eta))$
- `"probit"`: $g(\mu) = \Phi^{-1}(\mu)$, $g^{-1}(\eta) = \Phi(\eta)$
- `"cauchy"`: $g(\mu) = \tan(\pi * (\mu - 0.5))$, $g^{-1}(\eta) = 0.5 + (1/\pi)\arctan(\eta)$
- `"cloglog"`: $g(\mu) = \log(-\log(1-\mu))$, $g^{-1}(\eta) = 1 - \exp(-\exp(\eta))$
- `"identity"`: $g(\mu) = \mu$, $g^{-1}(\eta) = \eta$
- `"sqrt"`: $g(\mu) = \sqrt{\mu}$, $g^{-1}(\eta) = \eta^2$
- `"inverse"`: $g(\mu) = 1/\mu$, $g^{-1}(\eta) = 1/\eta$
- `"inverse-square"`: $g(\mu) = 1/\sqrt{\mu}$, $g^{-1}(\eta) = 1/\eta^2$

Family-Specific Constraints: The function enforces appropriate constraints for each distribution family:

- `"gkw"`: All 5 parameters $(\alpha, \beta, \gamma, \delta, \lambda)$ are used.
- `"bkw"`: $\lambda = 1$ is fixed.
- `"kkw"`: $\gamma = 1$ is fixed.
- `"ekw"`: $\gamma = 1, \delta = 0$ are fixed.
- `"mc"`: $\alpha = 1, \beta = 1$ are fixed.
- `"kw"`: $\gamma = 1, \delta = 0, \lambda = 1$ are fixed.
- `"beta"`: $\alpha = 1, \beta = 1, \lambda = 1$ are fixed.

Parameter Bounds: All parameters are constrained to their valid ranges:

- $\alpha, \beta, \gamma, \lambda > 0$
- $0 < \delta < 1$

Using with New Data: When providing `newdata`, ensure it contains all variables used in the model's formula. The function extracts the terms for each parameter's model matrix and applies the appropriate link functions to calculate predictions. If any variables are missing, the function will attempt to substitute reasonable defaults or raise an error if critical variables are absent.

Using for Model Evaluation: The function is useful for model checking, generating predicted values for plotting, and evaluating the fit of different distribution families. By specifying the `family` parameter, you can compare predictions under different distributional assumptions.

Value

The return value depends on the `type` argument:

- For `type = "response"`, `type = "variance"`, or individual parameters (`type = "alpha"`, etc.): A numeric vector of length equal to the number of rows in `newdata` (or the original data).
- For `type = "link"` or `type = "parameter"`: A data frame with columns for each parameter and rows corresponding to observations.
- For `type = "density"`, `type = "probability"`, or `type = "quantile"`:
  - If `elementwise = TRUE`: A numeric vector of length equal to the number of rows in `newdata` (or the original data).
  - If `elementwise = FALSE`: A matrix where rows correspond to observations and columns correspond to the values in `at`.

Author(s)

Lopes, J. E. and contributors

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.

Ferrari, S. L. P., & Cribari-Neto, F. (2004). Beta regression for modelling rates and proportions. *Journal of Applied Statistics*, **31**(7), 799-815.

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

See Also

`gkwreg` for fitting Generalized Kumaraswamy regression models, `fitted.gkwreg` for extracting fitted values, `residuals.gkwreg` for calculating residuals, `summary.gkwreg` for model summaries, `coef.gkwreg` for extracting coefficients.

