

Package ‘underdisp’

September 30, 2026

Type Package

Title Diagnostics and Models for Underdispersed Count Data

Version 0.1.2

Description Tools for detecting and modeling underdispersion in count data (conditional variance below the conditional mean), the case the Poisson and negative binomial defaults cannot represent. Provides a screening diagnostic that benchmarks at-risk dispersion against a zero-truncated Poisson, regression-adjusted tests of equidispersion, and a dispersion profile that compares the variance-to-mean curves of competing families against the data; the continuous parameter binomial (CPB) and generalized event count (Katz) regressions with zero-truncated, hurdle, and zero-inflated forms and high-dimensional fixed effects with a split-panel jackknife bias correction; matched Poisson, negative binomial, COM-Poisson (rate- and mean-parameterized), generalized Poisson, gamma-count, and double Poisson regressions through the same interface, with frequency weights, offsets, and analytic, robust, and cluster-robust standard errors; bootstrap and profile-likelihood inference; proper scoring rules, rootograms, PIT histograms, and simulation methods; and quantities of interest including predicted distributions, the implied ceiling, rate ratios, and first differences with an extensive/intensive decomposition. The likelihoods are implemented in C++.

License GPL-3

Encoding UTF-8

LazyData true

Depends R (≥ 4.0)

Imports Rcpp, stats, MASS, VGAM, graphics, methods, numDeriv, parallel

LinkingTo Rcpp

Suggests sandwich, pscl, DHARMA, testthat ($\geq 3.0.0$), knitr, rmarkdown, broom, modelsummary, texreg, gamlss.dist, rmutl, glmmTMB, COM-PoissonReg, Ecdat, wooldridge

VignetteBuilder knitr

RoxygenNote 7.3.3

URL <https://CRAN.R-project.org/package=underdisp>,

<https://github.com/bagozzib/underdisp>,

<https://bagozzib.github.io/underdisp/>

BugReports <https://github.com/bagozzib/underdisp/issues>

Config/testthat/edition 3

NeedsCompilation yes

Author Benjamin E. Bagozzi [aut, cre] (ORCID:

<<https://orcid.org/0000-0002-6233-6453>>)

Maintainer Benjamin E. Bagozzi <bagozzib@udel.edu>

Repository CRAN

Date/Publication 2026-09-30 21:20:08 UTC

Contents

alpha_confint	3
augment.cpb	4
calibrate_alpha	5
compare_dispersion	6
compare_models	8
compois-distribution	9
confint.cpb	10
confint.underdisp	10
count_reg	12
cpb	14
cpb-distribution	17
cpb_fe	18
cv_score	20
dispersion_profile	22
dispersion_test	23
doublepois-distribution	25
first_difference	27
first_difference.gec	28
first_difference.hurdle_cpb	30
fitted.underdisp	31
gammacount-distribution	32
gec	33
gec-distribution	35
gec_fe	36
genpois-distribution	38
glance.cpb	39
hurdle_count	40
hurdle_cpb	41
hurdle_gec	43
implied_ceiling	44

irr	45
irr.count_reg	46
mundlak	47
peacekeeping	48
pit_hist	49
predict.count_reg	50
predict.cpb	50
predict.cpb_fe	51
predict.gec	52
predict.hurdle_count	53
predict.hurdle_cpb	54
predict.hurdle_gec	54
predict.zi_count	55
predict.zi_cpb	56
predict.zi_gec	56
rcpb	57
rhurdle_cpb	57
rootogram	58
rzicpb	59
score	60
simulate.underdisp	61
summary.cpb	62
tidy.cpb	63
ud_screen	64
vcov.cpb_fe	66
zi_count	67
zi_cpb	68
zi_gec	70
zi_test	71

Index	73
--------------	-----------

alpha_confint	<i>Profile-likelihood interval for the dispersion parameter alpha</i>
---------------	---

Description

The interval inverts the likelihood-ratio test of alpha, re-maximizing the coefficients at each value. By default the cut is the chi-square one (a first-order interval); it covers about 0.90 to 0.94 in simulations from the CPB, with nearly all misses on the upper side, because the estimate of alpha is biased toward zero ([calibrate_alpha\(\)](#) explains the mechanism). A fit that went through [calibrate_alpha\(\)](#) gets the interval calibrated by parametric bootstrap instead. Either interval is model-based: it assumes the CPB and independent observations.

