Package 'gkwreg'

May 8, 2026

Title Generalized Kumaraswamy Regression Models for Bounded Data

Version 2.1.14

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

URL https://github.com/evandeilton/gkwreg, https://evandeilton.github.io/gkwreg/

BugReports https://github.com/evandeilton/gkwreg/issues

Encoding UTF-8

RoxygenNote 7.3.3

Imports Formula, stats, graphics, TMB, Rcpp, magrittr, ggplot2, ggpubr, gridExtra, numDeriv, gkwdist

LinkingTo TMB, Rcpp, RcppArmadillo, RcppEigen

Suggests betareg, knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

NeedsCompilation yes

Depends R (>= 3.5)

LazyData true

VignetteBuilder knitr

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

Maintainer José Evandeilton Lopes <evandeilton@gmail.com>

Repository CRAN

Date/Publication 2026-01-09 16:20:17 UTC

Contents

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

Description

Calculates the Akaike Information Criterion (AIC) for fitted Generalized Kumaraswamy regression models.

Usage

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

Arguments

`object`	An object of class `"gkwreg"`, typically obtained from `gkwreg`.
`...`	Optionally more fitted model objects.
`k`	Numeric, the penalty per parameter. Default is `k = 2` for classical AIC. Setting `k = log(n)` gives BIC-equivalent penalty.

Details

The AIC is computed as:

$$AIC = -2\ell(\hat{\theta}) + k \cdot p$$

where $\ell(\hat{\theta})$ is the maximized log-likelihood and p is the number of estimated parameters.

When multiple objects are provided, a data frame comparing all models is returned. Lower AIC values indicate better models, balancing goodness-of-fit against model complexity.

For small sample sizes, consider the corrected AIC (AICc):

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

where n is the sample size. This correction is not automatically applied but can be calculated manually.

Value

If only one object is provided, returns a numeric value with the AIC. If multiple objects are provided, returns a data frame with columns `df` and `AIC`, with rows named according to the object names in the call.

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. doi:10.1109/TAC.1974.1100705

Burnham, K. P., & Anderson, D. R. (2004). Multimodel inference: Understanding AIC and BIC in model selection. *Sociological Methods & Research*, **33**(2), 261–304. doi:10.1177/0049124104268644

See Also

gkwreg, logLik.gkwreg, BIC.gkwreg

Examples

```
# Load example data
data(GasolineYield)

# Fit competing models
fit1 <- gkwreg(yield ~ batch, data = GasolineYield, family = "kw")
fit2 <- gkwreg(yield ~ batch + temp, data = GasolineYield, family = "kw")
fit3 <- gkwreg(yield ~ temp, data = GasolineYield, family = "kw")

# Calculate AIC for single model
AIC(fit1)

# Compare multiple models (with proper names)
AIC(fit1, fit2, fit3)

# Use different penalty
AIC(fit1, k = 4)
```

---

anova.gkwreg *Analysis of Deviance for GKw Regression Models*

Description

Computes an analysis of deviance table for one or more fitted Generalized Kumaraswamy (GKw) regression model objects. When multiple models are provided, likelihood ratio tests are performed to compare nested models.

Usage

```
## S3 method for class 'gkwreg'
anova(object, ..., test = c("Chisq", "none"))
```

Arguments

`object`	An object of class `"gkwreg"`, typically obtained from `gkwreg`.
`...`	Additional objects of class `"gkwreg"` for model comparison. Models must be nested and fitted to the same dataset.
`test`	A character string specifying the test statistic to use. Currently only `"Chisq"` (default) is supported, which performs likelihood ratio tests. Can also be `"none"` for no tests.

Details

When a single model is provided, the function returns a table showing the residual degrees of freedom and deviance.

When multiple models are provided, the function compares them using likelihood ratio tests (LRT). Models are automatically ordered by their complexity (degrees of freedom). The LRT statistic is computed as:

$$LRT = 2(\ell_1 - \ell_0)$$

where ℓ_1 is the log-likelihood of the more complex model and ℓ_0 is the log-likelihood of the simpler (nested) model. Under the null hypothesis that the simpler model is adequate, the LRT statistic follows a chi-squared distribution with degrees of freedom equal to the difference in the number of parameters between the models.

Important: This method assumes that the models being compared are nested (i.e., one model is a special case of the other) and fitted to the same data. Comparing non-nested models or models fitted to different datasets will produce unreliable results. Use `AIC` or `BIC` for comparing non-nested models.

The deviance is defined as $-2 \times$ log-likelihood. For models fitted by maximum likelihood, smaller (more negative) deviances indicate better fit. Note that deviance can be negative when the log-likelihood is positive, which occurs when density values exceed 1 (common in continuous distributions on bounded intervals). What matters for inference is the *change* in deviance between models, which should be positive when the more complex model fits better.

Value

An object of class `c("anova.gkwreg", "anova", "data.frame")`, with the following columns:

`Resid. Df` Residual degrees of freedom

`Resid. Dev` Residual deviance (-2 × log-likelihood)

`Df` Change in degrees of freedom (for model comparisons)

`Deviance` Change in deviance (for model comparisons)

`Pr(>Chi)` P-value from the chi-squared test (if `test = "Chisq"`)

Author(s)

Lopes, J. E.

References

Wilks, S. S. (1938). The large-sample distribution of the likelihood ratio for testing composite hypotheses. *The Annals of Mathematical Statistics*, **9**(1), 60–62. doi:10.1214/aoms/1177732360

Pawitan, Y. (2001). *In All Likelihood: Statistical Modelling and Inference Using Likelihood*. Oxford University Press.

See Also

gkwreg, logLik.gkwreg, AIC.gkwreg, BIC.gkwreg, lrtest

Examples

```
# Load example data
data(GasolineYield)

# Fit a series of nested models
fit1 <- gkwreg(yield ~ 1, data = GasolineYield, family = "kw")
fit2 <- gkwreg(yield ~ temp, data = GasolineYield, family = "kw")
fit3 <- gkwreg(yield ~ batch + temp, data = GasolineYield, family = "kw")

# ANOVA table for single model
anova(fit3)

# Compare nested models using likelihood ratio tests
anova(fit1, fit2, fit3)
#> Model 1 vs 2: Adding temperature is highly significant (p < 0.001)
#> Model 2 vs 3: Adding batch is highly significant (p < 0.001)

# Compare two models
anova(fit2, fit3, test = "Chisq")

# Suppress test statistics
anova(fit1, fit2, fit3, test = "none")
```

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

Description

Calculates the Bayesian Information Criterion (BIC), also known as the Schwarz Information Criterion (SIC), for fitted Generalized Kumaraswamy regression models.

Usage

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

Arguments

`object`	An object of class `"gkwreg"`, typically obtained from `gkwreg`.
`...`	Optionally more fitted model objects.

Details

The BIC is computed as:

$$BIC = -2\ell(\hat{\theta}) + p \cdot \log(n)$$

where $\ell(\hat{\theta})$ is the maximized log-likelihood, p is the number of estimated parameters, and n is the sample size.

When multiple objects are provided, a data frame comparing all models is returned. Lower BIC values indicate better models. BIC penalizes model complexity more heavily than AIC, particularly for large samples, and tends to favor more parsimonious models.

The BIC can be derived from a Bayesian perspective as an approximation to the logarithm of the Bayes factor, under certain regularity conditions and assuming uniform priors.

Value

If only one object is provided, returns a numeric value with the BIC. If multiple objects are provided, returns a data frame with columns `df` and `BIC`, with rows named according to the object names in the call.

Author(s)

Lopes, J. E.

References

Schwarz, G. (1978). Estimating the dimension of a model. *The Annals of Statistics*, **6**(2), 461–464. doi:10.1214/aos/1176344136

See Also

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

Examples

```
# Load example data
data(GasolineYield)

# Fit competing models
fit1 <- gkwreg(yield ~ batch, data = GasolineYield, family = "kw")
fit2 <- gkwreg(yield ~ batch + temp, data = GasolineYield, family = "kw")
fit3 <- gkwreg(yield ~ temp, data = GasolineYield, family = "kw")

# Calculate BIC for single model
BIC(fit1)

# Compare multiple models (with proper names)
```

```
BIC(fit1, fit2, fit3)
```

`CarTask` *Partition-Primed Probability Judgement Task for Car Dealership*

Description

Data from a cognitive experiment examining how partition priming affects probability judgments in a car dealership context. Participants judged probabilities under different framing conditions.

Usage

```
CarTask
```

Format

A data frame with 155 observations on 3 variables:

probability numeric. Estimated probability (response variable).

task factor with levels `Car` and `Salesperson` indicating the condition/question type.

NFCCscale numeric. Combined score from the Need for Closure (NFC) and Need for Certainty (NCC) scales, which are strongly correlated.

Details

All participants in the study were undergraduate students at The Australian National University, some of whom obtained course credit in first-year Psychology for their participation.

Task questions:

- **Car condition:** "What is the probability that a customer trades in a coupe?"
- **Salesperson condition:** "What is the probability that a customer buys a car from Carlos?" (out of four possible salespersons)

The key manipulation is the implicit partition: In the Car condition, there are multiple car types (binary: coupe vs. not coupe), while in the Salesperson condition, there are four specific salespersons. Classical findings suggest that different partition structures lead to different probability estimates even when the actual probabilities are equivalent.

The NFCC scale (Need for Closure and Certainty) measures individual differences in tolerance for ambiguity. Higher scores indicate greater need for definitive answers and discomfort with uncertainty.

Source

Taken from Smithson et al. (2011) supplements.

References

Smithson, M., Merkle, E.C., and Verkuilen, J. (2011). Beta Regression Finite Mixture Models of Polarization and Priming. *Journal of Educational and Behavioral Statistics*, **36**(6), 804–831. doi:10.3102/1076998610396893

Smithson, M., and Segale, C. (2009). Partition Priming in Judgments of Imprecise Probabilities. *Journal of Statistical Theory and Practice*, **3**(1), 169–181.

Examples

```
require(gkwreg)
require(gkwdist)

data(CarTask)

# Example 1: Task effects on probability judgments
# Do people judge probabilities differently for car vs. salesperson?
fit_kw <- gkwreg(
  probability ~ task,
  data = CarTask,
  family = "kw"
)
summary(fit_kw)

# Interpretation:
# - Alpha: Task type affects mean probability estimate
#   Salesperson condition (1/4 = 0.25) vs. car type (unclear baseline)

# Example 2: Individual differences model
# Need for Closure/Certainty may moderate probability judgments
fit_kw_nfcc <- gkwreg(
  probability ~ task * NFCCscale |
    task,
  data = CarTask,
  family = "kw"
)
summary(fit_kw_nfcc)

# Interpretation:
# - Interaction: NFCC may have different effects depending on task
#   People high in need for certainty may respond differently to
#   explicit partitions (4 salespersons) vs. implicit partitions (car types)
# - Beta: Precision varies by task type

# Example 3: Exponentiated Kumaraswamy for extreme estimates
# Some participants may give very extreme probability estimates
fit_ekw <- gkwreg(
  probability ~ task * NFCCscale | # alpha
    task | # beta
    task, # lambda: extremity differs by task
  data = CarTask,
  family = "ekw"
```

```
)
summary(fit_ekw)

# Interpretation:
# - Lambda varies by task: Salesperson condition (explicit partition)
#   may produce more extreme estimates (closer to 0 or 1)

# Visualization: Probability by task and NFCC
plot(probability ~ NFCCscale,
  data = CarTask,
  col = c("blue", "red")[task], pch = 19,
  xlab = "Need for Closure/Certainty", ylab = "Probability Estimate",
  main = "Car Task: Individual Differences in Probability Judgment"
)
legend("topright",
  legend = levels(CarTask$task),
  col = c("blue", "red"), pch = 19
)

# Distribution comparison
boxplot(probability ~ task,
  data = CarTask,
  xlab = "Task Condition", ylab = "Probability Estimate",
  main = "Partition Priming Effects",
  col = c("lightblue", "lightcoral")
)
abline(h = 0.25, lty = 2, col = "gray")
text(1.5, 0.27, "Uniform (1/4)", col = "gray")
```

`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.gkwreg *Confidence Intervals for Generalized Kumaraswamy Regression Parameters*

Description

Computes confidence intervals for model parameters in fitted gkwreg objects using Wald (normal approximation) method based on asymptotic theory.

Usage

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

Arguments

object	An object of class "gkwreg" from gkwreg.
parm	A specification of which parameters are to be given confidence intervals, either a vector of numbers or a vector of names. If missing, all parameters are considered.
level	The confidence level required. Default is 0.95.
...	Additional arguments (currently unused).

Details

The confidence intervals are computed using the Wald method based on asymptotic normality of maximum likelihood estimators:

$$CI = \hat{\theta} \pm z_{\alpha/2} \times SE(\hat{\theta})$$

where $z_{\alpha/2}$ is the appropriate normal quantile and $SE(\hat{\theta})$ is the standard error from the Hessian matrix.

The model must have been fitted with `hessian = TRUE` (the default) in `gkw_control`. If standard errors are not available, an error is raised.

Value

A matrix (or vector) with columns giving lower and upper confidence limits for each parameter. These will be labeled as (1-level)/2 and 1 - (1-level)/2 in percent (by default 2.5 percent and 97.5 percent).

Author(s)

Lopes, J. E.

See Also

`gkwreg`, `summary.gkwreg`, `confint`

Examples

```
data(GasolineYield)
fit <- gkwreg(yield ~ batch + temp, data = GasolineYield, family = "kw")

# 95 percent confidence intervals
confint(fit)

# 90 percent confidence intervals
confint(fit, level = 0.90)

# Specific parameters
confint(fit, parm = "alpha:(Intercept)")
confint(fit, parm = 1:3)
```

`family.gkwreg` *Extract Family from GKw Regression Model*

Description

Extracts the family specification from a fitted Generalized Kumaraswamy regression model object.

Usage

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

Arguments

`object`	An object of class `"gkwreg"`.
`...`	Currently not used.

Value

A character string indicating the family used in the model.

Author(s)

Lopes, J. E.

See Also

`gkwreg`

Examples

```
data(GasolineYield)
fit <- gkwreg(yield ~ batch + temp, data = GasolineYield, family = "kw")
family(fit)
```

`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

```
fitted.gkwreg(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

```
require(gkwreg)
require(gkwdist)

# Example 1: Basic usage with FoodExpenditure data
data(FoodExpenditure)
FoodExpenditure$prop <- FoodExpenditure$food / FoodExpenditure$income

fit_kw <- gkwreg(prop ~ income + persons | income,
  data = FoodExpenditure,
  family = "kw"
)

# Extract fitted values
fitted_vals <- fitted(fit_kw)

# Visualize fit quality
plot(FoodExpenditure$prop, fitted_vals,
  xlab = "Observed Proportion",
  ylab = "Fitted Values",
  main = "Observed vs Fitted: Food Expenditure",
  pch = 19, col = rgb(0, 0, 1, 0.5)
)
abline(0, 1, col = "red", lwd = 2)

# Calculate R-squared analogue
cor(FoodExpenditure$prop, fitted_vals)^2

# Example 2: Comparing fitted values across families
data(GasolineYield)

fit_ekw <- gkwreg(yield ~ batch + temp | temp | batch,
  data = GasolineYield,
  family = "ekw"
)

# Fitted values under different family assumptions
fitted_ekw <- fitted(fit_ekw)
```

```
fitted_kw <- fitted(fit_ekw, family = "kw")
fitted_beta <- fitted(fit_ekw, family = "beta")

# Compare differences
comparison <- data.frame(
  EKW = fitted_ekw,
  KW = fitted_kw,
  Beta = fitted_beta,
  Diff_EKW_KW = fitted_ekw - fitted_kw,
  Diff_EKW_Beta = fitted_ekw - fitted_beta
)
head(comparison)

# Visualize differences
par(mfrow = c(1, 2))
plot(fitted_ekw, fitted_kw,
  xlab = "EKW Fitted", ylab = "KW Fitted",
  main = "EKW vs KW Family Assumptions",
  pch = 19, col = "darkblue"
)
abline(0, 1, col = "red", lty = 2)

plot(fitted_ekw, fitted_beta,
  xlab = "EKW Fitted", ylab = "Beta Fitted",
  main = "EKW vs Beta Family Assumptions",
  pch = 19, col = "darkgreen"
)
abline(0, 1, col = "red", lty = 2)
par(mfrow = c(1, 1))

# Example 3: Diagnostic plot with confidence bands
data(ReadingSkills)

fit_mc <- gkwreg(
  accuracy ~ dyslexia * iq | dyslexia + iq | dyslexia,
  data = ReadingSkills,
  family = "mc"
)

fitted_vals <- fitted(fit_mc)

# Residual plot
residuals_resp <- ReadingSkills$accuracy - fitted_vals

plot(fitted_vals, residuals_resp,
  xlab = "Fitted Values",
  ylab = "Raw Residuals",
  main = "Residual Plot: Reading Accuracy",
  pch = 19, col = ReadingSkills$dyslexia,
  ylim = range(residuals_resp) * 1.2
)
abline(h = 0, col = "red", lwd = 2, lty = 2)
lowess_fit <- lowess(fitted_vals, residuals_resp)
```

```
lines(lowess_fit, col = "blue", lwd = 2)
legend("topright",
  legend = c("Control", "Dyslexic", "Zero Line", "Lowess"),
  col = c("black", "red", "red", "blue"),
  pch = c(19, 19, NA, NA),
  lty = c(NA, NA, 2, 1),
  lwd = c(NA, NA, 2, 2)
)

# Example 4: Large dataset efficiency check
set.seed(2024)
n <- 5000
x1 <- rnorm(n)
x2 <- runif(n, -2, 2)
alpha <- exp(0.3 + 0.5 * x1)
beta <- exp(1.2 - 0.4 * x2)
y <- rkw(n, alpha, beta)
large_data <- data.frame(y = y, x1 = x1, x2 = x2)

fit_large <- gkwreg(y ~ x1 | x2,
  data = large_data,
  family = "kw"
)

# Time the extraction
system.time({
  fitted_large <- fitted(fit_large)
})

# Verify extraction
length(fitted_large)
summary(fitted_large)
```

`FoodExpenditure` *Proportion of Household Income Spent on Food*

Description

Cross-section data on annual food expenditure and annual income for a random sample of households in a large U.S. city. The dataset models the proportion of income spent on food as a function of total income and household size.

Usage