Examples

```
# Generate a sample dataset (n = 1000)
set.seed(123)
n <- 1000

# Create predictors
x1 <- runif(n, -2, 2)
x2 <- rnorm(n)
x3 <- factor(rbinom(n, 1, 0.4))

# Simulate Kumaraswamy distributed data
# True parameters with specific relationships to predictors
true_alpha <- exp(0.7 + 0.3 * x1)
```

```
true_beta <- exp(1.2 - 0.2 * x2 + 0.4 * (x3 == "1"))

# Generate random responses
y <- rkw(n, alpha = true_alpha, beta = true_beta)

# Ensure responses are strictly in (0, 1)
y <- pmax(pmin(y, 1 - 1e-7), 1e-7)

# Create data frame
df <- data.frame(y = y, x1 = x1, x2 = x2, x3 = x3)

# Split into training and test sets
set.seed(456)
train_idx <- sample(n, 800)
train_data <- df[train_idx, ]
test_data <- df[-train_idx, ]

# ====================================================================
# Example 1: Basic usage - Fit a Kumaraswamy model and make predictions
# ====================================================================

# Fit the model
kw_model <- gkwreg(y ~ x1 | x2 + x3, data = train_data, family = "kw")

# Predict mean response for test data
pred_mean <- predict(kw_model, newdata = test_data, type = "response")

# Calculate prediction error
mse <- mean((test_data$y - pred_mean)^2)
cat("Mean Squared Error:", mse, "\n")

# ====================================================================
# Example 2: Different prediction types
# ====================================================================

# Create a grid of values for visualization
x1_grid <- seq(-2, 2, length.out = 100)
grid_data <- data.frame(x1 = x1_grid, x2 = 0, x3 = 0)

# Predict different quantities
pred_mean <- predict(kw_model, newdata = grid_data, type = "response")
pred_var <- predict(kw_model, newdata = grid_data, type = "variance")
pred_params <- predict(kw_model, newdata = grid_data, type = "parameter")
pred_alpha <- predict(kw_model, newdata = grid_data, type = "alpha")
pred_beta <- predict(kw_model, newdata = grid_data, type = "beta")

# Plot predicted mean and parameters against x1
plot(x1_grid, pred_mean,
  type = "l", col = "blue",
  xlab = "x1", ylab = "Predicted Mean", main = "Mean Response vs x1"
)
plot(x1_grid, pred_var,
  type = "l", col = "red",
```

```
  xlab = "x1", ylab = "Predicted Variance", main = "Response Variance vs x1"
)
plot(x1_grid, pred_alpha,
  type = "l", col = "purple",
  xlab = "x1", ylab = "Alpha", main = "Alpha Parameter vs x1"
)
plot(x1_grid, pred_beta,
  type = "l", col = "green",
  xlab = "x1", ylab = "Beta", main = "Beta Parameter vs x1"
)

# ======================================================================
# Example 3: Computing densities, CDFs, and quantiles
# ======================================================================

# Select a single observation
obs_data <- test_data[1, ]

# Create a sequence of y values for plotting
y_seq <- seq(0.01, 0.99, length.out = 100)

# Compute density at each y value
dens_values <- predict(kw_model,
  newdata = obs_data,
  type = "density", at = y_seq, elementwise = FALSE
)

# Compute CDF at each y value
cdf_values <- predict(kw_model,
  newdata = obs_data,
  type = "probability", at = y_seq, elementwise = FALSE
)

# Compute quantiles for a sequence of probabilities
prob_seq <- seq(0.1, 0.9, by = 0.1)
quant_values <- predict(kw_model,
  newdata = obs_data,
  type = "quantile", at = prob_seq, elementwise = FALSE
)

# Plot density and CDF
plot(y_seq, dens_values,
  type = "l", col = "blue",
  xlab = "y", ylab = "Density", main = "Predicted PDF"
)
plot(y_seq, cdf_values,
  type = "l", col = "red",
  xlab = "y", ylab = "Cumulative Probability", main = "Predicted CDF"
)

# ======================================================================
# Example 4: Prediction under different distributional assumptions
# ======================================================================
```

```
# Fit models with different families
beta_model <- gkwreg(y ~ x1 | x2 + x3, data = train_data, family = "beta")
gkw_model <- gkwreg(y ~ x1 | x2 + x3 | 1 | 1 | x3, data = train_data, family = "gkw")

# Predict means using different families
pred_kw <- predict(kw_model, newdata = test_data, type = "response")
pred_beta <- predict(beta_model, newdata = test_data, type = "response")
pred_gkw <- predict(gkw_model, newdata = test_data, type = "response")

# Calculate MSE for each family
mse_kw <- mean((test_data$y - pred_kw)^2)
mse_beta <- mean((test_data$y - pred_beta)^2)
mse_gkw <- mean((test_data$y - pred_gkw)^2)

cat("MSE by family:\n")
cat("Kumaraswamy:", mse_kw, "\n")
cat("Beta:", mse_beta, "\n")
cat("GKw:", mse_gkw, "\n")

# Compare predictions from different families visually
plot(test_data$y, pred_kw,
  col = "blue", pch = 16,
  xlab = "Observed", ylab = "Predicted", main = "Predicted vs Observed"
)
points(test_data$y, pred_beta, col = "red", pch = 17)
points(test_data$y, pred_gkw, col = "green", pch = 18)
abline(0, 1, lty = 2)
legend("topleft",
  legend = c("Kumaraswamy", "Beta", "GKw"),
  col = c("blue", "red", "green"), pch = c(16, 17, 18)
)

# ============================================================================
# Example 5: Working with linear predictors and link functions
# ============================================================================

# Extract linear predictors and parameter values
lp <- predict(kw_model, newdata = test_data, type = "link")
params <- predict(kw_model, newdata = test_data, type = "parameter")

# Verify that inverse link transformation works correctly
# For Kumaraswamy model, alpha and beta use log links by default
alpha_from_lp <- exp(lp$alpha)
beta_from_lp <- exp(lp$beta)

# Compare with direct parameter predictions
cat("Manual inverse link vs direct parameter prediction:\n")
cat("Alpha difference:", max(abs(alpha_from_lp - params$alpha)), "\n")
cat("Beta difference:", max(abs(beta_from_lp - params$beta)), "\n")

# ============================================================================
# Example 6: Elementwise calculations
```

```
# ======================================================================

# Generate probabilities specific to each observation
probs <- runif(nrow(test_data), 0.1, 0.9)

# Calculate quantiles for each observation at its own probability level
quant_elementwise <- predict(kw_model,
  newdata = test_data,
  type = "quantile", at = probs, elementwise = TRUE
)

# Calculate probabilities at each observation's actual value
prob_at_y <- predict(kw_model,
  newdata = test_data,
  type = "probability", at = test_data$y, elementwise = TRUE