Usage

```
alpha_confint(object, level = 0.95)
```

Arguments

object	A "cpb" object.
level	Confidence level (default 0.95).

Value

A length-2 numeric vector (lower, upper) with attributes alpha (the point estimate), method (first-order or calibrated) and boundary (TRUE when the profile has not fallen to the cut by $\alpha = 0.005$, so that the lower limit is the parameter bound).

Examples

```
set.seed(7); x <- rnorm(200)
N <- pmax(round(exp(1.5 + 0.4 * x) / 0.5), 1); y <- rbinom(200, N, 0.5)
fit <- cpb(y ~ x, data.frame(y = y, x = x)[y > 0, ], se = "none")
alpha_confint(fit)
```

augment.cpb

Augment data with CPB fitted values (broom method)

Description

Augment data with CPB fitted values (broom method)

Usage

```
## S3 method for class 'cpb'
augment(x, ...)
```

Arguments

x	A "cpb" object.
...	Unused.

Value

A data frame with fitted values, response residuals, and the implied per-observation ceiling.

calibrate_alpha	<i>Calibrate the interval for the CPB dispersion parameter</i>
-----------------	--

Description

An opt-in step that stores, in the fit, the parametric-bootstrap distribution of the signed root of the profile likelihood ratio for alpha. Once it is stored, `alpha_confint()`, `confint.cpb()`, `implied_ceiling()` and `summary()` report the calibrated interval instead of the first-order one.

Usage

```
calibrate_alpha(object, B = 199, cores = 1L, seed = NULL, ...)
```

```
## S3 method for class 'cpb'
```

```
calibrate_alpha(object, B = 199, cores = 1L, seed = NULL, ...)
```

```
## S3 method for class 'hurdle_cpb'
```

```
calibrate_alpha(object, B = 199, cores = 1L, seed = NULL, ...)
```

```
## Default S3 method:
```

```
calibrate_alpha(object, B = 199, cores = 1L, seed = NULL, ...)
```

Arguments

object	A pooled <code>cpb()</code> fit, or a <code>hurdle_cpb()</code> fit whose intensity is a pooled CPB.
B	Number of bootstrap replicates (default 199).
cores	Worker processes for the refits.
seed	Optional seed for the simulated responses; the caller's random number state is restored afterwards, as in <code>stats::simulate()</code> .
...	Unused.

Details

The first-order interval cuts the profile log-likelihood $qchisq(level, 1) / 2$ below its maximum on both sides. For the CPB that cut is not calibrated: the support $0, \dots, \text{floor}(\lambda / (1 - \alpha))$ moves with the parameters, the log-ceiling coefficients behave like endpoint parameters, and $\hat{\alpha}$ inherits a bias toward zero from them. In simulations from the CPB the first-order 95% interval covers between 0.90 and 0.94 across a grid of nine designs, and nearly all of its misses have the true α above the upper limit.

The calibration simulates B responses from the fitted model (the fitted rates, $\hat{\alpha}$, the same truncation and offset), refits each one, and records the signed root $r^* = \text{sign}(\alpha^* - \hat{\alpha}) \sqrt{2(1 - 1^*_p(\alpha))}$. The interval is then $\{\alpha : q_{lo} \leq r(\alpha) \leq q_{hi}\}$, with q_{lo} and q_{hi} the order statistics of the r^* at ranks $(B + 1)(1 - level)/2$ from each end (the 5th smallest and 5th largest of 199 at the 95% level), so the upper limit sits $q_{lo}^2 / 2$ and the lower limit $q_{hi}^2 / 2$ below the maximum of the profile. In the same simulations the calibrated 95% interval covers 0.92 to 0.96, with its misses balanced between the two sides. The first-order interval falls short only when the

design has many distinct covariate patterns (the endpoint effect needs many distinct ceilings): with an intercept only, or a covariate taking ten or fewer values, it covers at or above its level, and the calibration, which stays close to the nominal level there too (0.95 with a binary covariate), is not needed.