```
FoodExpenditure
```

Format

A data frame with 38 observations on 3 variables:

food numeric. Annual food expenditure in U.S. dollars.

income numeric. Annual household income in U.S. dollars.

persons numeric. Number of persons in the household.

Details

This classic econometric dataset was taken from Griffiths et al. (1993, Table 15.4) who cite Leser (1963) as the original source. The data are used to model Engel curves, which describe how household expenditure on a particular good or service varies with household income.

The response variable of interest is typically `food/income`, the proportion of income spent on food, which follows beta distribution properties as it is bounded between 0 and 1.

Source

Taken from Griffiths et al. (1993, Table 15.4).

References

Cribari-Neto, F., and Zeileis, A. (2010). Beta Regression in R. *Journal of Statistical Software*, **34**(2), 1–24. doi:10.18637/jss.v034.i02

Ferrari, S.L.P., and Cribari-Neto, F. (2004). Beta Regression for Modeling Rates and Proportions. *Journal of Applied Statistics*, **31**(7), 799–815.

Griffiths, W.E., Hill, R.C., and Judge, G.G. (1993). *Learning and Practicing Econometrics*. New York: John Wiley and Sons.

Leser, C.E.V. (1963). Forms of Engel Functions. *Econometrica*, **31**(4), 694–703.

Examples

```
require(gkwreg)
require(gkwdist)

data(FoodExpenditure)
FoodExpenditure$prop <- FoodExpenditure$food / FoodExpenditure$income

# Example 1: Basic Kumaraswamy regression
# Proportion spent on food decreases with income (Engel's law)
# Larger households spend more on food
fit_kw <- gkwreg(prop ~ income + persons,
  data = FoodExpenditure,
  family = "kw"
)
summary(fit_kw)

# Interpretation:
# - Alpha: Negative income effect (Engel's law)
#   Positive household size effect
```

```
# - Beta: Constant precision (homoscedastic model)

# Example 2: Heteroscedastic model
# Variability in food proportion may differ by income and household size
fit_kw_hetero <- gkwreg(
  prop ~ income + persons |
    income + persons,
  data = FoodExpenditure,
  family = "kw"
)
summary(fit_kw_hetero)

# Interpretation:
# - Beta: Precision varies with both income and household size
#   Wealthier or larger households may show different spending variability

# Test for heteroscedasticity
anova(fit_kw, fit_kw_hetero)

# Example 3: Exponentiated Kumaraswamy for extreme spending patterns
# Some households may have unusual food spending (very frugal or lavish)
fit_ekw <- gkwreg(
  prop ~ income + persons | # alpha
    persons | # beta: household size affects precision
    income, # lambda: income affects extremity
  data = FoodExpenditure,
  family = "ekw"
)
summary(fit_ekw)

# Interpretation:
# - Lambda: Income level affects tail behavior
#   Rich households may show more extreme (unusual) spending patterns

# Visualization: Engel curve
plot(prop ~ income,
  data = FoodExpenditure,
  xlab = "Annual Income ($)", ylab = "Proportion Spent on Food",
  main = "Engel Curve for Food Expenditure"
)
# Add fitted values
FoodExpenditure$fitted_kw <- fitted(fit_kw)
points(FoodExpenditure$income, FoodExpenditure$fitted_kw,
  col = "blue", pch = 19, cex = 0.8
)
legend("topright",
  legend = c("Observed", "Fitted"),
  col = c("black", "blue"), pch = c(1, 19)
)
```

`formula.gkwreg` *Extract Formula from GKw Regression Model*

Description

Extracts the model formula from a fitted Generalized Kumaraswamy regression model object. Properly handles formulas with up to 5 parts.

Usage

```
## S3 method for class 'gkwreg'
formula(x, ...)
```

Arguments

`x`	An object of class `"gkwreg"`.
`...`	Currently not used.

Value

The formula used to fit the model. For multi-part formulas, returns an object of class `"Formula"`.

Author(s)

Lopes, J. E.

See Also

`gkwreg`, `update.gkwreg`

Examples

```
data(GasolineYield)

# Simple formula
fit1 <- gkwreg(yield ~ batch + temp, data = GasolineYield, family = "kw")
formula(fit1)

# Two-part formula
fit2 <- gkwreg(yield ~ temp | batch, data = GasolineYield, family = "kw")
formula(fit2)

# Five-part formula
fit3 <- gkwreg(yield ~ temp | batch | temp | 1 | 1,
  data = GasolineYield, family = "gkw"
)
formula(fit3)
```

GasolineYield *Gasoline Yield from Crude Oil*

Description

Operational data on the proportion of crude oil converted to gasoline after distillation and fractionation processes.

Usage

```
GasolineYield
```

Format

A data frame with 32 observations on 6 variables:

yield numeric. Proportion of crude oil converted to gasoline after distillation and fractionation (response variable).

gravity numeric. Crude oil gravity in degrees API (American Petroleum Institute scale).

pressure numeric. Vapor pressure of crude oil in pounds per square inch (psi).

temp10 numeric. Temperature in degrees Fahrenheit at which 10\ crude oil has vaporized.

temp numeric. Temperature in degrees Fahrenheit at which all gasoline has vaporized (end point).

batch factor. Batch indicator distinguishing the 10 different crude oils used in the experiment.

Details

This dataset was collected by Prater (1956) to study gasoline yield from crude oil. The dependent variable is the proportion of crude oil after distillation and fractionation. Atkinson (1985) analyzed this dataset using linear regression and noted that there is "indication that the error distribution is not quite symmetrical, giving rise to some unduly large and small residuals".

The dataset contains 32 observations. It has been noted (Daniel and Wood, 1971, Chapter 8) that there are only ten sets of values of the first three explanatory variables which correspond to ten different crudes subjected to experimentally controlled distillation conditions. These conditions are captured in variable `batch` and the data were ordered according to the ascending order of `temp10`.

Source

Taken from Prater (1956).

References

Atkinson, A.C. (1985). *Plots, Transformations and Regression: An Introduction to Graphical Methods of Diagnostic Regression Analysis*. New York: Oxford University Press.

Cribari-Neto, F., and Zeileis, A. (2010). Beta Regression in R. *Journal of Statistical Software*, **34**(2), 1–24. doi:10.18637/jss.v034.i02

Daniel, C., and Wood, F.S. (1971). *Fitting Equations to Data*. New York: John Wiley and Sons.

Ferrari, S.L.P., and Cribari-Neto, F. (2004). Beta Regression for Modeling Rates and Proportions. *Journal of Applied Statistics*, **31**(7), 799–815.

Prater, N.H. (1956). Estimate Gasoline Yields from Crudes. *Petroleum Refiner*, **35**(5), 236–238.

Examples

```
require(gkwreg)
require(gkwdist)

data(GasolineYield)

# Example 1: Kumaraswamy regression with batch effects
# Model mean yield as function of batch and temperature
# Allow precision to vary with temperature (heteroscedasticity)
fit_kw <- gkwreg(yield ~ batch + temp | temp,
  data = GasolineYield,
  family = "kw"
)
summary(fit_kw)

# Interpretation:
# - Alpha (mean): Different batches have different baseline yields
#   Temperature affects yield transformation
# - Beta (precision): Higher temperatures may produce more variable yields

# Example 2: Full model with all physical-chemical properties
fit_kw_full <- gkwreg(
  yield ~ gravity + pressure + temp10 + temp |
    temp10 + temp,
  data = GasolineYield,
  family = "kw"
)
summary(fit_kw_full)

# Interpretation:
# - Mean model captures effects of crude oil properties
# - Precision varies with vaporization temperatures

# Example 3: Exponentiated Kumaraswamy for extreme yields
# Some batches may produce unusually high/low yields
fit_ekw <- gkwreg(
  yield ~ batch + temp | # alpha: batch effects
    temp | # beta: temperature precision
    batch, # lambda: batch-specific tail behavior
  data = GasolineYield,
  family = "ekw"
)
summary(fit_ekw)

# Interpretation:
# - Lambda varies by batch: Some crude oils have more extreme
```

```
#   yield distributions (heavy tails for very high/low yields)

# Model comparison: Does tail flexibility improve fit?
anova(fit_kw, fit_ekw)

# Diagnostic plots
par(mfrow = c(2, 2))
plot(fit_kw, which = c(1, 2, 4, 5))
par(mfrow = c(1, 1))
```

`getCall.gkwreg` *Get Call from GKw Regression Model*

Description

Extracts the call that was used to fit a Generalized Kumaraswamy regression model.

Usage

```
## S3 method for class 'gkwreg'
getCall(x, ...)
```

Arguments

`x`	An object of class `"gkwreg"`.
`...`	Currently not used.

Value

The matched call.

Author(s)

Lopes, J. E.

See Also

`gkwreg`, `update.gkwreg`

Examples

```
data(GasolineYield)
fit <- gkwreg(yield ~ batch + temp, data = GasolineYield, family = "kw")
getCall(fit)
```

gkwreg *Fit Generalized Kumaraswamy Regression Models*

Description

Fits regression models using the Generalized Kumaraswamy (GKw) family of distributions for modeling response variables strictly bounded in the interval (0, 1). The function provides a unified interface for fitting seven nested submodels of the GKw family, allowing flexible modeling of proportions, rates, and other bounded continuous outcomes through regression on distributional parameters.

Maximum Likelihood Estimation is performed via automatic differentiation using the TMB (Template Model Builder) package, ensuring computational efficiency and numerical accuracy. The interface follows standard R regression modeling conventions similar to `lm`, `glm`, and `betareg`, making it immediately familiar to R users.

Usage

```
gkwreg(
  formula,
  data,
  family = c("gkw", "bkw", "kkw", "ekw", "mc", "kw", "beta"),
  link = NULL,
  link_scale = NULL,
  subset = NULL,
  weights = NULL,
  offset = NULL,
  na.action = getOption("na.action"),
  contrasts = NULL,
  control = gkw_control(),
  model = TRUE,
  x = FALSE,
  y = TRUE,
  ...
)
```

Arguments

`formula` An object of class `Formula` (or one that can be coerced to that class). The formula uses extended syntax to specify potentially different linear predictors for each distribution parameter:

`y ~ model_alpha | model_beta | model_gamma | model_delta | model_lambda`

where:

- `y` is the response variable (must be in the open interval (0, 1))
- `model_alpha` specifies predictors for the α parameter
- `model_beta` specifies predictors for the β parameter
- `model_gamma` specifies predictors for the γ parameter

- `model_delta` specifies predictors for the δ parameter
- `model_lambda` specifies predictors for the λ parameter

If a part is omitted or specified as `~ 1`, an intercept-only model is used for that parameter. Parts corresponding to fixed parameters (determined by `family`) are automatically ignored. See Details and Examples for proper usage.

`data` A data frame containing the variables specified in `formula`. Standard R subsetting and missing value handling apply.

`family` A character string specifying the distribution family from the Generalized Kumaraswamy hierarchy. Must be one of:

`"gkw"` Generalized Kumaraswamy (default). Five parameters: $\alpha, \beta, \gamma, \delta, \lambda$. Most flexible, suitable when data show complex behavior not captured by simpler families.

`"bkw"` Beta-Kumaraswamy. Four parameters: $\alpha, \beta, \gamma, \delta$ (fixes $\lambda = 1$). Combines Beta and Kumaraswamy flexibility.

`"kkw"` Kumaraswamy-Kumaraswamy. Four parameters: $\alpha, \beta, \delta, \lambda$ (fixes $\gamma = 1$). Alternative four-parameter generalization.

`"ekw"` Exponentiated Kumaraswamy. Three parameters: α, β, λ (fixes $\gamma = 1, \delta = 0$). Adds flexibility to standard Kumaraswamy.

`"mc"` McDonald (Beta Power). Three parameters: γ, δ, λ (fixes $\alpha = 1, \beta = 1$). Generalization of Beta distribution.

`"kw"` Kumaraswamy. Two parameters: α, β (fixes $\gamma = 1, \delta = 0, \lambda = 1$). Computationally efficient alternative to Beta with closed-form CDF.

`"beta"` Beta distribution. Two parameters: γ, δ (fixes $\alpha = 1, \beta = 1, \lambda = 1$). Standard choice for proportions and rates, corresponds to shape1 = γ, shape2 = δ.

See Details for guidance on family selection.

`link` Link function(s) for the distributional parameters. Can be specified as:

- **Single character string**: Same link for all relevant parameters. Example: `link = "log"` applies log link to all parameters.
- **Named list**: Parameter-specific links for fine control. Example: `link = list(alpha = "log", beta = "log", delta = "logit")`

Default links (used if `link = NULL`):

- `"log"` for $\alpha, \beta, \gamma, \lambda$ (positive parameters)
- `"logit"` for δ (parameter in (0, 1))

Available link functions:

`"log"` Logarithmic link. Maps $(0, \infty) \to (-\infty, \infty)$. Ensures positivity. Most common for shape parameters.

`"logit"` Logistic link. Maps $(0, 1) \to (-\infty, \infty)$. Standard for probability-type parameters like δ.

`"probit"` Probit link using normal CDF. Maps $(0, 1) \to (-\infty, \infty)$. Alternative to logit, symmetric tails.

`"cloglog"` Complementary log-log. Maps $(0, 1) \to (-\infty, \infty)$. Asymmetric, useful for skewed probabilities.

"cauchy" Cauchy link using Cauchy CDF. Maps $(0, 1) \to (-\infty, \infty)$. Heavy-tailed alternative to probit.

"identity" Identity link (no transformation). Use with caution; does not guarantee parameter constraints.

"sqrt" Square root link. Maps $x \to \sqrt{x}$. Variance-stabilizing for some contexts.

"inverse" Inverse link. Maps $x \to 1/x$. Useful for rate-type parameters.

"inverse-square" Inverse squared link. Maps $x \to 1/x^2$.

link_scale Numeric scale factor(s) controlling the transformation intensity of link functions. Can be:

- **Single numeric**: Same scale for all parameters.
- **Named list**: Parameter-specific scales for fine-tuning. Example: link_scale = list(alpha = 10, beta = 10, delta = 1)

Default scales (used if link_scale = NULL):

- 10 for $\alpha, \beta, \gamma, \lambda$
- 1 for δ

Larger values produce more gradual transformations; smaller values produce more extreme transformations. For probability-type links (logit, probit), smaller scales (e.g., 0.5-2) create steeper response curves, while larger scales (e.g., 5-20) create gentler curves. Adjust if convergence issues arise or if you need different response sensitivities.

subset Optional vector specifying a subset of observations to be used in fitting. Can be a logical vector, integer indices, or expression evaluating to one of these. Standard R subsetting rules apply.

weights Optional numeric vector of prior weights (e.g., frequency weights) for observations. Should be non-negative. Currently experimental; use with caution and validate results.

offset Optional numeric vector or matrix specifying an *a priori* known component to be included in the linear predictor(s). If a vector, it is applied to the first parameter's predictor. If a matrix, columns correspond to parameters in order $(\alpha, \beta, \gamma, \delta, \lambda)$. Offsets are added to the linear predictor *before* applying the link function.

na.action A function specifying how to handle missing values (NAs). Options include:

na.fail Stop with error if NAs present (default via getOption("na.action"))

na.omit Remove observations with NAs

na.exclude Like na.omit but preserves original length in residuals/fitted values

See na.action for details.

contrasts Optional list specifying contrasts for factor variables in the model. Format: named list where names are factor variable names and values are contrast specifications. See contrasts and the contrasts.arg argument of model.matrix.

control A list of control parameters from gkw_control specifying technical details of the fitting process. This includes:

- Optimization algorithm (method)

- Starting values (start)
- Fixed parameters (fixed)
- Convergence tolerances (maxit, reltol, abstol)
- Hessian computation (hessian)
- Verbosity (silent, trace)

Default is gkw_control() which uses sensible defaults for most problems. See gkw_control for complete documentation of all options. **Most users never need to modify control parameters.**

model — Logical. If TRUE (default), the model frame (data frame containing all variables used in fitting) is returned as component model of the result. Useful for prediction and diagnostics. Set to FALSE to reduce object size.

x — Logical. If TRUE, the list of model matrices (one for each modeled parameter) is returned as component x. Default FALSE. Set to TRUE if you need direct access to design matrices for custom calculations.

y — Logical. If TRUE (default), the response vector (after processing by na.action and subset) is returned as component y. Useful for residual calculations and diagnostics.

... — Additional arguments. Currently used only for backward compatibility with deprecated arguments from earlier versions. Using deprecated arguments triggers informative warnings with migration guidance. Examples of deprecated arguments: plot, conf.level, method, start, fixed, hessian, silent, optimizer.control. These should now be passed via the control argument.

Details

Distribution Family Selection:

The Generalized Kumaraswamy family provides a flexible hierarchy for modeling bounded responses. Selection should be guided by:

1. Start Simple: Begin with two-parameter families ("kw" or "beta") unless you have strong reasons to use more complex models.

2. Model Comparison: Use information criteria (AIC, BIC) and likelihood ratio tests to compare nested models:

```
# Fit sequence of nested models
fit_kw   <- gkwreg(y ~ x, data, family = "kw")
fit_ekw  <- gkwreg(y ~ x, data, family = "ekw")
fit_gkw  <- gkwreg(y ~ x, data, family = "gkw")

# Compare via AIC
AIC(fit_kw, fit_ekw, fit_gkw)

# Formal test (nested models only)
anova(fit_kw, fit_ekw, fit_gkw)
```

3. Family Characteristics:

- **Beta**: Traditional choice, well-understood, good for symmetric or moderately skewed data

- **Kumaraswamy (kw)**: Computationally efficient alternative to Beta, closed-form CDF, similar flexibility
- **Exponentiated Kumaraswamy (ekw)**: Adds flexibility for extreme values and heavy tails
- **Beta-Kumaraswamy (bkw)**, **Kumaraswamy-Kumaraswamy (kkw)**: Four-parameter alternatives when three parameters insufficient
- **McDonald (mc)**: Beta generalization via power parameter, useful for J-shaped distributions
- **Kumaraswamy-Kumaraswamy (kkw)**: Most flexible, use only when simpler families inadequate. It extends kw
- **Generalized Kumaraswamy (gkw)**: Most flexible, use only when simpler families inadequate

4. Avoid Overfitting: More different parameters better model. Use cross-validation or hold-out validation to assess predictive performance.

Formula Specification:

The extended formula syntax allows different predictors for each parameter:

Basic Examples:

```
# Same predictors for both parameters (two-parameter family)
y ~ x1 + x2
# Equivalent to: y ~ x1 + x2 | x1 + x2