)

# Create Q-Q plot
plot(sort(prob_at_y), seq(0, 1, length.out = length(prob_at_y)),
  xlab = "Empirical Probability", ylab = "Theoretical Probability",
  main = "P-P Plot", type = "l"
)
abline(0, 1, lty = 2, col = "red")

# ======================================================================
# Example 7: Predicting for the original data
# ======================================================================

# Fit a model with original data
full_model <- gkwreg(y ~ x1 + x2 + x3 | x1 + x2 + x3, data = df, family = "kw")

# Get fitted values using predict and compare with model's fitted.values
fitted_from_predict <- predict(full_model, type = "response")
fitted_from_model <- full_model$fitted.values

# Compare results
cat(
  "Max difference between predict() and fitted.values:",
  max(abs(fitted_from_predict - fitted_from_model)), "\n"
)

# ======================================================================
# Example 8: Handling missing data
# ======================================================================

# Create test data with some missing values
test_missing <- test_data
test_missing$x1[1:5] <- NA
test_missing$x2[6:10] <- NA

# Predict with different na.action options
pred_na_pass <- tryCatch(
  predict(kw_model, newdata = test_missing, na.action = na.pass),
```

```
  error = function(e) rep(NA, nrow(test_missing))
)
pred_na_omit <- tryCatch(
  predict(kw_model, newdata = test_missing, na.action = na.omit),
  error = function(e) rep(NA, nrow(test_missing))
)

# Show which positions have NAs
cat("Rows with missing predictors:", which(is.na(pred_na_pass)), "\n")
cat("Length after na.omit:", length(pred_na_omit), "\n")



## Example 1: Simple Kumaraswamy regression model ----
set.seed(123)
n <- 1000
x1 <- runif(n, -2, 2)
x2 <- rnorm(n)

# True regression coefficients
alpha_coef <- c(0.8, 0.3, -0.2) # Intercept, x1, x2
beta_coef <- c(1.2, -0.4, 0.1) # Intercept, x1, x2

# Generate linear predictors and transform to parameters using inverse link (exp)
eta_alpha <- alpha_coef[1] + alpha_coef[2] * x1 + alpha_coef[3] * x2
eta_beta <- beta_coef[1] + beta_coef[2] * x1 + beta_coef[3] * x2
alpha_true <- exp(eta_alpha)
beta_true <- exp(eta_beta)

# Generate responses from Kumaraswamy distribution (assuming rkw is available)
y <- rkw(n, alpha = alpha_true, beta = beta_true)
# Create data frame
df1 <- data.frame(y = y, x1 = x1, x2 = x2)

# Fit Kumaraswamy regression model using extended formula syntax
# Model alpha ~ x1 + x2 and beta ~ x1 + x2
kw_reg <- gkwreg(y ~ x1 + x2 | x1 + x2, data = df1, family = "kw", silent = TRUE)

# Display summary
summary(kw_reg)

## Example 2: Generalized Kumaraswamy regression ----
set.seed(456)
x1 <- runif(n, -1, 1)
x2 <- rnorm(n)
x3 <- factor(rbinom(n, 1, 0.5), labels = c("A", "B")) # Factor variable

# True regression coefficients
alpha_coef <- c(0.5, 0.2) # Intercept, x1
beta_coef <- c(0.8, -0.3, 0.1) # Intercept, x1, x2
gamma_coef <- c(0.6, 0.4) # Intercept, x3B
delta_coef <- c(0.0, 0.2) # Intercept, x3B (logit scale)
lambda_coef <- c(-0.2, 0.1) # Intercept, x2
```

```
# Design matrices
X_alpha <- model.matrix(~x1, data = data.frame(x1 = x1))
X_beta <- model.matrix(~ x1 + x2, data = data.frame(x1 = x1, x2 = x2))
X_gamma <- model.matrix(~x3, data = data.frame(x3 = x3))
X_delta <- model.matrix(~x3, data = data.frame(x3 = x3))
X_lambda <- model.matrix(~x2, data = data.frame(x2 = x2))

# Generate linear predictors and transform to parameters
alpha <- exp(X_alpha %*% alpha_coef)
beta <- exp(X_beta %*% beta_coef)
gamma <- exp(X_gamma %*% gamma_coef)
delta <- plogis(X_delta %*% delta_coef) # logit link for delta
lambda <- exp(X_lambda %*% lambda_coef)

# Generate response from GKw distribution (assuming rgkw is available)
y <- rgkw(n, alpha = alpha, beta = beta, gamma = gamma, delta = delta, lambda = lambda)

# Create data frame
df2 <- data.frame(y = y, x1 = x1, x2 = x2, x3 = x3)

# Fit GKw regression with parameter-specific formulas
# alpha ~ x1, beta ~ x1 + x2, gamma ~ x3, delta ~ x3, lambda ~ x2
gkw_reg <- gkwreg(y ~ x1 | x1 + x2 | x3 | x3 | x2, data = df2, family = "gkw")

# Compare true vs. estimated coefficients
print("Estimated Coefficients (GKw):")
print(coef(gkw_reg))
print("True Coefficients (approx):")
print(list(
  alpha = alpha_coef, beta = beta_coef, gamma = gamma_coef,
  delta = delta_coef, lambda = lambda_coef
))

## Example 3: Beta regression for comparison ----
set.seed(789)
x1 <- runif(n, -1, 1)

# True coefficients for Beta parameters (gamma = shape1, delta = shape2)
gamma_coef <- c(1.0, 0.5) # Intercept, x1 (log scale for shape1)
delta_coef <- c(1.5, -0.7) # Intercept, x1 (log scale for shape2)

# Generate linear predictors and transform (default link is log for Beta params here)
X_beta_eg <- model.matrix(~x1, data.frame(x1 = x1))
gamma_true <- exp(X_beta_eg %*% gamma_coef)
delta_true <- exp(X_beta_eg %*% delta_coef)

# Generate response from Beta distribution
y <- rbeta_(n, gamma_true, delta_true)

# Create data frame
df_beta <- data.frame(y = y, x1 = x1)
```

```
# Fit Beta regression model using gkwreg
# Formula maps to gamma and delta: y ~  x1 | x1
beta_reg <- gkwreg(y ~ x1 | x1,
  data = df_beta, family = "beta",
  link = list(gamma = "log", delta = "log")
) # Specify links if non-default

## Example 4: Model comparison using AIC/BIC ----
# Fit an alternative model, e.g., Kumaraswamy, to the same beta-generated data
kw_reg2 <- try(gkwreg(y ~ x1 | x1, data = df_beta, family = "kw"))

print("AIC Comparison (Beta vs Kw):")
c(AIC(beta_reg), AIC(kw_reg2))
print("BIC Comparison (Beta vs Kw):")
c(BIC(beta_reg), BIC(kw_reg2))

## Example 5: Predicting with a fitted model

# Use the Beta regression model from Example 3
newdata <- data.frame(x1 = seq(-1, 1, length.out = 20))

# Predict expected response (mean of the Beta distribution)
pred_response <- predict(beta_reg, newdata = newdata, type = "response")

# Predict parameters (gamma and delta) on the scale of the link function
pred_link <- predict(beta_reg, newdata = newdata, type = "link")

# Predict parameters on the original scale (shape1, shape2)
pred_params <- predict(beta_reg, newdata = newdata, type = "parameter")

# Plot original data and predicted mean response curve
plot(df_beta$x1, df_beta$y,
  pch = 20, col = "grey", xlab = "x1", ylab = "y",
  main = "Beta Regression Fit (using gkwreg)"
)
lines(newdata$x1, pred_response, col = "red", lwd = 2)
legend("topright", legend = "Predicted Mean", col = "red", lty = 1, lwd = 2)
```