The responses are all drawn first, on the calling process, and the refits use no random numbers, so the result depends on seed but not on cores. With $B = 199$ the two cuts carry simulation error (a standard deviation of about 0.2 in q_{lo} and q_{hi}), which moves a limit by a fraction of its distance from the estimate; $B = 999$ reduces it and is needed for a 99% interval. Each replicate costs about a third of the original fit.

The interval is model-based. It assumes the CPB and independent observations, and it is not cluster-robust: when the fit was given `cluster`, a message says so. Under misspecification α has no fixed target (on data with a soft upper tail its estimate rises with the sample size), and no interval computed under the fitted CPB repairs that.

Calibration is refused, with the reason, where it has nothing to work with: a fit whose ceiling reaches `max.support` (the guard, not the data, bounds α), weights that are not whole numbers, and a fit saved by version 0.1.0. Whole-number weights are treated as frequency weights: a row of weight w contributes w independent simulated responses.

Value

object, with the component `alpha.cal`: a list holding the signed roots r (NA for a replicate that failed to fit), B , B_{ok} and `seed`.

See Also

[alpha_confint\(\)](#), [dispersion_test\(\)](#)

Examples

```
set.seed(7); x <- rnorm(100)
d <- data.frame(y = rcpb(100, exp(1.5 + 0.4 * x), 0.5, truncated = TRUE), x = x)
fit <- cpb(y ~ x, d, se = "none")
alpha_confint(fit) # first-order
## 19 replicates keep the example short and can serve a 90% interval; use 199 or more
fit <- calibrate_alpha(fit, B = 19, seed = 1)
alpha_confint(fit, level = 0.90) # calibrated
```

compare_dispersion	<i>Compare count-model dispersion across the model family</i>
--------------------	---

Description

Fits the Poisson and negative binomial defaults alongside the two underdispersed workhorses—the soft-tail Conway–Maxwell–Poisson (fit natively, no external dependency) and the hard-ceiling CPB—and returns a comparison on both fit (log-likelihood, AIC, BIC) and calibration (mean logarithmic score and ranked probability score, lower is better, plus the fitted share of zeros against the

observed share). Whether the tail is better described by an accelerated decay (COM-Poisson) or a binding ceiling (CPB) is a testable question, not an assumption; the scoring rules are the calibration counterpart to the information criteria. The CPB's ceiling-exceedance share (observations above the implied ceiling) is reported as a falsification check on the hard bound.

Usage

```
compare_dispersion(
  formula,
  data,
  max.support = 500,
  hurdle = FALSE,
  zi = FALSE
)
```

Arguments

<code>formula</code>	A model formula.
<code>data</code>	A data frame.
<code>max.support</code>	Passed to cpb() .
<code>hurdle</code>	If TRUE and the data contain zeros, also fit and score a hurdle_cpb() .
<code>zi</code>	If TRUE and the data contain zeros, also fit and score a zi_cpb() . The mixture EM is slow on large panels, so it is a separate opt-in from hurdle.

Details

With `hurdle = TRUE` and zeros present, a hurdle-CPB is added, so a zero-inflated underdispersed process can be compared to the single-equation models on the same footing.