# Different predictors per parameter
y ~ x1 + x2 | x3 + x4
# alpha depends on x1, x2
# beta depends on x3, x4

# Intercept-only for some parameters
y ~ x1 | 1
# alpha depends on x1
# beta has only intercept

# Complex specification (five-parameter family)
y ~ x1 | x2 | x3 | x4 | x5
# alpha ~ x1, beta ~ x2, gamma ~ x3, delta ~ x4, lambda ~ x5
```

Important Notes:

- Formula parts correspond to parameters in order: α, β, γ, δ, λ
- Unused parts (due to family constraints) are automatically ignored
- Use `.` to include all predictors: `y ~ . | .`
- Standard R formula features work: interactions (`x1:x2`), polynomials (`poly(x, 2)`), transformations (`log(x)`), etc.

Link Functions and Scales:

Link functions map the range of distributional parameters to the real line, ensuring parameter constraints are satisfied during optimization.

Choosing Links:

- **Defaults are usually best**: The automatic choices (log for shape parameters, logit for delta) work well in most cases

- **Alternative links**: Consider if you have theoretical reasons (e.g., probit for latent variable interpretation) or convergence issues
- **Identity link**: Avoid unless you have constraints elsewhere; can lead to invalid parameter values during optimization

Link Scales: The `link_scale` parameter controls transformation intensity. Think of it as a "sensitivity" parameter:

- **Larger values** (e.g., 20): Gentler response to predictor changes
- **Smaller values** (e.g., 2): Steeper response to predictor changes
- **Default (10)**: Balanced, works well for most cases

Adjust only if:

- Convergence difficulties arise
- You need very steep or very gentle response curves
- Predictors have unusual scales (very large or very small)

Optimization and Convergence:

The default optimizer (`method = "nlminb"`) works well for most problems. If convergence issues occur:

1. Check Data:

- Ensure response is strictly in (0, 1)
- Check for extreme outliers or influential points
- Verify predictors aren't perfectly collinear
- Consider rescaling predictors to similar ranges

2. Try Alternative Optimizers:

```
# BFGS often more robust for difficult problems
fit <- gkwreg(y ~ x, data,
              control = gkw_control(method = "BFGS"))

# Nelder-Mead for non-smooth objectives
fit <- gkwreg(y ~ x, data,
              control = gkw_control(method = "Nelder-Mead"))
```

3. Adjust Tolerances:

```
# Increase iterations and loosen tolerance
fit <- gkwreg(y ~ x, data,
              control = gkw_control(maxit = 1000, reltol = 1e-6))
```

4. Provide Starting Values:

```
# Fit simpler model first, use as starting values
fit_simple <- gkwreg(y ~ 1, data, family = "kw")
start_vals <- list(
  alpha = c(coef(fit_simple)[1], rep(0, ncol(X_alpha) - 1)),
  beta  = c(coef(fit_simple)[2], rep(0, ncol(X_beta) - 1))
)
fit_complex <- gkwreg(y ~ x1 + x2 | x3 + x4, data, family = "kw",
                      control = gkw_control(start = start_vals))
```

5. Simplify Model:

- Use simpler family (e.g., "kw" instead of "gkw")
- Reduce number of predictors
- Use intercept-only for some parameters

Standard Errors and Inference:

By default, standard errors are computed via the Hessian matrix at the MLE. This provides valid asymptotic standard errors under standard regularity conditions.

When Standard Errors May Be Unreliable:

- Small sample sizes (n < 30-50 per parameter)
- Parameters near boundaries
- Highly collinear predictors
- Mis-specified models

Alternatives:

- Bootstrap confidence intervals (more robust, computationally expensive)
- Profile likelihood intervals via `confint(..., type = "profile")` (not yet implemented)
- Cross-validation for predictive performance assessment

To skip Hessian computation (faster, no SEs):

```
fit <- gkwreg(y ~ x, data,
              control = gkw_control(hessian = FALSE))
```

Model Diagnostics:

Always check model adequacy using diagnostic plots:

```
fit <- gkwreg(y ~ x, data, family = "kw")
plot(fit)  # Six diagnostic plots
```

Key diagnostics:

- **Residual plots**: Check for patterns, heteroscedasticity
- **Half-normal plot**: Assess distributional adequacy
- **Cook's distance**: Identify influential observations
- **Predicted vs observed**: Overall fit quality

See `plot.gkwreg` for detailed interpretation guidance.

Computational Considerations:

Performance Tips:

- GKw family: Most computationally expensive (~2-5x slower than kw/beta)
- Beta/Kw families: Fastest, use when adequate
- Large datasets (n > 10,000): Consider sampling for exploratory analysis
- TMB uses automatic differentiation: Fast gradient/Hessian computation
- Disable Hessian (`hessian = FALSE`) for faster fitting without SEs

Memory Usage:

- Set `model = FALSE`, `x = FALSE`, `y = FALSE` to reduce object size (but limits some post-fitting capabilities)
- Hessian matrix scales as $O(p^2)$ where p = number of parameters

Value

An object of class "gkwreg", which is a list containing the following components. Standard S3 methods are available for this class (see Methods section).

Model Specification:

call The matched function call

formula The Formula object used

family Character string: distribution family used

link Named list: link functions for each parameter

link_scale Named list: link scale values for each parameter

param_names Character vector: names of parameters for this family

fixed_params Named list: parameters fixed by family definition

control The gkw_control object used for fitting

Parameter Estimates:

coefficients Named numeric vector: estimated regression coefficients (on link scale). Names follow the pattern "parameter:predictor", e.g., "alpha:(Intercept)", "alpha:x1", "beta:(Intercept)", "beta:x2".

fitted_parameters Named list: mean values for each distribution parameter ($\alpha, \beta, \gamma, \delta, \lambda$) averaged across all observations

parameter_vectors Named list: observation-specific parameter values. Contains vectors alphaVec, betaVec, gammaVec, deltaVec, lambdaVec, each of length nobs

Fitted Values and Residuals:

fitted.values Numeric vector: fitted mean values $E[Y|X]$ for each observation

residuals Numeric vector: response residuals (observed - fitted) for each observation

Inference:

vcov Variance-covariance matrix of coefficient estimates. Only present if control$hessian = TRUE. NULL otherwise.

se Numeric vector: standard errors of coefficients. Only present if control$hessian = TRUE. NULL otherwise.

Model Fit Statistics:

loglik Numeric: maximized log-likelihood value

aic Numeric: Akaike Information Criterion (AIC = -2*loglik* + 2npar)

bic Numeric: Bayesian Information Criterion (BIC = -2*loglik + log(nobs)*npar)

deviance Numeric: deviance (-2 * loglik)

df.residual Integer: residual degrees of freedom (nobs - npar)

nobs Integer: number of observations used in fit

npar Integer: total number of estimated parameters

Diagnostic Statistics:

`rmse` Numeric: Root Mean Squared Error of response residuals

`efron_r2` Numeric: Efron's pseudo R-squared (1 - SSE/SST, where SSE = sum of squared errors, SST = total sum of squares)

`mean_absolute_error` Numeric: Mean Absolute Error of response residuals

Optimization Details:

`convergence` Logical: `TRUE` if optimizer converged successfully, `FALSE` otherwise

`message` Character: convergence message from optimizer

`iterations` Integer: number of iterations used by optimizer

`method` Character: optimization method used (e.g., "nlminb", "BFGS")

Optional Components (returned if requested via `model`, `x`, `y`):

`model` Data frame: the model frame (if `model = TRUE`)

`x` Named list: model matrices for each parameter (if `x = TRUE`)

`y` Numeric vector: the response variable (if `y = TRUE`)

Internal:

`tmb_object` The raw object returned by `MakeADFun`. Contains the TMB automatic differentiation function and environment. Primarily for internal use and advanced debugging.

Methods

The following S3 methods are available for objects of class `"gkwreg"`:

Basic Methods:

- `print.gkwreg`: Print basic model information
- `summary.gkwreg`: Detailed model summary with coefficient tables, tests, and fit statistics
- `coef.gkwreg`: Extract coefficients
- `vcov.gkwreg`: Extract variance-covariance matrix
- `logLik`: Extract log-likelihood
- `AIC`, `BIC`: Information criteria

Prediction and Fitted Values:

- `fitted.gkwreg`: Extract fitted values
- `residuals.gkwreg`: Extract residuals (multiple types available)
- `predict.gkwreg`: Predict on new data

Inference:

- `confint.gkwreg`: Confidence intervals for parameters
- `anova.gkwreg`: Compare nested models via likelihood ratio tests

Diagnostics:

- `plot.gkwreg`: Comprehensive diagnostic plots (6 types)

Author(s)

Lopes, J. E.

Maintainer: Lopes, J. E.

References

Generalized Kumaraswamy Distribution:

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

Kumaraswamy Distribution:

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

Jones, M. C. (2009). Kumaraswamy's distribution: A beta-type distribution with some tractability advantages. *Statistical Methodology*, **6**(1), 70-81. doi:10.1016/j.stamet.2008.04.001

Beta Regression:

Ferrari, S. L. P., & Cribari-Neto, F. (2004). Beta regression for modelling rates and proportions. *Journal of Applied Statistics*, **31**(7), 799-815. doi:10.1080/0266476042000214501

Smithson, M., & Verkuilen, J. (2006). A better lemon squeezer? Maximum-likelihood regression with beta-distributed dependent variables. *Psychological Methods*, **11**(1), 54-71. doi:10.1037/1082-989X.11.1.54

Template Model Builder (TMB):

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

Related Software:

Zeileis, A., & Croissant, Y. (2010). Extended Model Formulas in R: Multiple Parts and Multiple Responses. *Journal of Statistical Software*, **34**(1), 1-13. doi:10.18637/jss.v034.i01

See Also

Control and Inference: `gkw_control` for fitting control parameters, `confint.gkwreg` for confidence intervals, `anova.gkwreg` for model comparison

Methods: `summary.gkwreg`, `plot.gkwreg`, `coef.gkwreg`, `vcov.gkwreg`, `fitted.gkwreg`, `residuals.gkwreg`, `predict.gkwreg`

Distributions: `dgkw`, `pgkw`, `qgkw`, `rgkw` for the GKw distribution family functions

Related Packages: `betareg` for traditional beta regression, `Formula` for extended formula interface, `MakeADFun` for TMB functionality

Examples

```
# SECTION 1: Basic Usage - Getting Started
# Load packages and data
library(gkwreg)
library(gkwdist)
```

```
data(GasolineYield)

# Example 1.1: Simplest possible model (intercept-only, all defaults)
fit_basic <- gkwreg(yield ~ 1, data = GasolineYield, family = "kw")
summary(fit_basic)

# Example 1.2: Model with predictors (uses all defaults)
# Default: family = "gkw", method = "nlminb", hessian = TRUE
fit_default <- gkwreg(yield ~ batch + temp, data = GasolineYield)
summary(fit_default)

# Example 1.3: Kumaraswamy model (two-parameter family)
# Default link functions: log for both alpha and beta
fit_kw <- gkwreg(yield ~ batch + temp, data = GasolineYield, family = "kw")
summary(fit_kw)

par(mfrow = c(3, 2))
plot(fit_kw, ask = FALSE)

# Example 1.4: Beta model for comparison
# Default links: log for gamma and delta
fit_beta <- gkwreg(yield ~ batch + temp, data = GasolineYield, family = "beta")

# Compare models using AIC/BIC
AIC(fit_kw, fit_beta)
BIC(fit_kw, fit_beta)

# SECTION 2: Using gkw_control() for Customization

# Example 2.1: Change optimization method to BFGS
fit_bfgs <- gkwreg(
  yield ~ batch + temp,
  data = GasolineYield,
  family = "kw",
  control = gkw_control(method = "BFGS")
)
summary(fit_bfgs)

# Example 2.2: Increase iterations and enable verbose output
fit_verbose <- gkwreg(
  yield ~ batch + temp,
  data = GasolineYield,
  family = "kw",
  control = gkw_control(
    method = "nlminb",
    maxit = 1000,
    silent = FALSE, # Show optimization progress
    trace = 1 # Print iteration details
  )
)

# Example 2.3: Fast fitting without standard errors
# Useful for model exploration or large datasets
```

```
fit_fast <- gkwreg(
  yield ~ batch + temp,
  data = GasolineYield,
  family = "kw",
  control = gkw_control(hessian = FALSE)
)
# Note: Cannot compute confint() without hessian
coef(fit_fast) # Point estimates still available

# Example 2.4: Custom convergence tolerances
fit_tight <- gkwreg(
  yield ~ batch + temp,
  data = GasolineYield,
  family = "kw",
  control = gkw_control(
    reltol = 1e-10, # Tighter convergence
    maxit = 2000 # More iterations allowed
  )
)

# SECTION 3: Advanced Formula Specifications

# Example 3.1: Different predictors for different parameters
# alpha depends on batch, beta depends on temp
fit_diff <- gkwreg(
  yield ~ batch | temp,
  data = GasolineYield,
  family = "kw"
)
summary(fit_diff)

# Example 3.2: Intercept-only for one parameter
# alpha varies with predictors, beta is constant
fit_partial <- gkwreg(
  yield ~ batch + temp | 1,
  data = GasolineYield,
  family = "kw"
)

# Example 3.3: Complex model with interactions
fit_interact <- gkwreg(
  yield ~ batch * temp | temp + I(temp^2),
  data = GasolineYield,
  family = "kw"
)

# SECTION 4: Working with Different Families

# Example 4.1: Fit multiple families and compare
families <- c("beta", "kw", "ekw", "bkw", "gkw")
fits <- lapply(families, function(fam) {
  gkwreg(yield ~ batch + temp, data = GasolineYield, family = fam)
})
```

```
names(fits) <- families

# Compare via information criteria
comparison <- data.frame(
  Family = families,
  LogLik = sapply(fits, logLik),
  AIC = sapply(fits, AIC),
  BIC = sapply(fits, BIC),
  npar = sapply(fits, function(x) x$npar)
)
print(comparison)

# Example 4.2: Formal nested model testing
fit_kw <- gkwreg(yield ~ batch + temp, GasolineYield, family = "kw")
fit_ekw <- gkwreg(yield ~ batch + temp, GasolineYield, family = "ekw")
fit_gkw <- gkwreg(yield ~ batch + temp, GasolineYield, family = "gkw")
anova(fit_kw, fit_ekw, fit_gkw)

# SECTION 5: Link Functions and Scales

# Example 5.1: Custom link functions
fit_links <- gkwreg(
  yield ~ batch + temp,
  data = GasolineYield,
  family = "kw",
  link = list(alpha = "sqrt", beta = "log")
)

# Example 5.2: Custom link scales
# Smaller scale = steeper response curve
fit_scale <- gkwreg(
  yield ~ batch + temp,
  data = GasolineYield,
  family = "kw",
  link_scale = list(alpha = 5, beta = 15)
)

# Example 5.3: Uniform link for all parameters
fit_uniform <- gkwreg(
  yield ~ batch + temp,
  data = GasolineYield,
  family = "kw",
  link = "log" # Single string applied to all
)

# SECTION 6: Prediction and Inference

# Fit model for prediction examples
fit <- gkwreg(yield ~ batch + temp, GasolineYield, family = "kw")

# Example 6.1: Confidence intervals at different levels
confint(fit, level = 0.95) # 95% CI
confint(fit, level = 0.90) # 90% CI
```

```
confint(fit, level = 0.99) # 99% CI

# SECTION 7: Diagnostic Plots and Model Checking

fit <- gkwreg(yield ~ batch + temp, GasolineYield, family = "kw")

# Example 7.1: All diagnostic plots (default)
par(mfrow = c(3, 2))
plot(fit, ask = FALSE)

# Example 7.2: Select specific plots
par(mfrow = c(3, 1))
plot(fit, which = c(2, 4, 5)) # Cook's distance, Residuals, Half-normal

# Example 7.3: Using ggplot2 for modern graphics
plot(fit, use_ggplot = TRUE, arrange_plots = TRUE)

# Example 7.4: Customized half-normal plot
par(mfrow = c(1, 1))
plot(fit,
  which = 5,
  type = "quantile",
  nsim = 200, # More simulations for smoother envelope
  level = 0.95
) # 95% confidence envelope

# Example 7.5: Extract diagnostic data programmatically
diagnostics <- plot(fit, save_diagnostics = TRUE)
head(diagnostics$data) # Residuals, Cook's distance, etc.