`print.anova.gkwfit` *S3 method for class 'anova.gkwfit'*

Description

S3 method for class 'anova.gkwfit'

Usage

```
## S3 method for class 'anova.gkwfit'
```

```
print(
  x,
  digits = max(getOption("digits") - 2L, 3L),
  signif.stars = getOption("show.signif.stars", TRUE),
  ...
)
```

Arguments

`x`	An object of class `"anova.gkwfit"`.
`digits`	Minimum number of significant digits to print.
`signif.stars`	Logical; if TRUE, add significance stars.
`...`	Other args passed to

Value

An object of class `"anova.gkwfit"` and prints a summary.

`print.gkwfitall` *Print method for gkwfitall objects*

Description

Print method for gkwfitall objects

Usage

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

Arguments

`x`	An object of class `"gkwfitall"`
`...`	Additional arguments (currently ignored)

Value

Invisibly returns the input object

Author(s)

Lopes, J. E.

`print.gkwgof` *Print Method for gkwgof Objects*

Description

Prints a summary of the goodness-of-fit analysis for GKw family distributions.

Usage

```
## S3 method for class 'gkwgof'
print(x, verbose = FALSE, ...)
```

Arguments

`x`	An object of class "gkwgof", typically the result of a call to `gkwgof`.
`verbose`	Logical; if TRUE, provides additional details and explanations. Default is FALSE.
`...`	Additional arguments (currently unused).

Value

The input object `x` is returned invisibly.

`print.summary.gkwfitall` *Print method for summary.gkwfitall objects*

Description

Print method for summary.gkwfitall objects

Usage

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

Arguments

`x`	An object of class "`summary.gkwfitall`"
`...`	Additional arguments (currently ignored)

Value

Invisibly returns the input object

Author(s)

Lopes, J. E.

print.summary.gkwgof   *Print Method for summary.gkwgof Objects*

Description

Prints a formatted summary of goodness-of-fit results.

Usage

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

Arguments

x	An object of class "summary.gkwgof".
...	Additional arguments (not used).