Value

A list with `table` (a data frame of `df`, `logLik`, `AIC`, `BIC`, `logscore`, `rps`, and `zero_fit` per model), `obs_zero` (the observed zero share), `nu` (the COM-Poisson dispersion, `>1` = underdispersion, `NA` if the COM-Poisson fit failed), `alpha` (the CPB shape parameter), `ceiling_exceedance` (share of observations above the CPB ceiling), and `cpb_ok` (FALSE if the single-equation CPB could not satisfy its feasibility constraint on the data, e.g. under heavy zero-inflation with a wide count range — itself a signal that a hurdle or zero-inflated model is needed).

See Also

[ud_screen\(\)](#), [cpb\(\)](#), [score\(\)](#)

Examples

```
set.seed(1); x <- rnorm(120)
N <- pmax(round(exp(1.6 + 0.4 * x) / 0.5), 1); y <- rbinom(120, N, 0.5)
compare_dispersion(y ~ x, data = data.frame(y = y, x = x))$table
```

compare_models	<i>Compare fitted underdispersed-count models</i>
----------------	---

Description

Compares fitted models from this package — `cpb`, `cpb_fe`, `hurdle_cpb`, or `zi_cpb`, in any combination of fixed effects and robust/clustered standard errors — on information criteria and, where a predicted distribution is available, on proper scores. Unlike `zi_test()` this is not a hypothesis test: the models need not be nested, so it is the appropriate tool for the non-nested hurdle-versus-mixture choice. Passing an object that is not a fitted model from this package is an error, and models fit on different data raise a warning.

Usage

```
compare_models(...)
```

Arguments

... Two or more fitted models (`cpb`, `cpb_fe`, `hurdle_cpb`, `zi_cpb`), optionally named.

Value

A data frame with `df`, `logLik`, `AIC`, `BIC`, and (where the predicted distribution is available) `logscore` and `rps`, one row per model, ordered by AIC.

See Also

[zi_test\(\)](#), [compare_dispersion\(\)](#)

Examples

```
set.seed(1); n <- 400; x <- rnorm(n); z <- rnorm(n)
y <- rhurdle_cpb(n, exp(1.2 + 0.5 * x), 0.5, plogis(-0.2 + 0.8 * z))
d <- data.frame(y = y, x = x, z = z)
compare_models(hurdle = hurdle_cpb(y ~ x, data = d, participation = ~ z),
               zi = zi_cpb(y ~ x, data = d, zero = ~ z))
```

compois-distribution *Conway–Maxwell–Poisson distribution functions*

Description

Density, distribution, quantile, and random generation for the COM-Poisson with rate λ (or mean μ) and dispersion ν ($\nu > 1$ underdispersed, $\nu = 1$ Poisson, $\nu < 1$ overdispersed). Complements the estimators `count_reg(..., family = "compois")` (rate parameterization) and `count_reg(..., family = "mpcmp")` (mean parameterization).

Usage

```
dcompois(x, lambda, nu, log = FALSE, mu = NULL)

pcompois(q, lambda, nu, lower.tail = TRUE, log.p = FALSE, mu = NULL)

qcompois(p, lambda, nu, lower.tail = TRUE, log.p = FALSE, mu = NULL)

rcompois(n, lambda, nu, mu = NULL)
```

Arguments

<code>x, q</code>	Vector of quantiles (non-negative integers).
<code>lambda</code>	Rate parameter (scalar or vector, recycled). Give <code>mu</code> instead to specify the distribution by its mean.
<code>nu</code>	Dispersion parameter (scalar).
<code>log, log.p</code>	Return log probabilities.
<code>mu</code>	Optional mean (scalar or vector, recycled); when supplied, the rate λ solving $E(Y) = \mu$ is found numerically (Huang's 2017 mean parameterization, the one <code>count_reg(family = "mpcmp")</code> fits) and λ is ignored.
<code>lower.tail</code>	If TRUE (default), $P(X \leq x)$.
<code>p</code>	Vector of probabilities.
<code>n</code>	Number of draws.

Value

`dcompois` a density, `pcompois` a CDF, `qcompois` a quantile, `rcompois` a numeric vector of count draws.