# SECTION 8: Real Data Example - Food Expenditure

# Load and prepare data
data(FoodExpenditure, package = "betareg")
food_data <- FoodExpenditure
food_data$prop <- food_data$food / food_data$income

# Example 8.1: Basic model
fit_food <- gkwreg(
  prop ~ persons | income,
  data = food_data,
  family = "kw"
)
summary(fit_food)

# Example 8.2: Compare with Beta regression
fit_food_beta <- gkwreg(
  prop ~ persons | income,
  data = food_data,
  family = "beta"
)

# Which fits better?
```

```
AIC(fit_food, fit_food_beta)

# Example 8.3: Model diagnostics
par(mfrow = c(3, 1))
plot(fit_food, which = c(2, 5, 6))

# Example 8.4: Interpretation via effects
# How does proportion spent on food change with income?
income_seq <- seq(min(food_data$income), max(food_data$income), length = 50)
pred_data <- data.frame(
  persons = median(food_data$persons),
  income = income_seq
)
pred_food <- predict(fit_food, newdata = pred_data, type = "response")

par(mfrow = c(1, 1))
plot(food_data$income, food_data$prop,
  xlab = "Income", ylab = "Proportion Spent on Food",
  main = "Food Expenditure Pattern"
)
lines(income_seq, pred_food, col = "red", lwd = 2)

# SECTION 9: Simulation Studies

# Example 9.1: Simple Kumaraswamy simulation
set.seed(123)
n <- 500
x1 <- runif(n, -2, 2)
x2 <- rnorm(n)

# True model: log(alpha) = 0.8 + 0.3*x1, log(beta) = 1.2 - 0.2*x2
eta_alpha <- 0.8 + 0.3 * x1
eta_beta <- 1.2 - 0.2 * x2
alpha_true <- exp(eta_alpha)
beta_true <- exp(eta_beta)

# Generate response
y <- rkw(n, alpha = alpha_true, beta = beta_true)
sim_data <- data.frame(y = y, x1 = x1, x2 = x2)

# Fit and check parameter recovery
fit_sim <- gkwreg(y ~ x1 | x2, data = sim_data, family = "kw")

# Compare estimated vs true coefficients
cbind(
  True = c(0.8, 0.3, 1.2, -0.2),
  Estimated = coef(fit_sim),
  SE = fit_sim$se
)

# Example 9.2: Complex simulation with all five parameters
set.seed(2203)
n <- 2000
```

```
x <- runif(n, -1, 1)

# True parameters
alpha <- exp(0.5 + 0.3 * x)
beta <- exp(1.0 - 0.2 * x)
gamma <- exp(0.7 + 0.4 * x)
delta <- plogis(0.0 + 0.5 * x) # logit scale
lambda <- exp(-0.3 + 0.2 * x)

# Generate from GKw
y <- rgkw(n,
  alpha = alpha, beta = beta, gamma = gamma,
  delta = delta, lambda = lambda
)
sim_data2 <- data.frame(y = y, x = x)

# Fit GKw model
fit_gkw <- gkwreg(
  y ~ x | x | x | x | x,
  data = sim_data2,
  family = "gkw",
  control = gkw_control(method = "L-BFGS-B", maxit = 2000)
)
summary(fit_gkw)

# SECTION 10: Handling Convergence Issues

# Example 10.1: Try different optimizers
methods <- c("nlminb", "BFGS", "Nelder-Mead", "CG")
fits_methods <- lapply(methods, function(m) {
  tryCatch(
    gkwreg(yield ~ batch + temp, GasolineYield,
      family = "kw",
      control = gkw_control(method = m, silent = TRUE)
    ),
    error = function(e) NULL
  )
})
names(fits_methods) <- methods

# Check which converged
converged <- sapply(fits_methods, function(f) {
  if (is.null(f)) {
    return(FALSE)
  }
  f$convergence
})
print(converged)

# Example 10.2: Verbose mode for debugging
fit_debug <- gkwreg(
  yield ~ batch + temp,
  data = GasolineYield,
```

```
  family = "kw",
  control = gkw_control(
    method = "BFGS",
    silent = TRUE,
    trace = 0, # 2, Maximum verbosity
    maxit = 1000
  )
)

# SECTION 11: Memory and Performance Optimization

# Example 11.1: Minimal object for large datasets
fit_minimal <- gkwreg(
  yield ~ batch + temp,
  data = GasolineYield,
  family = "kw",
  model = FALSE, # Don't store model frame
  x = FALSE, # Don't store design matrices
  y = FALSE, # Don't store response
  control = gkw_control(hessian = FALSE) # Skip Hessian
)

# Much smaller object
object.size(fit_minimal)

# Trade-off: Limited post-fitting capabilities
# Can still use: coef(), logLik(), AIC(), BIC()
# Cannot use: predict(), some diagnostics

# Example 11.2: Fast exploratory analysis
# Fit many models quickly without standard errors
formulas <- list(
  yield ~ batch,
  yield ~ temp,
  yield ~ batch + temp,
  yield ~ batch * temp
)

fast_fits <- lapply(formulas, function(f) {
  gkwreg(f, GasolineYield,
    family = "kw",
    control = gkw_control(hessian = FALSE),
    model = FALSE, x = FALSE, y = FALSE
  )
})

# Compare models via AIC
sapply(fast_fits, AIC)

# Refit best model with full inference
best_formula <- formulas[[which.min(sapply(fast_fits, AIC))]]
fit_final <- gkwreg(best_formula, GasolineYield, family = "kw")
summary(fit_final)
```

```
# SECTION 12: Model Selection and Comparison

# Example 12.1: Nested model testing
fit1 <- gkwreg(yield ~ 1, GasolineYield, family = "kw")
fit2 <- gkwreg(yield ~ batch, GasolineYield, family = "kw")
fit3 <- gkwreg(yield ~ batch + temp, GasolineYield, family = "kw")

# Likelihood ratio tests
anova(fit1, fit2, fit3)

# Example 12.2: Information criteria table
models <- list(
  "Intercept only" = fit1,
  "Batch effect" = fit2,
  "Batch + Temp" = fit3
)

ic_table <- data.frame(
  Model = names(models),
  df = sapply(models, function(m) m$npar),
  LogLik = sapply(models, logLik),
  AIC = sapply(models, AIC),
  BIC = sapply(models, BIC),
  Delta_AIC = sapply(models, AIC) - min(sapply(models, AIC))
)
print(ic_table)

# Example 12.3: Cross-validation for predictive performance
# 5-fold cross-validation
set.seed(2203)
n <- nrow(GasolineYield)
folds <- sample(rep(1:5, length.out = n))

cv_rmse <- sapply(1:5, function(fold) {
  train <- GasolineYield[folds != fold, ]
  test <- GasolineYield[folds == fold, ]

  fit_train <- gkwreg(yield ~ batch + temp, train,
    family = "kw"
  )
  pred_test <- predict(fit_train, newdata = test, type = "response")

  sqrt(mean((test$yield - pred_test)^2))
})

cat("Cross-validated RMSE:", mean(cv_rmse), "\n")
```

---

gkw_control *Control Parameters for Generalized Kumaraswamy Regression*

Description

Auxiliary function for controlling gkwreg() fitting process. This function consolidates all technical/advanced fitting options in one place, keeping the main gkwreg() interface clean and user-friendly. Follows the same design pattern as glm.control, betareg.control, and similar control functions in R.

Usage

```
gkw_control(
  method = c("nlminb", "BFGS", "Nelder-Mead", "CG", "SANN", "L-BFGS-B"),
  start = NULL,
  fixed = NULL,
  hessian = TRUE,
  maxit = 500,
  reltol = sqrt(.Machine$double.eps),
  abstol = 0,
  trace = 0,
  silent = TRUE,
  eval.max = 500,
  iter.max = 300,
  step.min = 1e-08,
  step.max = 1,
  x.tol = 1.5e-08,
  rel.tol = sqrt(.Machine$double.eps),
  alpha = 1,
  beta = 0.5,
  gamma = 2,
  warn.1d.NelderMead = TRUE,
  type = 1,
  temp = 10,
  tmax = 10,
  lmm = 5,
  factr = 1e+07,
  pgtol = 0,
  REPORT = NULL,
  fnscale = 1,
  parscale = NULL,
  ndeps = NULL,
  ...
)

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

Arguments

method	Character string specifying the optimization algorithm. Options: "nlminb" (default), "BFGS", "Nelder-Mead", "CG", "SANN", "L-BFGS-B". If "nlminb", uses

nlminb; otherwise uses optim with the specified method.

start — Optional named list of starting values for regression coefficients. Names should match parameter names (alpha, beta, gamma, delta, lambda). If NULL (default), starting values are determined automatically.

fixed — Optional named list of parameters to hold fixed at specific values during estimation. Currently experimental. Default NULL.

hessian — Logical. If TRUE (default), compute the Hessian matrix via sdreport to obtain standard errors and variance-covariance matrix. Set to FALSE for faster fitting when standard errors are not needed.

maxit — Integer. Maximum number of iterations for the optimizer. Default 500 for derivative-based methods, 10000 for SANN. Increase for difficult optimization problems.

reltol — Numeric. Relative convergence tolerance for the optimizer. Default sqrt(.Machine$double.eps) approx. 1.5e-8. Smaller values require tighter convergence but may increase computation time. Used by Nelder-Mead, BFGS, and CG methods.

abstol — Numeric. Absolute convergence tolerance. Default 0. Used by some optimization methods as an additional stopping criterion.

trace — Integer. Controls verbosity of the optimizer.

- 0: Silent (default)
- 1: Print iteration progress
- 2+: Print detailed diagnostic information (up to 6 for L-BFGS-B)

Ignored if silent = TRUE.

silent — Logical. If TRUE (default), suppress all progress messages from TMB compilation and optimization. Set to FALSE for debugging or to monitor long-running fits.

eval.max — Integer. Maximum number of function evaluations (nlminb only). Default 500. Increase for difficult optimization problems.

iter.max — Integer. Maximum number of iterations (nlminb only). Default 300. Usually less than eval.max.

step.min — Numeric. Minimum step length (nlminb only). Default 1e-8. Controls how small steps can become before stopping.

step.max — Numeric. Maximum step length (nlminb only). Default 1. Useful for preventing overshooting in difficult optimization problems.

x.tol — Numeric. Tolerance for parameter convergence (nlminb only). Default 1.5e-8. Optimizer stops if parameter changes are smaller than this.

rel.tol — Numeric. Relative tolerance for function value (nlminb only). Default sqrt(.Machine$double.eps). Alternative specification of relative tolerance.

alpha — Numeric. Reflection factor for Nelder-Mead method. Default 1.0. Only used when method = "Nelder-Mead".

beta — Numeric. Contraction factor for Nelder-Mead method. Default 0.5. Only used when method = "Nelder-Mead".

gamma — Numeric. Expansion factor for Nelder-Mead method. Default 2.0. Only used when method = "Nelder-Mead".

`warn.1d.NelderMead`
: Logical. Whether to warn when Nelder-Mead is used for one-dimensional optimization. Default `TRUE`.

`type`
: Integer. Update formula for CG method. Options:

  - 1: Fletcher-Reeves update
  - 2: Polak-Ribiere update
  - 3: Beale-Sorenson update

  Default 1. Only used when `method = "CG"`.

`temp`
: Numeric. Starting temperature for SANN method. Default 10. Only used when `method = "SANN"`.

`tmax`
: Integer. Number of function evaluations at each temperature for SANN method. Default 10. Only used when `method = "SANN"`.

`lmm`
: Integer. Number of BFGS updates retained in L-BFGS-B method. Default 5. Only used when `method = "L-BFGS-B"`.

`factr`
: Numeric. Convergence tolerance factor for L-BFGS-B method. Convergence occurs when the reduction in the objective is within this factor of the machine tolerance. Default 1e7 (tolerance ~1e-8). Only used when `method = "L-BFGS-B"`.

`pgtol`
: Numeric. Tolerance on the projected gradient for L-BFGS-B method. Default 0 (check suppressed). Only used when `method = "L-BFGS-B"`.

`REPORT`
: Integer. Frequency of progress reports for BFGS, L-BFGS-B and SANN methods when `trace > 0`. Default 10 for BFGS/L-BFGS-B, 100 for SANN.

`fnscale`
: Numeric. Overall scaling to be applied to the function value and gradient during optimization. Default 1. If negative, turns the problem into a maximization problem.

`parscale`
: Numeric vector. Scaling values for parameters. Optimization is performed on par/parscale. Default `rep(1, n_params)`.

`ndeps`
: Numeric vector. Step sizes for finite-difference approximation to the gradient. Default 1e-3.

`...`
: Additional arguments passed to the optimizer. Allows fine-grained control without formally adding parameters. Advanced users only.

`x`
: An object of class `"gkw_control"`.

Details

This function provides a centralized way to set all technical parameters for model fitting. It serves several purposes:

- **Clean interface**: `gkwreg()` has fewer arguments
- **Organized documentation**: All technical options documented here
- **Input validation**: Parameters validated before fitting
- **Extensibility**: New options can be added without changing `gkwreg()`
- **Backward compatibility**: Old code continues working

Method-specific parameters:

Each optimization method accepts different control parameters:

- **Nelder-Mead**: `alpha`, `beta`, `gamma`, `maxit`, `reltol`, `abstol`, `trace`, `REPORT`, `warn.1d.NelderMead`
- **BFGS**: `maxit`, `reltol`, `abstol`, `trace`, `REPORT`
- **CG**: `type`, `maxit`, `reltol`, `abstol`, `trace`
- **SANN**: `temp`, `tmax`, `maxit`, `trace`, `REPORT`
- **L-BFGS-B**: `lmm`, `factr`, `pgtol`, `trace`, `REPORT`

When to use gkw_control():

Most users never need to adjust these settings. Use `gkw_control()` when:

- Optimization fails to converge (increase `maxit`, adjust tolerances)
- Debugging fit problems (set `silent = FALSE`, `trace = 1`)
- Comparing optimizers (try `method = "BFGS"` vs `"nlminb"`)
- Fine-tuning performance (disable `hessian` if SEs not needed)
- Using custom starting values (`start = list(...)`)

Recommended practices:

- Start with defaults, only adjust if needed
- Increase `maxit` before adjusting tolerances
- Use `trace = 1` to diagnose convergence issues
- Disable `hessian` for speed if only point estimates needed
- Try different `methods` if one fails (BFGS often more robust)
- For L-BFGS-B with bounds, adjust `factr` and `pgtol` if needed

Value

An object of class `"gkw_control"`, which is a list containing all control parameters with validated and default-filled values. This object is passed to `gkwreg()` via the `control` argument.

Author(s)

Lopes, J. E.

References

Nocedal, J., & Wright, S. J. (2006). *Numerical Optimization* (2nd ed.). Springer.

Belisle, C. J. P. (1992). Convergence theorems for a class of simulated annealing algorithms on R^d. *Journal of Applied Probability*, 29, 885-895.

Byrd, R. H., Lu, P., Nocedal, J. and Zhu, C. (1995). A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16, 1190-1208.

See Also

gkwreg for the main fitting function, nlminb, optim for optimizer details, betareg.control for similar design pattern.

Examples

```
# Default control (used automatically if not specified)
ctrl <- gkw_control()
print(ctrl)

# Increase iterations for difficult problem
ctrl_robust <- gkw_control(maxit = 1000, trace = 1)

# Try alternative optimizer
ctrl_bfgs <- gkw_control(method = "BFGS")

# Fast fitting without standard errors
ctrl_fast <- gkw_control(hessian = FALSE)

# Verbose debugging
ctrl_debug <- gkw_control(silent = FALSE, trace = 2)

# Custom starting values
ctrl_start <- gkw_control(
  start = list(
    alpha = c(0.5, 0.2),
    beta = c(1.0, -0.3)
  )
)

# Configure Nelder-Mead with custom reflection/contraction
ctrl_nm <- gkw_control(
  method = "Nelder-Mead",
  alpha = 1.5,
  beta = 0.75
)

# Configure L-BFGS-B for bounded optimization
ctrl_lbfgsb <- gkw_control(
  method = "L-BFGS-B",
  factr = 1e6,
  lmm = 10
)

# Configure SANN for rough surfaces
ctrl_sann <- gkw_control(
  method = "SANN",
  temp = 20,
  tmax = 20,
  maxit = 20000
)
```

`ImpreciseTask` *Imprecise Probabilities for Sunday Weather and Boeing Stock Task*

Description

Data from a cognitive psychology experiment where participants estimated upper and lower probabilities for events to occur and not to occur. The study examines judgment under uncertainty with imprecise probability assessments.

Usage

```
ImpreciseTask
```

Format

A data frame with 242 observations on 3 variables:

task factor with levels `Boeing stock` and `Sunday weather`. Indicates which task the participant performed.

location numeric. Average of the lower estimate for the event not to occur and the upper estimate for the event to occur (proportion).

difference numeric. Difference between upper and lower probability estimates, measuring imprecision or uncertainty.

Details

All participants in the study were either first- or second-year undergraduate students in psychology at Australian universities, none of whom had a strong background in probability theory or were familiar with imprecise probability theories.

For the Sunday weather task, participants were asked to estimate the probability that the temperature at Canberra airport on Sunday would be higher than a specified value.

For the Boeing stock task, participants were asked to estimate the probability that Boeing's stock would rise more than those in a list of 30 companies.

For each task, participants were asked to provide lower and upper estimates for the event to occur and not to occur.

Source

Taken from Smithson et al. (2011) supplements.

References

Smithson, M., Merkle, E.C., and Verkuilen, J. (2011). Beta Regression Finite Mixture Models of Polarization and Priming. *Journal of Educational and Behavioral Statistics*, **36**(6), 804–831. doi:10.3102/1076998610396893

Smithson, M., and Segale, C. (2009). Partition Priming in Judgments of Imprecise Probabilities. *Journal of Statistical Theory and Practice*, **3**(1), 169–181.

Examples

```
require(gkwreg)
require(gkwdist)

data(ImpreciseTask)

# Example 1: Basic model with task effects
# Probability location varies by task type and uncertainty level
fit_kw <- gkwreg(location ~ task * difference,
  data = ImpreciseTask,
  family = "kw"
)
summary(fit_kw)

# Interpretation:
# - Alpha: Task type and uncertainty (difference) interact to affect
#   probability estimates
# - Different tasks may have different baseline probability assessments

# Example 2: Heteroscedastic model
# Precision of estimates may vary by task and uncertainty
fit_kw_hetero <- gkwreg(
  location ~ task * difference |
    task + difference,
  data = ImpreciseTask,
  family = "kw"
)
summary(fit_kw_hetero)

# Interpretation:
# - Beta: Variability in estimates differs between tasks
#   Higher uncertainty (difference) may lead to less precise estimates

# Example 3: McDonald distribution for extreme uncertainty
# Some participants may show very extreme probability assessments
fit_mc <- gkwreg(
  location ~ task * difference | # gamma: full interaction
    task * difference | # delta: full interaction
    task, # lambda: task affects extremity
  data = ImpreciseTask,
  family = "mc",
  control = gkw_control(
    method = "BFGS",
```

```
    maxit = 1500
  )
)
summary(fit_mc)

# Interpretation:
# - Lambda varies by task: Weather vs. stock may produce
#   different patterns of extreme probability assessments
```

`logLik.gkwreg` *Extract Log-Likelihood from Generalized Kumaraswamy Regression Models*

Description

Extracts the log-likelihood value from a fitted Generalized Kumaraswamy (GKw) regression model object.

Usage

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

Arguments

`object`	An object of class `"gkwreg"`, typically obtained from `gkwreg`.
`...`	Currently not used.

Details

The log-likelihood is extracted from the fitted model object and returned as an object of class `"logLik"` with appropriate attributes for the number of parameters (`df`) and observations (`nobs`). These attributes are required for information criteria calculations.

For a GKw regression model with parameter vector θ, the log-likelihood is defined as:

$$\ell(\theta \mid y) = \sum_{i=1}^{n} \log f(y_i; \alpha_i, \beta_i, \gamma_i, \delta_i, \lambda_i)$$

where $f(\cdot)$ is the probability density function of the specified GKw family distribution, and the parameters may depend on covariates through link functions.

Value

An object of class `"logLik"` containing the log-likelihood value with the following attributes:

`df` Number of estimated parameters

`nobs` Number of observations

Author(s)

Lopes, J. E.