Value

The input object invisibly.

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 $p = P(X \le q)$, 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 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 $v \le 1$.
4. Calculate $q = (1 - w)^{1/\alpha}$. Note: Requires $w \le 1$.

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 $p = P(X \le q)$, 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 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 $p = P(X \leq q)$, 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 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)
# Check: qkw(p, ..., lt=F) == qkw(1-p, ..., lt=T)
print(qkw(1 - p_vals, alpha_par, beta_par))

# 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
```

`residuals.gkwreg` *Extract Residuals from a Generalized Kumaraswamy Regression Model*

Description

Extracts or calculates various types of residuals from a fitted Generalized Kumaraswamy (GKw) regression model object of class "gkwreg", useful for model diagnostics.

Usage

```
## S3 method for class 'gkwreg'
residuals(
  object,
  type = c("response", "pearson", "deviance", "quantile", "modified.deviance",
    "cox-snell", "score", "partial"),
  covariate_idx = 1,
```

```
  parameter = "alpha",
  family = NULL,
  ...
)
```

Arguments

object
: An object of class "gkwreg", typically the result of a call to gkwreg.

type
: Character string specifying the type of residuals to compute. Available options are:

- "response": (Default) Raw response residuals: $y - \mu$, where μ is the fitted mean.
- "pearson": Pearson residuals: $(y - \mu)/\sqrt{V(\mu)}$, where $V(\mu)$ is the variance function of the specified family.
- "deviance": Deviance residuals: Signed square root of the unit deviances. Sum of squares equals the total deviance.
- "quantile": Randomized quantile residuals (Dunn & Smyth, 1996). Transformed via the model's CDF and the standard normal quantile function. Should approximate a standard normal distribution if the model is correct.
- "modified.deviance": (Not typically implemented, placeholder) Standardized deviance residuals, potentially adjusted for leverage.
- "cox-snell": Cox-Snell residuals: $-\log(1 - F(y))$, where $F(y)$ is the model's CDF. Should approximate a standard exponential distribution if the model is correct.
- "score": (Not typically implemented, placeholder) Score residuals, related to the derivative of the log-likelihood.
- "partial": Partial residuals for a specific predictor in one parameter's linear model: $eta_p + \beta_{pk} x_{ik}$, where eta_p is the partial linear predictor and $\beta_{pk} x_{ik}$ is the component associated with the k-th covariate for the i-th observation. Requires parameter and covariate_idx.

covariate_idx
: Integer. Only used if type = "partial". Specifies the index (column number in the corresponding model matrix) of the covariate for which to compute partial residuals.

parameter
: Character string. Only used if type = "partial". Specifies the distribution parameter ("alpha", "beta", "gamma", "delta", or "lambda") whose linear predictor contains the covariate of interest.

family
: Character string specifying the distribution family assumptions to use when calculating residuals (especially for types involving variance, deviance, CDF, etc.). If NULL (default), the family stored within the fitted object is used. Specifying a different family may be useful for diagnostic comparisons. Available options match those in gkwreg: "gkw", "bkw", "kkw", "ekw", "mc", "kw", "beta".

...
: Additional arguments, currently ignored by this method.

Details

This function calculates various types of residuals useful for diagnosing the adequacy of a fitted GKw regression model.

- **Response residuals** (`type="response"`) are the simplest, showing raw differences between observed and fitted mean values.
- **Pearson residuals** (`type="pearson"`) account for the mean-variance relationship specified by the model family. Constant variance when plotted against fitted values suggests the variance function is appropriate.
- **Deviance residuals** (`type="deviance"`) are related to the log-likelihood contribution of each observation. Their sum of squares equals the total model deviance. They often have more symmetric distributions than Pearson residuals.
- **Quantile residuals** (`type="quantile"`) are particularly useful for non-standard distributions as they should always be approximately standard normal if the assumed distribution and model structure are correct. Deviations from normality in a QQ-plot indicate model misspecification.
- **Cox-Snell residuals** (`type="cox-snell"`) provide another check of the overall distributional fit. A plot of the sorted residuals against theoretical exponential quantiles should approximate a straight line through the origin with slope 1.
- **Partial residuals** (`type="partial"`) help visualize the marginal relationship between a specific predictor and the response on the scale of the linear predictor for a chosen parameter, adjusted for other predictors.

Calculations involving the distribution's properties (variance, CDF, PDF) depend heavily on the specified `family`. The function relies on internal helper functions (potentially implemented in C++ for efficiency) to compute these based on the fitted parameters for each observation.

Value

A numeric vector containing the requested type of residuals. The length corresponds to the number of observations used in the model fit.

Author(s)

Lopes, J. E.

References

Dunn, P. K., & Smyth, G. K. (1996). Randomized Quantile Residuals. *Journal of Computational and Graphical Statistics*, **5**(3), 236-244.

Cox, D. R., & Snell, E. J. (1968). A General Definition of Residuals. *Journal of the Royal Statistical Society, Series B (Methodological)*, **30**(2), 248-275.

McCullagh, P., & Nelder, J. A. (1989). *Generalized Linear Models* (2nd ed.). Chapman and Hall/CRC.

See Also

`gkwreg`, `fitted.gkwreg`, `predict.gkwreg`, `residuals`

Examples