Examples

```
dcompois(0:5, lambda = 3, nu = 1.5)
mean(rcompois(1000, lambda = 3, nu = 1.5))
```

confint.cpb

Confidence intervals for a CPB fit

Description

Coefficient intervals use the cold-multistart bootstrap percentile method (validated to nominal coverage). The interval for alpha is a profile-likelihood interval: first-order by default (it covers about 0.90 to 0.94 in simulations from the CPB, with its misses on the upper side; see [calibrate_alpha\(\)](#) for the reason), and calibrated by parametric bootstrap when the fit went through [calibrate_alpha\(\)](#).

Usage

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

Arguments

object	A "cpb" object fit with se = "bootstrap".
parm	Optional subset of parameters (coefficient names and/or "alpha").
level	Confidence level (default 0.95).
...	Unused.

Value

A matrix of lower/upper bounds.

confint.underdisp

Confidence intervals for every model class

Description

confint() methods for the whole family, one interval rule per inference type: bootstrap percentile intervals for [cpb\(\)](#) (coefficients; the dispersion parameter gets its profile-likelihood interval, first-order or calibrated by [calibrate_alpha\(\)](#)) and [gec\(\)](#) (coefficients and delta); normal-approximation intervals from the bootstrap standard errors for the fixed-effects fits ([cpb_fe\(\)](#), [gec_fe\(\)](#)) and the zero-inflated mixtures ([zi_cpb\(\)](#), [zi_gec\(\)](#)); Wald intervals from the analytic covariance for [count_reg\(\)](#) and [zi_count\(\)](#) (the dispersion parameter's interval is mapped from its estimation scale to the natural scale); and, for the hurdles, the participation model's Wald intervals (prefixed participation:) stacked over the intensity model's intervals (prefixed intensity:); the zero-inflated classes prefix count: and zero:. The same names label vcov(), tidy(), and the texreg tables. A fit without inference errors informatively.

Usage

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

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

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

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

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

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

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

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

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

Arguments

object	A fitted model from this package.
parm	Optional subset of parameter names.
level	Confidence level (default 0.95).
...	Unused.

Value

A two-column matrix of lower and upper bounds.

See Also

[confint.cpb\(\)](#)

count_reg	<i>Matched count regressions: Poisson, negative binomial, COM-Poisson, generalized Poisson, gamma-count, double Poisson</i>
-----------	---

Description

Fits a count regression with a log link from any of the package's count families, optionally zero-truncated and/or with unit fixed effects, returning an object that `compare_models()`, `score()`, `dispersion_test()`, `dispersion_profile()`, and the **broom** methods treat on the same footing as a `cpb()` or `gec()` fit. The families share one interface, so a gamma-count and a COM-Poisson can be compared to a CPB with identical degrees-of-freedom, log-likelihood, proper-score, weight, and robust-standard-error accounting rather than reconciled across packages.

Usage

```
count_reg(
  formula,
  data,
  family = c("poisson", "negbin", "compois", "mpcmp", "genpois", "gammacount",
    "doublepois"),
  truncated = FALSE,
  fe = NULL,
  offset = NULL,
  weights = NULL,
  se = c("analytic", "robust", "cluster", "none"),
  cluster = NULL
)
```

Arguments

formula	A model formula.
data	A data frame.
family	One of "poisson", "negbin", "compois", "mpcmp", "genpois", "gammacount", "doublepois" (see Families).
truncated	Logical; if TRUE, fit the zero-truncated form (requires all $Y \geq 1$), the count analogue of the CPB's zero-truncated default.
fe	Optional column name(s) for fixed effects, entered as factor dummies (so the degrees of freedom count each absorbed intercept, matching <code>cpb_fe()</code>). Pass a vector of two columns for two-way (e.g. unit and time) fixed effects.
offset	Optional offset entered on the linear-predictor (log) scale – the log-mean for the mean-parameterized families, the log-rate ($\log \lambda$) for "compois". A numeric vector or the name of a column in data, e.g. <code>log(exposure)</code> so the model becomes a rate model.