See Also

gkwreg, AIC.gkwreg, BIC.gkwreg

Examples

```
# Load example data
data(GasolineYield)

# Fit a Kumaraswamy regression model
fit <- gkwreg(yield ~ batch + temp, data = GasolineYield, family = "kw")

# Extract log-likelihood
ll <- logLik(fit)
print(ll)

# Access attributes
cat("Log-likelihood:", as.numeric(ll), "\n")
cat("Parameters:", attr(ll, "df"), "\n")
cat("Observations:", attr(ll, "nobs"), "\n")
```

LossAversion *(No) Myopic Loss Aversion in Adolescents*

Description

Data from a behavioral economics experiment assessing the extent of myopic loss aversion among adolescents aged 11 to 19 years. The experiment tests whether short-term investment horizons lead to more conservative investment behavior.

Usage

```
LossAversion
```

Format

A data frame with 570 observations on 7 variables:

invest numeric. Average proportion of tokens invested across all 9 rounds of the experiment (response variable).

gender factor. Gender of the player (or team of players).

male factor. Was (at least one of) the player(s) male (in the team)?

age numeric. Age of the player (or average age in case of team).

grade factor. School grade of the player(s).

arrangement factor. Investment horizon treatment with levels `short` (1 round), `medium` (3 rounds), and `long` (9 rounds).

treatment factor. Type of treatment: long vs. short.

Details

The data were collected by Matthias Sutter and Daniela Glätzle-Rützler (Universität Innsbruck) in an experiment with high-school students in Tyrol, Austria (Schwaz and Innsbruck). The experiment tests the theory of myopic loss aversion, which proposes that investors with shorter evaluation periods are more loss-averse and thus invest less in risky assets.

Classical theory predicts that players with short investment horizons (myopic view) should invest less due to loss aversion. However, Sutter et al. (2015) found no evidence of myopic loss aversion in adolescents, contrary to findings in adult populations.

The investment game structure: In each round, players could invest tokens in a risky asset with 50% chance of doubling or losing the investment. The treatment varied the feedback frequency (short = every round, medium = every 3 rounds, long = only at the end).

Source

Data collected by Matthias Sutter and Daniela Glätzle-Rützler, Universität Innsbruck.

References

Sutter, M., Kocher, M.G., Glätzle-Rützler, D., and Trautmann, S.T. (2015). No Myopic Loss Aversion in Adolescents? – An Experimental Note. *Journal of Economic Behavior & Organization*, **111**, 169–176. doi:10.1016/j.jebo.2014.12.021

Kosmidis, I., and Zeileis, A. (2024). Extended-Support Beta Regression for (0, 1) Responses. *arXiv:2409.07233*. doi:10.48550/arXiv.2409.07233

Examples

```
require(gkwreg)
require(gkwdist)

data(LossAversion)
# Control bounds

LossAversion$invest <- with(
  LossAversion,
  ifelse(invest <= 0, 0.000001,
    ifelse(invest >= 1, 0.999999, invest)
  )
)
# Example 1: Test for myopic loss aversion
# Do short-term players invest less? (They shouldn't, per Sutter et al.)
fit_kw <- gkwreg(
  invest ~ arrangement + age + male + grade |
    arrangement + male,
```

```
  data = LossAversion,
  family = "kw"
)
summary(fit_kw)

# Interpretation:
# - Alpha: Effect of investment horizon (arrangement) on mean investment
#   Age and gender effects on risk-taking
# - Beta: Precision varies by horizon and gender
#   (some groups more consistent than others)

# Example 2: Interaction effects
# Does the horizon effect differ by age/grade?
fit_kw_interact <- gkwreg(
  invest ~ grade * (arrangement + age) + male |
    arrangement + male + grade,
  data = LossAversion,
  family = "kw"
)
summary(fit_kw_interact)

# Interpretation:
# - Grade × arrangement interaction tests if myopic loss aversion
#   emerges differently at different developmental stages

# Example 3: Extended-support for boundary observations
# Some students invest 0% or 100% of tokens
# Original 'invest' variable may include exact 0 and 1 values
fit_xbx <- gkwreg(
  invest ~ grade * (arrangement + age) + male |
    arrangement + male + grade,
  data = LossAversion,
  family = "kw" # Note: for true [0,1] support, use extended-support models
)
summary(fit_xbx)

# Interpretation:
# - Model accommodates extreme risk-taking (all-in or all-out strategies)

# Compare models
anova(fit_kw, fit_kw_interact)

# Visualization: Investment by horizon
boxplot(invest ~ arrangement,
  data = LossAversion,
  xlab = "Investment Horizon", ylab = "Proportion Invested",
  main = "No Myopic Loss Aversion in Adolescents",
  col = c("lightblue", "lightgreen", "lightyellow")
)
```

`lrtest` *Likelihood Ratio Test for Nested GKw Models*

Description

Performs a likelihood ratio test to compare two nested Generalized Kumaraswamy regression models.

Usage

```
lrtest(object, object2)
```

Arguments

`object`	A fitted model object of class `"gkwreg"` (the restricted model).
`object2`	A fitted model object of class `"gkwreg"` (the full model).

Details

This function performs a likelihood ratio test (LRT) to compare two nested models. The test statistic is:

$$LRT = 2(\ell_{\text{full}} - \ell_{\text{restricted}})$$

which follows a chi-squared distribution with degrees of freedom equal to the difference in the number of parameters.

The models must be nested (one is a special case of the other) and fitted to the same data for the test to be valid.

Value

A list with class `"htest"` containing:

`statistic` The LRT test statistic

`parameter` Degrees of freedom for the test

`p.value` P-value from the chi-squared distribution

`method` Description of the test

`data.name` Names of the compared models

Author(s)

Lopes, J. E.

See Also

`anova.gkwreg`

Examples

```
data(GasolineYield)

# Fit nested models
fit_restricted <- gkwreg(yield ~ temp, data = GasolineYield, family = "kw")
fit_full <- gkwreg(yield ~ batch + temp, data = GasolineYield, family = "kw")

# Likelihood ratio test
lrtest(fit_restricted, fit_full)
```

MockJurors *Confidence of Mock Jurors in Their Verdicts*

Description

Data from a study examining factors that influence mock juror confidence in verdicts for criminal trials. The experiment manipulates verdict options (two-option vs. three-option) and presence of conflicting testimonial evidence.

Usage

```
MockJurors
```

Format

A data frame with 104 observations on 3 variables:

confidence numeric. Juror confidence in their verdict, scaled to the open unit interval (0, 1). Original scale was 0-100.

verdict factor indicating whether a two-option verdict (guilty vs. acquittal) or three-option verdict (with Scottish 'not proven' alternative) was requested. Sum contrast coding is employed.

conflict factor. Is there conflicting testimonial evidence? Values are `no` or `yes`. Sum contrast coding is employed.

Details

The data were collected by Deady (2004) among first-year psychology students at Australian National University. The experiment examined how the availability of a third verdict option ('not proven') and conflicting evidence affect juror confidence.

Smithson and Verkuilen (2006) employed the data, scaling the original confidence (on a scale 0-100) to the open unit interval using the transformation: `((original_confidence/100) * 103 - 0.5) / 104`.

Important note: The original coding of `conflict` in the data provided from Smithson's homepage is -1/1 which Smithson and Verkuilen (2006) describe to mean no/yes. However, all their results (sample statistics, histograms, etc.) suggest that it actually means yes/no, which was employed in the corrected `MockJurors` dataset.

Source

Data collected by Deady (2004), analyzed by Smithson and Verkuilen (2006).

References

Deady, S. (2004). *The Psychological Third Verdict: 'Not Proven' or 'Not Willing to Make a Decision'?* Unpublished honors thesis, The Australian National University, Canberra.

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

Examples

```
require(gkwreg)
require(gkwdist)

data(MockJurors)

# Example 1: Main effects model with heteroscedasticity
# Confidence depends on verdict options and conflicting evidence
# Variability may also depend on these factors
fit_kw <- gkwreg(
  confidence ~ verdict + conflict |
    verdict * conflict,
  data = MockJurors,
  family = "kw"
)
summary(fit_kw)

# Interpretation:
# - Alpha (mean): Additive effects of verdict type and conflict
#   Three-option verdicts may reduce confidence
#   Conflicting evidence reduces confidence
# - Beta (precision): Interaction suggests confidence variability
#   depends on combination of verdict options and evidence type

# Example 2: Full interaction in mean model
fit_kw_interact <- gkwreg(
  confidence ~ verdict * conflict |
    verdict * conflict,
  data = MockJurors,
  family = "kw"
)
summary(fit_kw_interact)

# Interpretation:
# - Full interaction: Third verdict option may have different effects
#   depending on whether evidence is conflicting

# Test interaction significance
anova(fit_kw, fit_kw_interact)
```

```
# Example 3: McDonald distribution for extreme confidence patterns
# Jurors may show very high confidence (ceiling effects) or very low
# confidence depending on conditions
fit_mc <- gkwreg(
  confidence ~ verdict * conflict | # gamma: full interaction
    verdict * conflict | # delta: full interaction
    verdict + conflict, # lambda: additive extremity effects
  data = MockJurors,
  family = "mc",
  control = gkw_control(
    method = "BFGS",
    maxit = 1500,
    reltol = 1e-8
  )
)
summary(fit_mc)

# Interpretation:
# - Lambda: Models asymmetry and extreme confidence
#   Some conditions produce more polarized confidence (very high or very low)

# Example 4: Exponentiated Kumaraswamy alternative
fit_ekw <- gkwreg(
  confidence ~ verdict * conflict | # alpha
    verdict + conflict | # beta
    conflict, # lambda: conflict affects extremity
  data = MockJurors,
  family = "ekw",
  control = gkw_control(
    method = "BFGS",
    maxit = 1500
  )
)
summary(fit_ekw)

# Compare 3-parameter models
AIC(fit_ekw, fit_mc)
```

`model.frame.gkwreg` *Extract Model Frame from GKw Regression Model*

Description

Extracts the model frame from a fitted Generalized Kumaraswamy regression model object.

Usage

```
## S3 method for class 'gkwreg'
model.frame(formula, ...)
```

Arguments

formula	An object of class "gkwreg".
...	Currently not used.

Value

A data frame containing the variables used in fitting the model.

Author(s)

Lopes, J. E.

See Also

gkwreg, model.matrix.gkwreg

Examples

```
data(GasolineYield)
fit <- gkwreg(yield ~ batch + temp, data = GasolineYield, family = "kw")
head(model.frame(fit))
```

---

model.matrix.gkwreg *Extract Model Matrix from GKw Regression Model*

Description

Extracts the model matrix (design matrix) from a fitted Generalized Kumaraswamy regression model object.

Usage

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

Arguments

object	An object of class "gkwreg".
...	Currently not used.

Value

A design matrix.

Author(s)

Lopes, J. E.

See Also

gkwreg, model.frame.gkwreg

Examples

```
data(GasolineYield)
fit <- gkwreg(yield ~ batch + temp, data = GasolineYield, family = "kw")
head(model.matrix(fit))
```

---

nobs.gkwreg *Number of Observations for GKw Regression Models*

Description

Extracts the number of observations from a fitted Generalized Kumaraswamy regression model.

Usage

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

Arguments

object	An object of class "gkwreg", typically obtained from gkwreg.
...	Currently not used.

Value

Integer representing the number of observations used in model fitting.

Author(s)

Lopes, J. E.

See Also

gkwreg

Examples

```
data(GasolineYield)
fit <- gkwreg(yield ~ batch + temp, data = GasolineYield, family = "kw")
nobs(fit)
```

`plot.gkwreg` *Diagnostic Plots for Generalized Kumaraswamy Regression Models*

Description

Produces a comprehensive set of diagnostic plots for assessing the adequacy of a fitted Generalized Kumaraswamy (GKw) regression model (objects of class `"gkwreg"`). The function offers flexible plot selection, multiple residual types, and support for both base R graphics and `ggplot2` with extensive customization options. Designed for thorough model evaluation including residual analysis, influence diagnostics, and goodness-of-fit assessment.

Usage

```
## S3 method for class 'gkwreg'
plot(
  x,
  which = 1:6,
  type = c("quantile", "pearson", "deviance"),
  family = NULL,
  caption = NULL,
  main = "",
  sub.caption = "",
  ask = NULL,
  use_ggplot = FALSE,
  arrange_plots = FALSE,
  nsim = 100,
  level = 0.9,
  sample_size = NULL,
  theme_fn = NULL,
  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` (all plots). The plots correspond to:

1. **Residuals vs. Observation Indices**: Checks for temporal patterns, trends, or autocorrelation in residuals across observation order.
2. **Cook's Distance Plot**: Identifies influential observations that have disproportionate impact on model estimates. Points exceeding the 4/n threshold warrant investigation.
3. **Generalized Leverage vs. Fitted Values**: Identifies high leverage points with unusual predictor combinations. Points exceeding 2p/n threshold may be influential.
4. **Residuals vs. Linear Predictor**: Checks for non-linearity in the predictor-response relationship and heteroscedasticity (non-constant variance).
5. **Half-Normal Plot with Simulated Envelope**: Assesses normality of residuals (particularly useful for quantile residuals) by comparing observed residuals against simulated quantiles. Points outside the envelope indicate potential model misspecification.
6. **Predicted vs. Observed Values**: Overall goodness-of-fit check showing model prediction accuracy and systematic bias.

`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. Most interpretable with standard diagnostic tools.
- `"pearson"`: Pearson residuals (response residual standardized by estimated standard deviation). Useful for checking the variance function and identifying heteroscedasticity patterns.
- `"deviance"`: Deviance residuals. Related to the log-likelihood contribution of each observation. Sum of squared deviance residuals equals the model deviance.

`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 across competing model specifications. Available options match those in `gkwreg`: `"gkw"`, `"bkw"`, `"kkw"`, `"ekw"`, `"mc"`, `"kw"`, `"beta"`.

`caption` Titles for the diagnostic plots. Can be specified in three ways:

- `NULL` (default): Uses standard default captions for all plots.
- **Character vector** (backward compatibility): A vector of 6 strings corresponding to plots 1-6. Must provide all 6 titles even if only customizing some.
- **Named list** (recommended): A list with plot numbers as names (e.g., `list("3" = "My Custom Title")`). Only specified plots are customized; others use defaults. This allows partial customization without repeating all titles.

Default captions are:

1. "Residuals vs. Observation Indices"
2. "Cook's Distance Plot"
3. "Generalized Leverage vs. Fitted Values"

4. "Residuals vs. Linear Predictor"
5. "Half-Normal Plot of Residuals"
6. "Predicted vs. Observed Values"

`main` Character string to be prepended to individual plot captions (from the `caption` argument). Useful for adding a common prefix to all plot titles. Defaults to `""` (no prefix).

`sub.caption` Character string used as a common subtitle positioned above all plots (especially when multiple plots are arranged). If `NULL` (default), automatically generates a subtitle from the model call (`deparse(x$call)`). Set to `""` to suppress the subtitle entirely.

`ask` Logical. If `TRUE` (and using base R graphics with multiple plots on an interactive device), the user is prompted before displaying each plot. If `NULL` (default), automatically determined: `TRUE` if more plots are requested than fit on the current screen layout and the session is interactive; `FALSE` otherwise. Explicitly set to `FALSE` to disable prompting or `TRUE` to force prompting.

`use_ggplot` Logical. If `TRUE`, plots are generated using the `ggplot2` package, providing modern, publication-quality graphics with extensive theming capabilities. If `FALSE` (default), uses base R graphics, which are faster and require no additional dependencies. 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 layout using either the `gridExtra` or `ggpubr` package (requires one of them to be installed). If `FALSE` (default), plots are displayed individually in sequence. Ignored when using base R graphics.

`nsim` Integer. Number of simulations used to generate the confidence envelope in the half-normal plot (`which = 5`). Higher values provide more accurate envelopes but increase computation time. Defaults to 100, which typically provides adequate precision. Must be a positive integer. Typical range: 50-500.

`level` Numeric. The confidence level (between 0 and 1) for the simulated envelope in the half-normal plot (`which = 5`). Defaults to 0.90 (90\ falling outside this envelope suggest potential model inadequacy or outliers.

`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 significantly speed up plot generation for very large datasets (n > 10,000) with minimal impact on diagnostic interpretation. Defaults to `NULL` (use all observations). Recommended values: 1000-5000 for large datasets.

`theme_fn` A function. Only relevant if `use_ggplot = TRUE`. Specifies a `ggplot2` theme function to apply to all plots for consistent styling (e.g., `ggplot2::theme_bw`, `ggplot2::theme_classic`, `ggplot2::theme_minimal`). If `NULL` (default), automatically uses `ggplot2::theme_minimal` when `use_ggplot = TRUE`. Can also be a custom theme function. Ignored when using base R graphics.

`save_diagnostics`

Logical. If `TRUE`, the function invisibly returns a list containing all calculated diagnostic measures (residuals, leverage, Cook's distance, fitted values, etc.) instead of the model object. Useful for programmatic access to diagnostic values

for custom analysis or reporting. If FALSE (default), the function invisibly returns the original model object x. The function is primarily called for its side effect of generating plots.

... Additional graphical parameters passed to the underlying plotting functions. For base R graphics, these are standard par() parameters such as col, pch, cex, lwd, etc. For ggplot2, these are typically ignored but can be used for specific geom customizations in advanced usage. Always specified last to follow R best practices.

Details

Diagnostic plots are essential for evaluating the assumptions and adequacy of fitted regression models. This function provides a comprehensive suite of standard diagnostic tools adapted specifically for gkwreg objects, which model bounded responses in the (0,1) interval.

Residual Types and Interpretation: The choice of residual type (type) is important and depends on the diagnostic goal:

- **Quantile Residuals** (type = "quantile"): **Recommended as default** for bounded response models. These residuals are constructed to be approximately N(0,1) under a correctly specified model, making standard diagnostic tools (QQ-plots, hypothesis tests) directly applicable. They are particularly effective for detecting model misspecification in the distributional family or systematic bias.
- **Pearson Residuals** (type = "pearson"): Standardized residuals that account for the mean-variance relationship. Useful for assessing whether the assumed variance function is appropriate. If plots show patterns or non-constant spread, this suggests the variance model may be misspecified.
- **Deviance Residuals** (type = "deviance"): Based on the contribution of each observation to the model deviance. Often have more symmetric distributions than Pearson residuals and are useful for identifying observations that fit poorly according to the likelihood criterion.

Individual Plot Interpretations:

Plot 1 - Residuals vs. Observation Indices:

- *Purpose*: Detect temporal patterns or autocorrelation
- *What to look for*: Random scatter around zero. Any systematic patterns (trends, cycles, clusters) suggest autocorrelation or omitted time-varying predictors.
- *Action*: If patterns are detected, consider adding time-related predictors or modeling autocorrelation structure.

Plot 2 - Cook's Distance:

- *Purpose*: Identify influential observations affecting coefficient estimates
- *What to look for*: Points exceeding the 4/n reference line have high influence. These observations, if removed, would substantially change model estimates.
- *Action*: Investigate high-influence points for data entry errors, outliers, or legitimately unusual cases. Consider sensitivity analysis.

Plot 3 - Leverage vs. Fitted Values:

- *Purpose*: Identify observations with unusual predictor combinations

- *What to look for*: Points exceeding the 2p/n reference line have high leverage. These are unusual in predictor space but may or may not be influential.
- *Action*: High leverage points deserve scrutiny but are only problematic if they also have large residuals (check Plots 1, 4).

Plot 4 - Residuals vs. Linear Predictor:

- *Purpose*: Detect non-linearity and heteroscedasticity
- *What to look for*: Random scatter around zero with constant spread. Curved patterns suggest non-linear relationships. Funnel shapes indicate heteroscedasticity (non-constant variance).
- *Action*: For non-linearity, add polynomial terms or use splines. For heteroscedasticity, consider alternative link functions or variance models.

Plot 5 - Half-Normal Plot with Envelope:

- *Purpose*: Assess overall distributional adequacy
- *What to look for*: Points should follow the reference line and stay within the simulated envelope. Systematic deviations indicate distributional misspecification. Isolated points outside the envelope suggest outliers.
- *Action*: If many points fall outside the envelope, try a different distributional family or check for outliers and data quality issues.

Plot 6 - Predicted vs. Observed:

- *Purpose*: Overall model fit and prediction accuracy
- *What to look for*: Points should cluster around the 45-degree line. Systematic deviations above or below indicate over- or under-prediction. Large scatter indicates poor predictive performance.
- *Action*: Poor fit suggests missing predictors, incorrect functional form, or inappropriate distributional family.

Using Caption Customization: The new **named list interface** for `caption` allows elegant partial customization:

```
# OLD WAY (still supported): Must repeat all 6 titles
plot(model, caption = c(
  "Residuals vs. Observation Indices",
  "Cook's Distance Plot",
  "MY CUSTOM TITLE FOR PLOT 3",  # Only want to change this
  "Residuals vs. Linear Predictor",
  "Half-Normal Plot of Residuals",
  "Predicted vs. Observed Values"
))

# NEW WAY: Specify only what changes
plot(model, caption = list(
  "3" = "MY CUSTOM TITLE FOR PLOT 3"
))
# Plots 1,2,4,5,6 automatically use defaults

# Customize multiple plots
plot(model, caption = list(
```

```
  "1" = "Time Series of Residuals",
  "5" = "Distributional Assessment"
))
```

The vector interface remains fully supported for backward compatibility.

NULL Defaults and Intelligent Behavior: Several arguments default to `NULL`, triggering intelligent automatic behavior:

- `sub.caption = NULL`: Automatically generates subtitle from model call
- `ask = NULL`: Automatically prompts only when needed (multiple plots on interactive device)
- `theme_fn = NULL`: Automatically uses `theme_minimal` when `use_ggplot = TRUE`

You can override these by explicitly setting values:

```
plot(model, sub.caption = "")           # Disable subtitle
plot(model, ask = FALSE)                # Never prompt
plot(model, theme_fn = theme_classic)   # Custom theme
```

Performance Considerations: For large datasets (n > 10,000):

- Use `sample_size` to work with a random subset (e.g., `sample_size = 2000`)
- Reduce `nsim` for half-normal plot (e.g., `nsim = 50`)
- Use base R graphics (`use_ggplot = FALSE`) for faster rendering
- Skip computationally intensive plots: `which = c(1,2,4,6)` (excludes half-normal plot)

Graphics Systems: Base R Graphics (`use_ggplot = FALSE`):

- Faster rendering, especially for large datasets
- No external dependencies beyond base R
- Traditional R look and feel
- Interactive `ask` prompting supported
- Customize via `...` parameters (standard `par()` settings)

ggplot2 Graphics (`use_ggplot = TRUE`):

- Modern, publication-quality aesthetics
- Consistent theming via `theme_fn`
- Grid arrangement support via `arrange_plots`
- Requires `ggplot2` package (and optionally `gridExtra` or `ggpubr` for arrangements)
- No interactive `ask` prompting (ggplot limitation)

Value

Invisibly returns either:

- The original fitted model object `x` (if `save_diagnostics = FALSE`, the default). This allows piping or chaining operations.
- A list containing diagnostic measures (if `save_diagnostics = TRUE`), including:
  - `data`: Data frame with observation indices, observed values, fitted values, residuals, Cook's distance, leverage, and linear predictors
  - `model_info`: List with model metadata (n, p, thresholds, family, type, etc.)

- `half_normal`: Data frame with half-normal plot data and envelope (if `which` includes 5)

The function is primarily called for its side effect of generating diagnostic plots. The invisible return allows:

```
# Silent plotting
plot(model)

# Or capture for further use
diag <- plot(model, save_diagnostics = TRUE)
head(diag$data)
```

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. doi:10.1080/10618600.1996.10474708

Cook, R. D. (1977). Detection of Influential Observation in Linear Regression. *Technometrics*, **19**(1), 15-18. doi:10.1080/00401706.1977.10489493

Atkinson, A. C. (1985). *Plots, Transformations and Regression*. Oxford University Press.

See Also

- `gkwreg` for fitting Generalized Kumaraswamy regression models
- `residuals.gkwreg` for extracting different types of residuals
- `fitted.gkwreg` for extracting fitted values
- `summary.gkwreg` for model summaries
- `plot.lm` for analogous diagnostics in linear models
- `ggplot` for ggplot2 graphics system
- `grid.arrange` for arranging ggplot2 plots

Examples

```
# EXAMPLE 1: Basic Usage with Default Settings

# Simulate data
library(gkwdist)

set.seed(123)
n <- 200
x1 <- runif(n, -2, 2)
x2 <- rnorm(n)

# True model parameters
alpha_true <- exp(0.7 + 0.3 * x1)
```

```
beta_true <- exp(1.2 - 0.2 * x2)

# Generate response
y <- rkw(n, alpha = alpha_true, beta = beta_true)
df <- data.frame(y = y, x1 = x1, x2 = x2)

# Fit model
model <- gkwreg(y ~ x1 | x2, data = df, family = "kw")

# Generate all diagnostic plots with defaults
par(mfrow = c(3, 2))
plot(model, ask = FALSE)

# EXAMPLE 2: Selective Plots with Custom Residual Type

# Focus on key diagnostic plots only
par(mfrow = c(3, 1))
plot(model,
  which = c(2, 4, 5), # Cook's distance, Resid vs LinPred, Half-normal
  type = "pearson"
) # Use Pearson residuals

# Check for influential points (plot 2) and non-linearity (plot 4)
par(mfrow = c(2, 1))
plot(model,
  which = c(2, 4),
  type = "deviance"
)

# EXAMPLE 3: Caption Customization - New Named List Interface

# Customize only specific plot titles (RECOMMENDED NEW WAY)
par(mfrow = c(3, 1))
plot(model,
  which = c(1, 4, 6),
  caption = list(
    "1" = "Time Pattern Check",
    "4" = "Linearity Assessment",
    "6" = "Predictive Accuracy"
  )
)

# Customize subtitle and main title
par(mfrow = c(2, 1))
plot(model,
  which = c(1, 5),
  main = "Model Diagnostics",
  sub.caption = "Kumaraswamy Regression - Training Data",
  caption = list("5" = "Normality Check with 95% Envelope")
)

# Suppress subtitle entirely
par(mfrow = c(3, 2))
```

```
plot(model, sub.caption = "")

# EXAMPLE 4: Backward Compatible Caption (Vector Interface)

# OLD WAY - still fully supported
par(mfrow = c(3, 2))
plot(model,
  which = 1:6,
  caption = c(
    "Residual Pattern Analysis",
    "Influence Diagnostics",
    "Leverage Assessment",
    "Linearity Check",
    "Distributional Fit",
    "Prediction Quality"
  )
)

# EXAMPLE 5: ggplot2 Graphics with Theming

# Modern publication-quality plots
plot(model,
  use_ggplot = TRUE,
  arrange_plots = TRUE
)

# With custom theme
plot(model,
  use_ggplot = TRUE,
  theme_fn = ggplot2::theme_bw,
  arrange_plots = TRUE
)

# With classic theme and custom colors (via ...)
plot(model,
  use_ggplot = TRUE,
  theme_fn = ggplot2::theme_classic,
  arrange_plots = TRUE
)

# EXAMPLE 6: Arranged Multi-Panel ggplot2 Display

# Requires gridExtra or ggpubr package
plot(model,
  which = 1:4,
  use_ggplot = TRUE,
  arrange_plots = TRUE, # Arrange in grid
  theme_fn = ggplot2::theme_minimal
)

# Focus plots in 2x2 grid
plot(model,
  which = c(2, 3, 4, 6),
```

```
  use_ggplot = TRUE,
  arrange_plots = TRUE,
  caption = list(
    "2" = "Influential Cases",
    "3" = "High Leverage Points"
  )
)

# EXAMPLE 7: Half-Normal Plot Customization

# Higher precision envelope (more simulations)
par(mfrow = c(1, 2))
plot(model,
  which = 5,
  nsim = 500, # More accurate envelope
  level = 0.95
) # 95% confidence level

# Quick envelope for large datasets
plot(model,
  which = 5,
  nsim = 500, # Faster computation
  level = 0.90
)

# EXAMPLE 8: Different Residual Types Comparison

# Compare different residual types
par(mfrow = c(2, 2))
plot(model, which = 4, type = "quantile", main = "Quantile")
plot(model, which = 4, type = "pearson", main = "Pearson")
plot(model, which = 4, type = "deviance", main = "Deviance")
par(mfrow = c(1, 1))

# Quantile residuals for half-normal plot (recommended)
plot(model, which = 5, type = "quantile")

# EXAMPLE 9: Family Comparison Diagnostics

# Compare diagnostics under different distributional assumptions
# Helps assess if alternative family would fit better
par(mfrow = c(2, 2))
plot(model,
  which = c(5, 6),
  family = "kw", # Original family
  main = "Kumaraswamy"
)

plot(model,
  which = c(5, 6),
  family = "beta", # Alternative family
  main = "Beta"
)
```

```
par(mfrow = c(1, 1))

# EXAMPLE 10: Large Dataset - Performance Optimization

# Simulate large dataset
set.seed(456)
n_large <- 50000
x1_large <- runif(n_large, -2, 2)
x2_large <- rnorm(n_large)
alpha_large <- exp(0.5 + 0.2 * x1_large)
beta_large <- exp(1.0 - 0.1 * x2_large)
y_large <- rkw(n_large, alpha = alpha_large, beta = beta_large)
df_large <- data.frame(y = y_large, x1 = x1_large, x2 = x2_large)

model_large <- gkwreg(y ~ x1 | x2, data = df_large, family = "kw")

# Optimized plotting for large dataset
par(mfrow = c(2, 2), mar = c(3, 3, 2, 2))
plot(model_large,
  which = c(1, 2, 4, 6), # Skip computationally intensive plot 5
  sample_size = 2000, # Use random sample of 2000 observations
  ask = FALSE
) # Don't prompt

# If half-normal plot needed, reduce simulations
par(mfrow = c(1, 1))
plot(model_large,
  which = 5,
  sample_size = 1000, # Smaller sample
  nsim = 50
) # Fewer simulations

# EXAMPLE 11: Saving Diagnostic Data for Custom Analysis

# Extract diagnostic measures without plotting
par(mfrow = c(1, 1))
diag_data <- plot(model_large,
  which = 1:6,
  save_diagnostics = TRUE
)

# Examine structure
str(diag_data)

# Access diagnostic measures
head(diag_data$data) # Residuals, Cook's distance, leverage, etc.

# Identify influential observations
influential <- which(diag_data$data$cook_dist > diag_data$model_info$cook_threshold)
cat("Influential observations:", head(influential), "\n")

# High leverage points
high_lev <- which(diag_data$data$leverage > diag_data$model_info$leverage_threshold)
```

```
cat("High leverage points:", head(high_lev), "\n")

# Custom diagnostic plot using saved data
plot(diag_data$data$fitted, diag_data$data$resid,
  xlab = "Fitted Values", ylab = "Residuals",
  main = "Custom Diagnostic Plot",
  col = ifelse(diag_data$data$cook_dist >
    diag_data$model_info$cook_threshold, "red", "black"),
  pch = 16
)
abline(h = 0, col = "gray", lty = 2)
legend("topright", legend = "Influential", col = "red", pch = 16)

# EXAMPLE 12: Interactive Plotting Control

# ask = TRUE Force prompting between plots (useful for presentations)
# Disable prompting (batch processing)
par(mfrow = c(3, 2))
plot(model,
  which = 1:6,
  ask = FALSE
) # Never prompts

# EXAMPLE 13: Base R Graphics Customization via ...

# Customize point appearance
par(mfrow = c(2, 2))
plot(model,
  which = c(1, 4, 6),
  pch = 16, # Filled circles
  col = "steelblue", # Blue points
  cex = 0.8
) # Smaller points

# Multiple customizations
plot(model,
  which = 2,
  pch = 21, # Circles with border
  col = "black", # Border color
  bg = "lightblue", # Fill color
  cex = 1.2, # Larger points
  lwd = 2
) # Thicker lines

# EXAMPLE 14: Comparing Models

# Fit competing models
model_kw <- gkwreg(y ~ x1 | x2, data = df, family = "kw")
model_beta <- gkwreg(y ~ x1 | x2, data = df, family = "beta")

# Compare diagnostics side-by-side
par(mfrow = c(2, 2))
```

```
# Kumaraswamy model
plot(model_kw, which = 5, main = "Kumaraswamy - Half-Normal")
plot(model_kw, which = 6, main = "Kumaraswamy - Pred vs Obs")

# Beta model
plot(model_beta, which = 5, main = "Beta - Half-Normal")
plot(model_beta, which = 6, main = "Beta - Pred vs Obs")

par(mfrow = c(1, 1))
```

`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)
library(gkwdist)
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")
```

`print.anova.gkwreg` *Print Method for ANOVA of GKw Models*

Description

Print method for analysis of deviance tables produced by `anova.gkwreg`.

Usage

```
## S3 method for class 'anova.gkwreg'
print(
  x,
  digits = max(getOption("digits") - 2L, 3L),
  signif.stars = getOption("show.signif.stars", TRUE),
  dig.tst = digits,
  ...
)
```

Arguments

`x`	An object of class `"anova.gkwreg"` from `anova.gkwreg`.
`digits`	Minimum number of significant digits to print. Default is `max(getOption("digits") - 2, 3)`.
`signif.stars`	Logical; if `TRUE` (default), significance stars are printed alongside p-values. Can be controlled globally via `options(show.signif.stars = FALSE)`.
`dig.tst`	Number of digits for test statistics. Default is `digits`.
`...`	Additional arguments (currently ignored).

Value

The object `x`, invisibly.

Author(s)

Lopes, J. E.

See Also

`anova.gkwreg`

print.gkwreg *Print Method for Generalized Kumaraswamy Regression Models*

Description

Print method for objects of class "gkwreg". Provides a concise summary of the fitted model following the style of print.lm.

Usage

```
## S3 method for class 'gkwreg'
print(x, digits = max(3, getOption("digits") - 3), ...)
```

Arguments

x	An object of class "gkwreg", typically obtained from gkwreg.
digits	Minimum number of significant digits to print. Default is max(3, getOption("digits") - 3).
...	Additional arguments passed to or from other methods.

Details

The print method provides a concise overview of the fitted model, showing: the call, deviance residuals summary, coefficient estimates, link functions, and basic fit statistics. For more detailed output including standard errors and significance tests, use summary.gkwreg.

Value

The object x, invisibly.

Author(s)

Lopes, J. E.

See Also

gkwreg, summary.gkwreg

Examples

```
data(GasolineYield)
fit <- gkwreg(yield ~ batch + temp, data = GasolineYield, family = "kw")
print(fit)

# With more digits
print(fit, digits = 5)
```

`ReadingSkills` *Dyslexia and IQ Predicting Reading Accuracy*

Description

Data for assessing the contribution of non-verbal IQ to children's reading skills in dyslexic and non-dyslexic children. This is a classic dataset demonstrating beta regression with interaction effects and heteroscedasticity.

Usage

```
ReadingSkills
```

Format

A data frame with 44 observations on 4 variables:

accuracy numeric. Reading accuracy score scaled to the open unit interval (0, 1). Perfect scores of 1 were replaced with 0.99.

accuracy1 numeric. Unrestricted reading accuracy score in (0, 1), including boundary observations.

dyslexia factor. Is the child dyslexic? Levels: `no` (control group) and `yes` (dyslexic group). Sum contrast coding is employed.

iq numeric. Non-verbal intelligence quotient transformed to z-scores (mean = 0, SD = 1).

Details

The data were collected by Pammer and Kevan (2004) and employed by Smithson and Verkuilen (2006) in their seminal beta regression paper. The sample includes 19 dyslexic children and 25 controls recruited from primary schools in the Australian Capital Territory. Children's ages ranged from 8 years 5 months to 12 years 3 months.

Mean reading accuracy was 0.606 for dyslexic readers and 0.900 for controls. The study investigates whether dyslexia contributes to reading accuracy even when controlling for IQ (which is on average lower for dyslexics).

Transformation details: The original reading accuracy score was transformed by Smithson and Verkuilen (2006) to fit beta regression requirements:

1. First, the original accuracy was scaled using the minimal and maximal scores (a and b) that can be obtained in the test: `accuracy1 = (original - a)/(b - a)` (a and b values are not provided).
2. Subsequently, `accuracy` was obtained from `accuracy1` by replacing all observations with a value of 1 with 0.99 to fit the open interval (0, 1).

The data clearly show asymmetry and heteroscedasticity (especially in the control group), making beta regression more appropriate than standard linear regression.

Source

Data collected by Pammer and Kevan (2004).