```
# Assume 'mydata' exists with response 'y' and predictors 'x1', 'x2'
# and that rgkw() is available and data is appropriate (0 < y < 1).
set.seed(456)
n <- 150
x1 <- runif(n, -1, 1)
x2 <- rnorm(n)
alpha <- exp(0.5 + 0.2 * x1)
beta <- exp(0.8 - 0.3 * x1 + 0.1 * x2)
gamma <- exp(0.6)
delta <- plogis(0.0 + 0.2 * x1)
lambda <- exp(-0.2 + 0.1 * x2)
# Use stats::rbeta as placeholder if rgkw is not available
y <- stats::rbeta(n, shape1 = gamma * alpha, shape2 = delta * beta) # Approximation
y <- pmax(pmin(y, 1 - 1e-7), 1e-7)
mydata <- data.frame(y = y, x1 = x1, x2 = x2)

# Fit a GKw model
model <- gkwreg(y ~ x1 | x1 + x2 | 1 | x1 | x2, data = mydata, family = "gkw")

# --- Extract different types of residuals ---
resp_res <- residuals(model, type = "response")
pearson_res <- residuals(model, type = "pearson")
quant_res <- residuals(model, type = "quantile")
cs_res <- residuals(model, type = "cox-snell")

# --- Diagnostic Plots ---
# QQ-plot for quantile residuals (should be approx. normal)
stats::qqnorm(quant_res, main = "QQ Plot: GKw Quantile Residuals")
stats::qqline(quant_res, col = "blue")

# Cox-Snell residuals plot (should be approx. exponential -> linear on exp-QQ)
plot(stats::qexp(stats::ppoints(length(cs_res))), sort(cs_res),
  xlab = "Theoretical Exponential Quantiles", ylab = "Sorted Cox-Snell Residuals",
  main = "Cox-Snell Residual Plot", pch = 1
)
abline(0, 1, col = "red")

# --- Compare residuals using a different family assumption ---
# Calculate quantile residuals assuming underlying Beta dist
quant_res_beta <- residuals(model, type = "quantile", family = "beta")

# Compare QQ-plots
stats::qqnorm(quant_res, main = "GKw Quantile Residuals")
stats::qqline(quant_res, col = "blue")
stats::qqnorm(quant_res_beta, main = "Beta Quantile Residuals (from GKw Fit)")
stats::qqline(quant_res_beta, col = "darkgreen")

# --- Partial Residuals ---
# Examine effect of x1 on the alpha parameter's linear predictor
if ("x1" %in% colnames(model$x$alpha)) { # Check if x1 is in alpha model matrix
  # Find index for 'x1' (could be 2 if intercept is first)
```

```
    x1_idx_alpha <- which(colnames(model$x$alpha) == "x1")
    if (length(x1_idx_alpha) == 1) {
      part_res_alpha_x1 <- residuals(model,
        type = "partial",
        parameter = "alpha", covariate_idx = x1_idx_alpha
      )
      # Plot partial residuals against the predictor
      plot(mydata$x1, part_res_alpha_x1,
        xlab = "x1", ylab = "Partial Residual (alpha predictor)",
        main = "Partial Residual Plot for alpha ~ x1"
      )
      # Add a smoother to see the trend
      lines(lowess(mydata$x1, part_res_alpha_x1), col = "red")
    }
  }

  # Examine effect of x2 on the beta parameter's linear predictor
  if ("x2" %in% colnames(model$x$beta)) {
    x2_idx_beta <- which(colnames(model$x$beta) == "x2")
    if (length(x2_idx_beta) == 1) {
      part_res_beta_x2 <- residuals(model,
        type = "partial",
        parameter = "beta", covariate_idx = x2_idx_beta
      )
      plot(mydata$x2, part_res_beta_x2,
        xlab = "x2", ylab = "Partial Residual (beta predictor)",
        main = "Partial Residual Plot for beta ~ x2"
      )
      lines(lowess(mydata$x2, part_res_beta_x2), col = "red")
    }
  }
```

`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
```

`summary.gkwfit` *Summary Method for gkwfit Objects*

Description

Calculates and prepares a detailed summary of a model fitted by gkwfit. This includes coefficients, standard errors, test statistics (z-values), p-values, log-likelihood, information criteria (AIC, BIC, AICc), number of estimated parameters, convergence status, and optimizer details.

Usage

```
## S3 method for class 'gkwfit'
summary(object, correlation = FALSE, ...)
```

Arguments

object	An object of class "gkwfit", typically the result of a call to gkwfit.
correlation	Logical; if TRUE, the correlation matrix of the estimated parameters is computed from the vcov component and included in the summary. Defaults to FALSE.
...	Additional arguments (currently unused).