weights	Optional frequency weights: a numeric vector or the name of a column in data. A weight of w_i is equivalent to w_i copies of row i in the log-likelihood, the information, and the score products, and <code>nobs()</code> returns the weight total. Survey (probability) weights call for <code>se = "robust"</code> or <code>se = "cluster"</code> .
se	Standard errors: "analytic" (inverse information, default), "robust" (heteroskedasticity-consistent sandwich), "cluster" (cluster-robust; needs <code>cluster</code>), or "none".
cluster	Optional cluster identifier (a column name in data or a vector aligned to its rows) for <code>se = "cluster"</code> ; supplying it selects <code>se = "cluster"</code> unless <code>se</code> is given explicitly.

Value

An object of class "count_reg". The component `$theta` holds the dispersion parameter on its natural scale (the negative-binomial size, the COM-Poisson ν , the generalized-Poisson λ , the gamma-count α , the double-Poisson θ ; NA for the Poisson); `$fitted.values` and `$residuals` are on the mean scale (the conditional mean $E(Y | Y \geq 1)$ for a zero-truncated fit), as `fitted()` and `residuals()` return them, and `$mu` is the family's natural parameter $\exp(\text{offset} + x'b)$.

Families

- "poisson": the equidispersed baseline.
- "negbin": negative binomial (overdispersion only); `$theta` is the size.
- "compois": the classical Conway–Maxwell–Poisson in its rate parameterization, $\log \lambda = x'\beta$. The rate λ is not the mean (though `predict(type = "response")` and `fitted()` return the mean); `$theta` is the dispersion ν ($\nu > 1$ underdispersed, $\nu = 1$ Poisson, $\nu < 1$ overdispersed).
- "mpcmp": the COM-Poisson in Huang's (2017) mean parameterization, $\log E(Y) = x'\beta$, with the same dispersion ν ; the coefficients are effects on the log mean and rate ratios are exact.
- "genpois": the Consul–Jain generalized Poisson with constant dispersion $\lambda \in (-1, 1)$ and mean $\mu = \exp(x'\beta)$, so $\text{Var}/\text{Mean} = 1/(1 - \lambda)^2$; $\lambda < 0$ is underdispersion, on a finite support that the pmf is renormalized over (see [genpois-distribution](#)).
- "gammacount": Winkelmann's (1995) gamma-count renewal-process model, underdispersed when the waiting times between events are more regular than exponential ($\alpha > 1$); $\exp(x'\beta)$ is the long-run event rate and the exact mean is reported (see [gammacount-distribution](#)).
- "doublepois": Efron's (1986) double Poisson with the exact normalizing constant, $\theta > 1$ underdispersed (see [doublepois-distribution](#)).

The Poisson is the equidispersed member of every family, so `dispersion_test()` tests each family's dispersion parameter against it, and `dispersion_profile()` compares the families' implied variance-to-mean curves against the data.

Runtime: the Poisson, negative binomial, gamma-count, and double Poisson and generalized Poisson families fit in under a second at a few hundred rows; the COM-Poisson families evaluate a normalizing sum per observation (and the mean parameterization solves a root per observation), so they take seconds at a few hundred rows and minutes at several thousand; `se = "robust"` and `"cluster"` add numerical scores at the same cost per parameter.

References

Huang, A. (2017). Mean-parametrized Conway–Maxwell–Poisson regression models for dispersed counts. *Statistical Modelling*, 17(6), 359-380. Winkelmann, R. (1995). Duration dependence and dispersion in count-data models. *Journal of Business & Economic Statistics*, 13(4), 467-474. Efron, B. (1986). Double exponential families and their use in generalized linear regression. *Journal of the American Statistical Association*, 81(395), 709-721. Consul, P. C. and Famoye, F. (1992). Generalized Poisson regression model. *Communications in Statistics – Theory and Methods*, 21(1), 89-109.