References

Cribari-Neto, F., and Zeileis, A. (2010). Beta Regression in R. *Journal of Statistical Software*, **34**(2), 1–24. doi:10.18637/jss.v034.i02

Grün, B., Kosmidis, I., and Zeileis, A. (2012). Extended Beta Regression in R: Shaken, Stirred, Mixed, and Partitioned. *Journal of Statistical Software*, **48**(11), 1–25. doi:10.18637/jss.v048.i11

Kosmidis, I., and Zeileis, A. (2024). Extended-Support Beta Regression for (0, 1) Responses. *arXiv:2409.07233*. doi:10.48550/arXiv.2409.07233

Pammer, K., and Kevan, A. (2004). *The Contribution of Visual Sensitivity, Phonological Processing and Nonverbal IQ to Children's Reading*. Unpublished manuscript, The Australian National University, Canberra.

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

Examples

```
require(gkwreg)
require(gkwdist)

data(ReadingSkills)

# Example 1: Standard Kumaraswamy with interaction and heteroscedasticity
# Mean: Dyslexia × IQ interaction (do groups differ in IQ effect?)
# Precision: Main effects (variability differs by group and IQ level)
fit_kw <- gkwreg(
  accuracy ~ dyslexia * iq |
    dyslexia + iq,
  data = ReadingSkills,
  family = "kw",
  control = gkw_control(method = "L-BFGS-B", maxit = 2000)
)
summary(fit_kw)

# Interpretation:
# - Alpha (mean): Interaction shows dyslexic children benefit less from
#   higher IQ compared to controls
# - Beta (precision): Controls show more variable accuracy (higher precision)
#   IQ increases consistency of performance

# Example 2: Simpler model without interaction
fit_kw_simple <- gkwreg(
  accuracy ~ dyslexia + iq |
    dyslexia + iq,
  data = ReadingSkills,
  family = "kw",
  control = gkw_control(method = "L-BFGS-B", maxit = 2000)
)
```

```
# Test if interaction is significant
anova(fit_kw_simple, fit_kw)

# Example 3: Exponentiated Kumaraswamy for ceiling effects
# Reading accuracy often shows ceiling effects (many perfect/near-perfect scores)
# Lambda parameter can model this right-skewed asymmetry
fit_ekw <- gkwreg(
  accuracy ~ dyslexia * iq | # alpha
    dyslexia + iq | # beta
    dyslexia, # lambda: ceiling effect by group
  data = ReadingSkills,
  family = "ekw",
  control = gkw_control(method = "L-BFGS-B", maxit = 2000)
)
summary(fit_ekw)

# Interpretation:
# - Lambda varies by dyslexia status: Controls have stronger ceiling effect
#   (more compression at high accuracy) than dyslexic children

# Test if ceiling effect modeling improves fit
anova(fit_kw, fit_ekw)

# Example 4: McDonald distribution alternative
# Provides different parameterization for extreme values
fit_mc <- gkwreg(
  accuracy ~ dyslexia * iq | # gamma
    dyslexia + iq | # delta
    dyslexia * iq, # lambda: interaction affects tails
  data = ReadingSkills,
  family = "mc",
  control = gkw_control(method = "L-BFGS-B", maxit = 2000)
)
summary(fit_mc)

# Compare 3-parameter models
AIC(fit_ekw, fit_mc)
```

`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

```
require(gkwreg)
require(gkwdist)

# Example 1: Comprehensive residual analysis for FoodExpenditure
data(FoodExpenditure)
FoodExpenditure$prop <- FoodExpenditure$food / FoodExpenditure$income

fit_kw <- gkwreg(
  prop ~ income + persons | income + persons,
  data = FoodExpenditure,
  family = "kw"
)

# Extract different types of residuals
res_response <- residuals(fit_kw, type = "response")
res_pearson <- residuals(fit_kw, type = "pearson")
res_deviance <- residuals(fit_kw, type = "deviance")
res_quantile <- residuals(fit_kw, type = "quantile")
res_coxsnell <- residuals(fit_kw, type = "cox-snell")

# Summary statistics
residual_summary <- data.frame(
  Type = c("Response", "Pearson", "Deviance", "Quantile", "Cox-Snell"),
  Mean = c(
    mean(res_response), mean(res_pearson),
    mean(res_deviance), mean(res_quantile),
    mean(res_coxsnell)
  ),
  SD = c(
    sd(res_response), sd(res_pearson),
    sd(res_deviance), sd(res_quantile),
    sd(res_coxsnell)
  ),
  Min = c(
    min(res_response), min(res_pearson),
    min(res_deviance), min(res_quantile),
    min(res_coxsnell)
  ),
  Max = c(
    max(res_response), max(res_pearson),
    max(res_deviance), max(res_quantile),
    max(res_coxsnell)
  )
)
print(residual_summary)

# Example 2: Diagnostic plots for model assessment
```

```
data(GasolineYield)

fit_ekw <- gkwreg(
  yield ~ batch + temp | temp | batch,
  data = GasolineYield,
  family = "ekw"
)

# Set up plotting grid
par(mfrow = c(2, 3))

# Plot 1: Residuals vs Fitted
fitted_vals <- fitted(fit_ekw)
res_pears <- residuals(fit_ekw, type = "pearson")
plot(fitted_vals, res_pears,
  xlab = "Fitted Values", ylab = "Pearson Residuals",
  main = "Residuals vs Fitted",
  pch = 19, col = rgb(0, 0, 1, 0.5)
)
abline(h = 0, col = "red", lwd = 2, lty = 2)
lines(lowess(fitted_vals, res_pears), col = "blue", lwd = 2)

# Plot 2: Normal QQ-plot (Quantile Residuals)
res_quant <- residuals(fit_ekw, type = "quantile")
qqnorm(res_quant,
  main = "Normal Q-Q Plot (Quantile Residuals)",
  pch = 19, col = rgb(0, 0, 1, 0.5)
)
qqline(res_quant, col = "red", lwd = 2)

# Plot 3: Scale-Location (sqrt of standardized residuals)
plot(fitted_vals, sqrt(abs(res_pears)),
  xlab = "Fitted Values", ylab = expression(sqrt("|Std. Residuals|")),
  main = "Scale-Location",
  pch = 19, col = rgb(0, 0, 1, 0.5)
)
lines(lowess(fitted_vals, sqrt(abs(res_pears))), col = "red", lwd = 2)

# Plot 4: Histogram of Quantile Residuals
hist(res_quant,
  breaks = 15, probability = TRUE,
  xlab = "Quantile Residuals",
  main = "Histogram with Normal Overlay",
  col = "lightblue", border = "white"
)
curve(dnorm(x, mean(res_quant), sd(res_quant)),
  add = TRUE, col = "red", lwd = 2
)

# Plot 5: Cox-Snell Residual Plot
res_cs <- residuals(fit_ekw, type = "cox-snell")
plot(qexp(ppoints(length(res_cs))), sort(res_cs),
  xlab = "Theoretical Exponential Quantiles",
```

```
  ylab = "Ordered Cox-Snell Residuals",
  main = "Cox-Snell Residual Plot",
  pch = 19, col = rgb(0, 0, 1, 0.5)
)
abline(0, 1, col = "red", lwd = 2)

# Plot 6: Residuals vs Index
plot(seq_along(res_pears), res_pears,
  xlab = "Observation Index", ylab = "Pearson Residuals",
  main = "Residuals vs Index",
  pch = 19, col = rgb(0, 0, 1, 0.5)
)
abline(h = 0, col = "red", lwd = 2, lty = 2)

par(mfrow = c(1, 1))

# Example 3: Partial residual plots for covariate effects
data(ReadingSkills)

fit_interact <- gkwreg(
  accuracy ~ dyslexia * iq | dyslexia + iq,
  data = ReadingSkills,
  family = "kw"
)

# Partial residuals for IQ effect on alpha parameter
X_alpha <- fit_interact$model_matrices$alpha
iq_col_alpha <- which(colnames(X_alpha) == "iq")

if (length(iq_col_alpha) > 0) {
  res_partial_alpha <- residuals(fit_interact,
    type = "partial",
    parameter = "alpha",
    covariate_idx = iq_col_alpha
  )

  par(mfrow = c(1, 2))

  # Partial residual plot for alpha
  plot(ReadingSkills$iq, res_partial_alpha,
    xlab = "IQ (z-scores)",
    ylab = "Partial Residual (alpha)",
    main = "Effect of IQ on Mean (alpha)",
    pch = 19, col = ReadingSkills$dyslexia
  )
  lines(lowess(ReadingSkills$iq, res_partial_alpha),
    col = "blue", lwd = 2
  )
  legend("topleft",
    legend = c("Control", "Dyslexic"),
    col = c("black", "red"), pch = 19
  )
```

```
  # Partial residuals for IQ effect on beta parameter
  X_beta <- fit_interact$model_matrices$beta
  iq_col_beta <- which(colnames(X_beta) == "iq")

  if (length(iq_col_beta) > 0) {
    res_partial_beta <- residuals(fit_interact,
      type = "partial",
      parameter = "beta",
      covariate_idx = iq_col_beta
    )

    plot(ReadingSkills$iq, res_partial_beta,
      xlab = "IQ (z-scores)",
      ylab = "Partial Residual (beta)",
      main = "Effect of IQ on Precision (beta)",
      pch = 19, col = ReadingSkills$dyslexia
    )
    lines(lowess(ReadingSkills$iq, res_partial_beta),
      col = "blue", lwd = 2
    )
  }

  par(mfrow = c(1, 1))
}

# Example 4: Comparing residuals across different families
data(StressAnxiety)

fit_kw_stress <- gkwreg(
  anxiety ~ stress | stress,
  data = StressAnxiety,
  family = "kw"
)

# Quantile residuals under different family assumptions
res_quant_kw <- residuals(fit_kw_stress, type = "quantile", family = "kw")
res_quant_beta <- residuals(fit_kw_stress, type = "quantile", family = "beta")

# Compare normality
par(mfrow = c(1, 2))

qqnorm(res_quant_kw,
  main = "QQ-Plot: Kumaraswamy Residuals",
  pch = 19, col = rgb(0, 0, 1, 0.5)
)
qqline(res_quant_kw, col = "red", lwd = 2)

qqnorm(res_quant_beta,
  main = "QQ-Plot: Beta Residuals",
  pch = 19, col = rgb(0, 0.5, 0, 0.5)
)
qqline(res_quant_beta, col = "red", lwd = 2)
```

```
par(mfrow = c(1, 1))

# Formal normality tests
shapiro_kw <- shapiro.test(res_quant_kw)
shapiro_beta <- shapiro.test(res_quant_beta)

cat("\nShapiro-Wilk Test Results:\n")
cat(
  "Kumaraswamy:  W =", round(shapiro_kw$statistic, 4),
  ", p-value =", round(shapiro_kw$p.value, 4), "\n"
)
cat(
  "Beta:         W =", round(shapiro_beta$statistic, 4),
  ", p-value =", round(shapiro_beta$p.value, 4), "\n"
)

# Example 5: Outlier detection using standardized residuals
data(MockJurors)

fit_mc <- gkwreg(
  confidence ~ verdict * conflict | verdict + conflict,
  data = MockJurors,
  family = "mc"
)

res_dev <- residuals(fit_mc, type = "deviance")
res_quant <- residuals(fit_mc, type = "quantile")

# Identify potential outliers (|z| > 2.5)
outlier_idx <- which(abs(res_quant) > 2.5)

if (length(outlier_idx) > 0) {
  cat("\nPotential outliers detected at indices:", outlier_idx, "\n")

  # Display outlier information
  outlier_data <- data.frame(
    Index = outlier_idx,
    Confidence = MockJurors$confidence[outlier_idx],
    Verdict = MockJurors$verdict[outlier_idx],
    Conflict = MockJurors$conflict[outlier_idx],
    Quantile_Residual = round(res_quant[outlier_idx], 3),
    Deviance_Residual = round(res_dev[outlier_idx], 3)
  )
  print(outlier_data)

  # Influence plot
  plot(seq_along(res_quant), res_quant,
    xlab = "Observation Index",
    ylab = "Quantile Residual",
    main = "Outlier Detection: Mock Jurors",
    pch = 19, col = rgb(0, 0, 1, 0.5)
  )
  points(outlier_idx, res_quant[outlier_idx],
```

```
    col = "red", pch = 19, cex = 1.5
  )
  abline(
    h = c(-2.5, 0, 2.5), col = c("orange", "black", "orange"),
    lty = c(2, 1, 2), lwd = 2
  )
  legend("topright",
    legend = c("Normal", "Outlier", "±2.5 SD"),
    col = c(rgb(0, 0, 1, 0.5), "red", "orange"),
    pch = c(19, 19, NA),
    lty = c(NA, NA, 2),
    lwd = c(NA, NA, 2)
  )
} else {
  cat("\nNo extreme outliers detected (|z| > 2.5)\n")
}
```

`response` *Extract Response Variable from GKw Regression Model*

Description

Extracts the response variable from a fitted Generalized Kumaraswamy regression model object.

Usage

```
response(object, ...)

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

Arguments

`object`	An object of class `"gkwreg"`.
`...`	Currently not used.

Value

A numeric vector containing the response variable values.

Author(s)

Lopes, J. E.

See Also

`gkwreg`, `fitted.gkwreg`, `residuals.gkwreg`

Examples

```
data(GasolineYield)
fit <- gkwreg(yield ~ batch + temp, data = GasolineYield, family = "kw")
y <- response(fit)
head(y)
```

`retinal` *Intraocular Gas Decay in Retinal Surgery*

Description

Longitudinal data on the recorded decay of intraocular gas (perfluoropropane) in complex retinal surgeries. The dataset tracks the proportion of gas remaining over time following vitrectomy procedures.

Usage

```
retinal
```

Format

A data frame with 40 observations on 7 variables:

ID integer. Patient identification number for longitudinal tracking.

Gas numeric. Proportion of intraocular gas remaining (0-1 scale). Response variable measuring the fraction of perfluoropropane gas still present in the vitreous cavity.

Time numeric. Time point of measurement (days or weeks post-surgery).

LogT numeric. Logarithm of time, log(Time). Used to linearize the exponential decay pattern.

LogT2 numeric. Squared logarithm of time, (log(Time))^2. Captures nonlinear decay patterns.

Level factor. Initial gas concentration level at the time of injection. Different starting concentrations affect decay kinetics.

Details

This longitudinal dataset comes from a study of gas decay following vitreoretinal surgery. Perfluoropropane (C3F8) is commonly used as a temporary tamponade agent in retinal detachment repair and other complex vitreoretinal procedures.

Clinical background: During vitrectomy for retinal detachment, gas bubbles are injected into the vitreous cavity to help reattach the retina by providing internal tamponade. The gas gradually absorbs and dissipates over time. Understanding the decay rate is important for:

- Predicting when patients can resume normal activities (esp. air travel)
- Assessing treatment efficacy
- Planning follow-up examinations

Decay kinetics: Gas decay typically follows a nonlinear pattern that can be approximated by exponential or power-law functions. The log transformation (LogT, LogT2) helps linearize these relationships for regression modeling.

Data structure: This is a longitudinal/panel dataset with repeated measurements on the same patients over time. Correlation structures (exchangeable, AR(1), etc.) should be considered when modeling.

The proportional nature of the gas variable (bounded between 0 and 1) makes this dataset ideal for:

- Simplex marginal models (original application by Song & Tan 2000)
- Beta regression with longitudinal correlation structures
- Kumaraswamy regression with heteroscedastic errors

Source

Based on clinical data from vitreoretinal surgery patients. Originally analyzed in Song and Tan (2000).

References

Meyers, S.M., Ambler, J.S., Tan, M., Werner, J.C., and Huang, S.S. (1992). Variation of Perfluoropropane Disappearance After Vitrectomy. *Retina*, **12**, 359–363.

Song, P.X.-K., and Tan, M. (2000). Marginal Models for Longitudinal Continuous Proportional Data. *Biometrics*, **56**, 496–502. doi:10.1111/j.0006341x.2000.00496.x

Song, P.X.-K., Qiu, Z., and Tan, M. (2004). Modelling Heterogeneous Dispersion in Marginal Models for Longitudinal Proportional Data. *Biometrical Journal*, **46**, 540–553.

Qiu, Z., Song, P.X.-K., and Tan, M. (2008). Simplex Mixed-Effects Models for Longitudinal Proportional Data. *Scandinavian Journal of Statistics*, **35**, 577–596. doi:10.1111/j.14679469.2008.00603.x

Zhang, P., Qiu, Z., and Shi, C. (2016). simplexreg: An R Package for Regression Analysis of Proportional Data Using the Simplex Distribution. *Journal of Statistical Software*, **71**(11), 1–21. doi:10.18637/jss.v071.i11

Examples

```
require(gkwreg)
require(gkwdist)

data(retinal)

# Example 1: Nonlinear time decay with level effects
# Model gas decay as quadratic function of log-time
# Allow precision to vary by initial gas concentration
fit_kw <- gkwreg(
  Gas ~ LogT + LogT2 + Level |
    Level,
  data = retinal,
  family = "kw"
)
summary(fit_kw)
```

```
# Interpretation:
# - Alpha: Decay curve shape varies by initial gas concentration
#   LogT + LogT2 capture nonlinear exponential-like decay
# - Beta: Precision differs by concentration level
#   Higher concentration may produce more/less variable decay

# Example 2: Heteroscedastic model
# Variability in gas proportion may change over time
fit_kw_hetero <- gkwreg(
  Gas ~ LogT + LogT2 + Level |
    Level + LogT,
  data = retinal,
  family = "kw"
)
summary(fit_kw_hetero)

# Interpretation:
# - Beta: Precision varies with both level and time
#   Early measurements may be more variable than late measurements

# Test heteroscedasticity
anova(fit_kw, fit_kw_hetero)

# Example 3: Exponentiated Kumaraswamy for decay tails
# Gas decay may show different tail behavior at extreme time points
# (very fast initial decay or very slow residual decay)
fit_ekw <- gkwreg(
  Gas ~ LogT + LogT2 + Level | # alpha: decay curve
    Level + LogT | # beta: heteroscedasticity
    Level, # lambda: tail heaviness by level
  data = retinal,
  family = "ekw"
)
summary(fit_ekw)

# Interpretation:
# - Lambda varies by level: Different initial concentrations may have
#   different rates of extreme decay (very fast or very slow residual gas)
# - Important for predicting complete absorption time