Details

This function computes standard errors, z-values ($Estimate/SE$), and p-values (two-tailed test based on the standard normal distribution) for the estimated model parameters by utilizing the coefficient estimates (coef) and their variance-covariance matrix (vcov). This requires that the variance-covariance matrix was successfully computed and is available in the object (typically requires hessian = TRUE in the original gkwfit call and successful Hessian inversion).

If standard errors cannot be reliably calculated (e.g., vcov is missing, invalid, or indicates non-positive variance), the corresponding columns in the coefficient table will contain NA values, and the se_available flag will be set to FALSE.

The returned object is of class "summary.gkwfit", and its printing is handled by print.summary.gkwfit.

Value

An object of class "summary.gkwfit", which is a list containing:

call	The original function call.
family	The specified distribution family.
coefficients	A matrix of estimates, standard errors, z-values, and p-values. Contains NAs if SEs could not be computed.
loglik	The maximized log-likelihood value (numeric).
df	The number of estimated parameters.
aic	Akaike Information Criterion.
bic	Bayesian Information Criterion.
aicc	Corrected Akaike Information Criterion.
nobs	Number of observations used in fitting (integer).
convergence	The convergence code returned by the optimizer.
message	The message returned by the optimizer.

- `se_available` — Logical indicating if standard errors could be computed.
- `correlation` — The correlation matrix of coefficients (if `correlation = TRUE` and calculable), otherwise `NULL`.
- `fixed` — A named list of parameters that were held fixed during estimation, or `NULL`.
- `fit_method` — The primary fitting method specified ('tmb' or 'nr').
- `optimizer_method` — The specific optimizer used (e.g., 'nlminb', 'optim', 'Newton-Raphson').

Author(s)

Lopes, J. E. (with refinements)

See Also

`gkwfit`, `print.summary.gkwfit`, `coef.gkwfit`, `vcov.gkwfit`, `logLik.gkwfit`, `AIC.gkwfit`

Examples

```
# Generate data and fit model
set.seed(2203)
y <- rkw(50, alpha = 2, beta = 3)
fit <- gkwfit(data = y, family = "kw", plot = FALSE)

# Display detailed summary with parameter estimates and standard errors
summary(fit)

# Control digits in output
summary(fit, digits = 4)
```

---

summary.gkwfitall — *Summary method for gkwfitall objects*

Description

Summary method for gkwfitall objects

Usage

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

Arguments

- `object` — An object of class `"gkwfitall"`
- `...` — Additional arguments (currently ignored)

Value

A summarized version of the gkwfitall object

Author(s)

Lopes, J. E.

`summary.gkwgof` *Summary Method for gkwgof Objects*

Description

Provides a concise summary of the goodness-of-fit analysis results.

Usage

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

Arguments

`object`	An object of class "gkwgof".
`...`	Additional arguments (not used).

Value

A list containing key summary statistics.

`summary.gkwreg` *Summary Method for Generalized Kumaraswamy Regression Models*

Description

Computes and returns a detailed statistical summary for a fitted Generalized Kumaraswamy (GKw) regression model object of class `"gkwreg"`.

Usage

```
## S3 method for class 'gkwreg'
summary(object, conf.level = 0.95, ...)
```

Arguments

`object`	An object of class `"gkwreg"`, typically the result of a call to `gkwreg`.
`conf.level`	Numeric. The desired confidence level for constructing confidence intervals for the regression coefficients. Default is 0.95.
`...`	Additional arguments, currently ignored by this method.

Details

This method provides a comprehensive summary of the fitted gkwreg model. It calculates z-values and p-values for the regression coefficients based on the estimated standard errors (if available) and computes confidence intervals at the specified `conf.level`. The summary includes:

- The model call.
- The distribution family used.
- A table of coefficients including estimates, standard errors, z-values, and p-values. Note: Significance stars are typically added by the corresponding `print.summary.gkwreg` method.
- Confidence intervals for the coefficients.
- Link functions used for each parameter.
- Mean values of the fitted distribution parameters ($\alpha, \beta, \gamma, \delta, \lambda$).
- A five-number summary (Min, Q1, Median, Q3, Max) plus the mean of the response residuals.
- Key model fit statistics (Log-likelihood, AIC, BIC, RMSE, Efron's R^2).
- Information about model convergence and optimizer iterations.

If standard errors were not computed (e.g., `hessian = FALSE` in the original `gkwreg` call), the coefficient table will only contain estimates, and confidence intervals will not be available.

Value

An object of class `"summary.gkwreg"`, which is a list containing the following components:

`call`	The original function call that created the `object`.
`family`	Character string specifying the distribution family.
`coefficients`	A data frame (matrix) containing the coefficient estimates, standard errors, z-values, and p-values.
`conf.int`	A matrix containing the lower and upper bounds of the confidence intervals for the coefficients (if standard errors are available).
`link`	A list of character strings specifying the link functions used.
`fitted_parameters`	A list containing the mean values of the estimated distribution parameters.
`residuals`	A named numeric vector containing summary statistics for the response residuals.
`nobs`	Number of observations used in the fit.
`npar`	Total number of estimated regression coefficients.
`df.residual`	Residual degrees of freedom.
`loglik`	The maximized log-likelihood value.
`aic`	Akaike Information Criterion.
`bic`	Bayesian Information Criterion.
`rmse`	Root Mean Squared Error of the residuals.
`efron_r2`	Efron's pseudo-R-squared value.

mean_absolute_error
: Mean Absolute Error of the residuals.

convergence
: Convergence code from the optimizer.

iterations
: Number of iterations reported by the optimizer.

conf.level
: The confidence level used for calculating intervals.