# Example 4: McDonald distribution for asymmetric decay
# Alternative parameterization for skewed decay patterns
fit_mc <- gkwreg(
  Gas ~ LogT + LogT2 + Level | # gamma
    LogT + Level | # delta
    Level, # lambda
  data = retinal,
  family = "mc",
  control = gkw_control(
    method = "BFGS",
    maxit = 1500,
    reltol = 1e-8
```

```
  )
)
summary(fit_mc)

# Model comparison
AIC(fit_kw, fit_kw_hetero, fit_ekw, fit_mc)
```

---

sdac *Autologous Peripheral Blood Stem Cell Transplants Data*

Description

Data on Autologous Peripheral Blood Stem Cell Transplants from the Stem Cell Lab in the Cross Cancer Institute, Alberta Health Services. The dataset examines recovery rates of CD34+ cells after peripheral blood stem cell (PBSC) transplants.

Usage

```
sdac
```

Format

A data frame with 60 observations on 5 variables:

rcd numeric. Recovery rate of CD34+ cells (proportion in (0, 1)). Response variable measuring the proportion of CD34+ cells recovered after PBSC transplant.

age numeric. Patient age in years (range: 18-71 years).

ageadj numeric. Age-adjusted covariate. Centered and scaled version of age for improved numerical stability in regression models.

chemo factor. Type of chemotherapy protocol used for stem cell mobilization. Levels include: `1-day`, `3-day`, `G-CSF only`, and `other`.

gender factor. Patient gender. Most patients in the study are male.

Details

This dataset contains clinical data from autologous peripheral blood stem cell (PBSC) transplant patients treated at the Cross Cancer Institute, Alberta Health Services. CD34+ cells are hematopoietic stem and progenitor cells critical for successful transplantation and hematopoietic recovery.

Clinical context: Autologous PBSC transplantation is used to treat various hematological malignancies including multiple myeloma, non-Hodgkin's lymphoma, acute leukemia, and some solid tumors. The recovery rate of CD34+ cells is a crucial predictor of engraftment success and patient outcomes.

Chemotherapy protocols:

- **1-day protocol:** Single-day high-dose chemotherapy for mobilization

- **3-day protocol:** Multi-day chemotherapy regimen
- **G-CSF only:** Granulocyte colony-stimulating factor without chemotherapy
- **Other:** Alternative or combined protocols

The proportion of recovered CD34+ cells naturally falls in the interval (0, 1), making it ideal for proportional data regression modeling. Age effects are particularly important as older patients may show different recovery patterns.

This dataset is particularly suitable for:

- Simplex regression (original application by Zhang et al. 2016)
- Beta regression with variable dispersion
- Kumaraswamy regression for flexible distributional modeling

Source

Stem Cell Lab, Cross Cancer Institute, Alberta Health Services, Canada.

References

Zhang, P., Qiu, Z., and Shi, C. (2016). simplexreg: An R Package for Regression Analysis of Proportional Data Using the Simplex Distribution. *Journal of Statistical Software*, **71**(11), 1–21. doi:10.18637/jss.v071.i11

Examples

```
require(gkwreg)
require(gkwdist)

data(sdac)

# Example 1: Basic Kumaraswamy regression
# Mean recovery depends on age and chemotherapy protocol
# Precision varies with age (older patients more variable)
fit_kw <- gkwreg(
  rcd ~ ageadj + chemo |
    age,
  data = sdac,
  family = "kw"
)
summary(fit_kw)

# Interpretation:
# - Alpha (mean recovery): Depends on age-adjusted covariate and chemo protocol
#   Different protocols show different baseline recovery rates
#   G-CSF-only may differ from multi-day chemotherapy protocols
# - Beta (precision): Raw age affects recovery variability
#   Hypothesis: Older patients show more heterogeneous responses

# Example 2: Include gender effects
# Gender may influence stem cell recovery rates
```

```
fit_kw_gender <- gkwreg(
  rcd ~ ageadj + chemo + gender |
    age + gender,
  data = sdac,
  family = "kw"
)
summary(fit_kw_gender)

# Interpretation:
# - Gender effects in both mean and precision
# - Precision may differ between males and females

# Test gender significance
anova(fit_kw, fit_kw_gender)

# Example 3: Exponentiated Kumaraswamy for extreme recovery patterns
# Some patients show unusually high or low recovery (outliers)
# Lambda parameter captures tail heaviness
fit_ekw <- gkwreg(
  rcd ~ ageadj + chemo + gender | # alpha: mean model
    age + chemo | # beta: precision varies with age and protocol
    chemo, # lambda: protocol affects extremity
  data = sdac,
  family = "ekw"
)
summary(fit_ekw)

# Clinical interpretation:
# - Lambda varies by chemotherapy protocol: Some protocols produce more
#   extreme recovery patterns (very high or very low CD34+ counts)
# - G-CSF-only vs multi-day protocols may differ in tail behavior
# - Important for risk stratification and clinical decision-making

# Test if extreme patterns differ by protocol
anova(fit_kw_gender, fit_ekw)

# Example 4: Interaction between age and protocol
# Protocol effectiveness may vary with patient age
fit_kw_interact <- gkwreg(
  rcd ~ ageadj * chemo |
    age * chemo,
  data = sdac,
  family = "kw"
)
summary(fit_kw_interact)

# Interpretation:
# - Interaction: Does protocol effectiveness decline with age?
# - Critical for personalized treatment selection
```

`StressAnxiety` *Dependency of Anxiety on Stress*

Description

Data from a study examining the relationship between stress and anxiety levels among nonclinical women in Townsville, Queensland, Australia.

Usage

```
StressAnxiety
```

Format

A data frame with 166 observations on 2 variables:

stress numeric. Stress score transformed to the open unit interval (0, 1). Original scale ranged from 0 to 42 on the Depression Anxiety Stress Scales (DASS).

anxiety numeric. Anxiety score transformed to the open unit interval (0, 1). Original scale ranged from 0 to 42 on the DASS.

Details

Both stress and anxiety were assessed using the Depression Anxiety Stress Scales (DASS), ranging from 0 to 42. Smithson and Verkuilen (2006) transformed these scores to the open unit interval (without providing specific details about the transformation method).

The dataset is particularly interesting for demonstrating heteroscedastic relationships: not only does mean anxiety increase with stress, but the variability in anxiety also changes systematically with stress levels. This makes it an ideal case for beta regression with variable dispersion.

Source

Data from Smithson and Verkuilen (2006) supplements. Original data source not specified.

References

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

Examples

```
require(gkwreg)
require(gkwdist)

data(StressAnxiety)

# Example 1: Basic heteroscedastic relationship
# Mean anxiety increases with stress
```

```
# Variability in anxiety also changes with stress
fit_kw <- gkwreg(
  anxiety ~ stress |
    stress,
  data = StressAnxiety,
  family = "kw"
)
summary(fit_kw)

# Interpretation:
# - Alpha: Positive relationship between stress and mean anxiety
# - Beta: Precision changes with stress level
#   (anxiety becomes more/less variable at different stress levels)

# Compare to homoscedastic model
fit_kw_homo <- gkwreg(anxiety ~ stress,
  data = StressAnxiety, family = "kw"
)
anova(fit_kw_homo, fit_kw)

# Example 2: Nonlinear stress effects via polynomial
# Stress-anxiety relationship often shows threshold or saturation effects
fit_kw_poly <- gkwreg(
  anxiety ~ poly(stress, 2) | # quadratic mean
    poly(stress, 2), # quadratic precision
  data = StressAnxiety,
  family = "kw"
)
summary(fit_kw_poly)

# Interpretation:
# - Quadratic terms allow for:
#   * Threshold effects (anxiety accelerates at high stress)
#   * Saturation effects (anxiety plateaus at extreme stress)

# Test nonlinearity
anova(fit_kw, fit_kw_poly)

# Example 3: Exponentiated Kumaraswamy for extreme anxiety patterns
# Some individuals may show very extreme anxiety responses to stress
fit_ekw <- gkwreg(
  anxiety ~ poly(stress, 2) | # alpha: quadratic mean
    poly(stress, 2) | # beta: quadratic precision
    stress, # lambda: linear tail effect
  data = StressAnxiety,
  family = "ekw"
)
summary(fit_ekw)

# Interpretation:
# - Lambda: Linear component captures asymmetry at extreme stress levels
#   (very high stress may produce different tail behavior)
```

```
# Example 4: McDonald distribution for highly skewed anxiety
# Anxiety distributions are often right-skewed (ceiling effects)
fit_mc <- gkwreg(
  anxiety ~ poly(stress, 2) | # gamma
    poly(stress, 2) | # delta
    stress, # lambda: extremity
  data = StressAnxiety,
  family = "mc",
  control = gkw_control(method = "BFGS", maxit = 1500)
)
summary(fit_mc)

# Compare models
AIC(fit_kw, fit_kw_poly, fit_ekw, fit_mc)

# Visualization: Stress-Anxiety relationship
plot(anxiety ~ stress,
  data = StressAnxiety,
  xlab = "Stress Level", ylab = "Anxiety Level",
  main = "Stress-Anxiety Relationship with Heteroscedasticity",
  pch = 19, col = rgb(0, 0, 1, 0.3)
)

# Add fitted curve
stress_seq <- seq(min(StressAnxiety$stress), max(StressAnxiety$stress),
  length.out = 100
)
pred_mean <- predict(fit_kw, newdata = data.frame(stress = stress_seq))
lines(stress_seq, pred_mean, col = "red", lwd = 2)

# Add lowess smooth for comparison
lines(lowess(StressAnxiety$stress, StressAnxiety$anxiety),
  col = "blue", lwd = 2, lty = 2
)
legend("topleft",
  legend = c("Kumaraswamy fit", "Lowess smooth"),
  col = c("red", "blue"), lwd = 2, lty = c(1, 2)
)
```

`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)
```

`terms.gkwreg` *Extract Terms from GKw Regression Model*

Description

Extracts the terms object from a fitted Generalized Kumaraswamy regression model.

Usage

```
## S3 method for class 'gkwreg'
terms(x, ...)
```

Arguments

`x`	An object of class "gkwreg".
`...`	Currently not used.

Value

A terms object.

Author(s)

Lopes, J. E.

See Also

`gkwreg`, `formula.gkwreg`

Examples

```
data(GasolineYield)
fit <- gkwreg(yield ~ batch + temp, data = GasolineYield, family = "kw")
terms(fit)
```

`update.gkwreg` *Update and Re-fit a GKw Regression Model*

Description

Updates and (by default) re-fits a Generalized Kumaraswamy regression model. This method allows modification of the model formula, data, or other arguments without having to completely re-specify the model call. Supports formulas with up to 5 parts (alpha, beta, gamma, delta, lambda) using the **Formula** package.

Usage

```
## S3 method for class 'gkwreg'
update(object, formula., ..., data. = NULL, evaluate = TRUE)
```

Arguments

`object`	An object of class `"gkwreg"`, typically obtained from `gkwreg`.
`formula.`	Changes to the formula. This is a formula where `.` refers to the corresponding part of the old formula. For multi-part formulas (e.g., `y ~ x1 | x2 | x3`), you can update each part separately using the `|` separator.
`...`	Additional arguments to the call, or arguments with changed values. Use `name = NULL` to remove an argument.
`data.`	Optional. A new data frame in which to evaluate the updated model. If omitted, the original data is used.
`evaluate`	Logical. If `TRUE` (default), the updated model is fitted. If `FALSE`, the updated call is returned without fitting.

Details

The `update` method allows you to modify a fitted model and re-fit it with the changes. The GKw regression model supports formulas with up to 5 parts: `y ~ model_alpha | model_beta | model_gamma | model_delta | model_lambda`

Each part can be updated independently using `.` to refer to the current specification:

- `. ~ . + x | . | . | . | .` - Add `x` to alpha only
- `. ~ . | . + x | . | . | .` - Add `x` to beta only
- `. ~ . | . | . + x | . | .` - Add `x` to gamma only
- `. ~ . + x | . + x | . | . | .` - Add `x` to alpha and beta
- `. ~ . - x | . | . | . | .` - Remove `x` from alpha

Omitting parts at the end is allowed (they default to `.`):

- `. ~ . + x | .` is equivalent to `. ~ . + x | . | . | . | .`
- `. ~ . | . + x` is equivalent to `. ~ . | . + x | . | . | .`

Value

If `evaluate = TRUE`, a new fitted model object of class `"gkwreg"`. If `evaluate = FALSE`, an updated call.

Author(s)

Lopes, J. E.

See Also

gkwreg, update, Formula

Examples

```
# Load example data
require(gkwreg)

data(GasolineYield)

# EXAMPLE 1: Simple formulas (1 part - alpha only)

m1_0 <- gkwreg(yield ~ 1, data = GasolineYield, family = "kw")
m1_1 <- update(m1_0, . ~ . + temp)
m1_2 <- update(m1_1, . ~ . + batch)
m1_3 <- update(m1_2, . ~ . - temp)

anova(m1_0, m1_1, m1_2)
AIC(m1_0, m1_1, m1_2, m1_3)
BIC(m1_0, m1_1, m1_2, m1_3)

# EXAMPLE 2: Two-part formulas (alpha | beta)

# Start with intercept-only for both
m2_0 <- gkwreg(yield ~ 1 | 1, data = GasolineYield, family = "kw")

# Add temp to alpha
m2_1 <- update(m2_0, . ~ . + temp | .)

# Add batch to beta
m2_2 <- update(m2_1, . ~ . | . + batch)

# Add batch to alpha too
m2_3 <- update(m2_2, . ~ . + batch | .)

anova(m2_0, m2_1, m2_2, m2_3)
AIC(m2_0, m2_1, m2_2, m2_3)

# EXAMPLE 3: Three-part formulas (alpha | beta | gamma)

m3_0 <- gkwreg(yield ~ 1,
  data = GasolineYield,
  family = "gkw",
```

```
  control = gkw_control(method = "BFGS", maxit = 2000)
)

m3_1 <- update(m3_0, . ~ . + temp | . | .)
m3_2 <- update(m3_1, . ~ . | . + batch | .)
m3_3 <- update(m3_2, . ~ . | . | . + temp)

anova(m3_0, m3_1, m3_2, m3_3)

# EXAMPLE 4: Practical nested model comparison

# Null model
fit0 <- gkwreg(yield ~ 1,
  data = GasolineYield,
  family = "kw",
  control = gkw_control(method = "BFGS", maxit = 2000)
)

# Add main effects to alpha
fit1 <- update(fit0, . ~ . + temp)
fit2 <- update(fit1, . ~ . + batch)

# Model beta parameter
fit3 <- update(fit2, . ~ . | temp)
fit4 <- update(fit3, . ~ . | . + batch)

# Full comparison
anova(fit0, fit1, fit2, fit3, fit4)
AIC(fit0, fit1, fit2, fit3, fit4)
BIC(fit0, fit1, fit2, fit3, fit4)

# EXAMPLE 5: Changing other parameters

# Change family
fit_gkw <- update(fit2, family = "gkw")

# Change link function
fit_logit <- update(fit2, link = list(alpha = "logit"))

# View call without fitting
update(fit2, . ~ . | . + temp, evaluate = FALSE)
```

`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

WeatherTask *Weather Task with Priming and Precise and Imprecise Probabilities*

Description

Data from a cognitive psychology experiment on probabilistic learning and probability judgments. Participants estimated probabilities for weather events under different priming and precision conditions.

Usage

```
WeatherTask
```

Format

A data frame with 345 observations on 4 variables:

agreement numeric. Probability indicated by participants, or the average between minimum and maximum estimates in the imprecise condition. Response variable scaled to (0, 1).

priming factor with levels `two-fold` (case prime) and `seven-fold` (class prime). Indicates the partition priming condition.

eliciting factor with levels `precise` and `imprecise` (lower and upper limit). Indicates whether participants gave point estimates or interval estimates.

Details

All participants in the study were either first- or second-year undergraduate students in psychology, none of whom had a strong background in probability or were familiar with imprecise probability theories.

Task description: Participants were asked: "What is the probability that the temperature at Canberra airport on Sunday will be higher than 'specified temperature'?"

Experimental manipulations:

- **Priming:** Two-fold (simple binary: above/below) vs. seven-fold (multiple temperature categories)
- **Eliciting:** Precise (single probability estimate) vs. imprecise (lower and upper bounds)

The study examines how partition priming (number of response categories) and elicitation format affect probability judgments. Classical findings suggest that more categories (seven-fold) lead to different probability assessments than binary categories (two-fold).

Source

Taken from Smithson et al. (2011) supplements.

References

Smithson, M., Merkle, E.C., and Verkuilen, J. (2011). Beta Regression Finite Mixture Models of Polarization and Priming. *Journal of Educational and Behavioral Statistics*, **36**(6), 804–831. doi:10.3102/1076998610396893

Smithson, M., and Segale, C. (2009). Partition Priming in Judgments of Imprecise Probabilities. *Journal of Statistical Theory and Practice*, **3**(1), 169–181.

Examples

```
require(gkwreg)
require(gkwdist)

data(WeatherTask)

# Example 1: Main effects model
# Probability judgments affected by priming and elicitation format
fit_kw <- gkwreg(
  agreement ~ priming + eliciting,
  data = WeatherTask,
  family = "kw"
)
summary(fit_kw)

# Interpretation:
# - Alpha: Seven-fold priming may shift probability estimates
#   Imprecise elicitation may produce different mean estimates

# Example 2: Interaction model with heteroscedasticity
# Priming effects may differ by elicitation format
# Variability may also depend on conditions
fit_kw_interact <- gkwreg(
  agreement ~ priming * eliciting |
    priming + eliciting,
  data = WeatherTask,
  family = "kw"
)
summary(fit_kw_interact)

# Interpretation:
# - Alpha: Interaction tests if partition priming works differently
#   for precise vs. imprecise probability judgments
# - Beta: Precision varies by experimental condition

# Test interaction
anova(fit_kw, fit_kw_interact)

# Example 3: McDonald distribution for polarized responses
# Probability judgments often show polarization (clustering at extremes)
# particularly under certain priming conditions
fit_mc <- gkwreg(
  agreement ~ priming * eliciting | # gamma
    priming * eliciting | # delta
    priming, # lambda: priming affects polarization
  data = WeatherTask,
  family = "mc",
  control = gkw_control(method = "BFGS", maxit = 1500)
)
summary(fit_mc)

# Interpretation:
```

```
# - Lambda varies by priming: Seven-fold priming may produce more
#   extreme/polarized probability judgments
```

Index