Author(s)

Lopes, J. E.

See Also

gkwreg, print.summary.gkwreg, coef, confint

Examples

```
set.seed(123)
n <- 100
x1 <- runif(n, -2, 2)
x2 <- rnorm(n)
alpha_coef <- c(0.8, 0.3, -0.2)
beta_coef <- c(1.2, -0.4, 0.1)
eta_alpha <- alpha_coef[1] + alpha_coef[2] * x1 + alpha_coef[3] * x2
eta_beta <- beta_coef[1] + beta_coef[2] * x1 + beta_coef[3] * x2
alpha_true <- exp(eta_alpha)
beta_true <- exp(eta_beta)
# Use stats::rbeta as a placeholder if rkw is unavailable
y <- stats::rbeta(n, shape1 = alpha_true, shape2 = beta_true)
y <- pmax(pmin(y, 1 - 1e-7), 1e-7)
df <- data.frame(y = y, x1 = x1, x2 = x2)

# Fit a Kumaraswamy regression model
kw_reg <- gkwreg(y ~ x1 + x2 | x1 + x2, data = df, family = "kw")

# Generate detailed summary using the summary method
summary_kw <- summary(kw_reg)

# Print the summary object (uses print.summary.gkwreg)
print(summary_kw)

# Extract coefficient table directly from the summary object
coef_table <- coef(summary_kw) # Equivalent to summary_kw$coefficients
print(coef_table)
```

vcov.gkwfit *Extract Variance-Covariance Matrix from a gkwfit Object*

Description

Extracts the variance-covariance matrix of the estimated parameters from a model fitted by gkwfit. This matrix is typically derived from the inverse of the observed Hessian matrix calculated during fitting (requires hessian = TRUE in the gkwfit call). This is an S3 method for the generic vcov function.

Usage

```
## S3 method for class 'gkwfit'
vcov(object, ...)
```

Arguments

object	An object of class "gkwfit", typically the result of a call to gkwfit.
...	Additional arguments (currently ignored).

Value

A numeric matrix representing the variance-covariance matrix of the estimated model parameters. Row and column names correspond to the parameter names (e.g., "alpha", "beta", etc.). Returns NULL or raises a warning/error if the matrix is not available (e.g., if hessian=FALSE was used or if the Hessian computation failed).

Author(s)

Lopes, J. E.

See Also

gkwfit, vcov, coef.gkwfit, logLik.gkwfit

Examples

```
# Generate data and fit model (ensure hessian = TRUE for vcov)
set.seed(2203)
y <- rbkw(50, alpha = 2, beta = 3, gamma = 1.5, delta = 0.5)
fit <- gkwfit(data = y, family = "bkw", plot = FALSE, hessian = TRUE)

# Extract variance-covariance matrix
vcov_matrix <- vcov(fit)
print(vcov_matrix)

# Extract standard errors from the diagonal
std_errors <- sqrt(diag(vcov_matrix))
```

```
print(std_errors)

# Compare with standard errors from summary
summary_se <- summary(fit)$coefficients[, "Std. Error"]
all.equal(std_errors, summary_se)
```

---

vcov.gkwreg *Extract Variance-Covariance Matrix from a Generalized Kumaraswamy Regression Model*

Description

This function extracts the variance-covariance matrix of the estimated parameters from a fitted Generalized Kumaraswamy regression model. The variance-covariance matrix is essential for statistical inference, including hypothesis testing and confidence interval calculation.

Usage

```
## S3 method for class 'gkwreg'
vcov(object, complete = TRUE, ...)
```

Arguments

object	An object of class "gkwreg", typically the result of a call to gkwreg.
complete	Logical indicating whether the complete variance-covariance matrix should be returned in case some coefficients were omitted from the original fit. Currently ignored for gkwreg objects.
...	Additional arguments (currently not used).

Details

The variance-covariance matrix is estimated based on the observed information matrix, which is derived from the second derivatives of the log-likelihood function with respect to the model parameters. For gkwreg objects, this matrix is typically computed using the TMB (Template Model Builder) automatic differentiation framework during model fitting.

The diagonal elements of the variance-covariance matrix correspond to the squared standard errors of the parameter estimates, while the off-diagonal elements represent the covariances between pairs of parameters.

Value

A square matrix with row and column names corresponding to the coefficients in the model. If the variance-covariance matrix is not available (for example, if the model was fitted with `hessian = FALSE`), the function returns `NULL` with a warning.

See Also

`gkwreg`, `confint`, `summary.gkwreg`

Index