See Also

[cpb\(\)](#), [gec\(\)](#), [compare_models\(\)](#), [dispersion_test\(\)](#), [dispersion_profile\(\)](#), [hurdle_count\(\)](#), [zi_count\(\)](#)

Examples

```
set.seed(1); n <- 400; x <- rnorm(n)
y <- rgamma(n, exp(1 + 0.5 * x), alpha = 2)      # regular event timing
d <- data.frame(y = y, x = x)
m <- count_reg(y ~ x, data = d, family = "gamma")
m
compare_models(poisson = count_reg(y ~ x, d, family = "poisson"), gamma = m)
```

cpb

Fit a Continuous Parameter Binomial (CPB) regression

Description

Fits the underdispersed continuous parameter binomial model of King (1989), in which the conditional variance is a fraction of the conditional mean, $\text{Var}(Y | x) = \alpha E(Y | x)$ with $0 < \alpha < 1$, and each observation has an endogenous ceiling $\lambda_i / (1 - \alpha)$. A zero-truncated variant (the default) conditions on $Y \geq 1$, appropriate when underdispersion lives among the positive counts of an otherwise zero-inflated outcome.

Usage

```
cpb(
  formula,
  data,
  truncated = TRUE,
  se = c("none", "bootstrap"),
  B = 500,
  cluster = NULL,
  offset = NULL,
  weights = NULL,
  cores = 1L,
  alpha.start = 0.5,
```

```

    max.support = NULL,
    maxit = 20000,
    reltol = 1e-08
  )

```

Arguments

formula	A model formula.
data	A data frame.
truncated	Logical; if TRUE (default) fit the zero-truncated CPB (requires all $Y \geq 1$); if FALSE fit the untruncated CPB on $Y \geq 0$. Note the sibling default differs: <code>cpb_fe()</code> defaults to <code>truncated = FALSE</code> , because its typical call fits a whole panel including zeros, while <code>cpb()</code> 's typical call fits the positive counts of a zero-inflated outcome. State <code>truncated</code> explicitly when moving a specification between the two.
se	Inference method: "none" (default; fast, no standard errors) or "bootstrap" (cold-multistart pairs/cluster bootstrap; slower).
B	Number of bootstrap resamples (default 500).
cluster	Optional cluster identifier for cluster-robust inference: a column name in data or a vector aligned to its rows. When supplied, the bootstrap resamples whole clusters (a block bootstrap), giving cluster-robust standard errors and intervals; when NULL (default) it resamples observations, giving heteroskedasticity-robust errors. This is the appropriate route to robust inference here because the CPB's parameter-dependent support makes a Hessian-based sandwich unreliable.
offset	Optional offset on the log-mean scale (an exposure): a numeric vector or the name of a column in data, making the model a rate model.
weights	Optional frequency weights: a numeric vector or the name of a column in data. A weight of w_i is equivalent to w_i copies of row i ; the bootstrap resamples rows together with their weights, and <code>nobs()</code> returns the weight total.
cores	Number of worker processes for the bootstrap (default 1). With <code>cores > 1</code> the replicates run on a socket cluster whose random streams are seeded from the calling session, so <code>set.seed()</code> before the call reproduces a parallel run (parallel and serial draws differ).
alpha.start	Starting value for the dispersion parameter (default 0.5); the fit also multi-starts over a spread of alpha values.
max.support	Guard on the maximum evaluated support: parameter values whose implied ceiling exceeds it are treated as infeasible (they lie in the alpha near 1 region where the support explodes). The default NULL sets it to <code>max(500, 10 * max(y))</code> . A fit whose largest ceiling reaches the guard is flagged with a warning and in <code>\$support_binding</code> , because alpha is then bounded by the guard rather than by the data.
maxit, reltol	Optimizer controls passed to <code>stats::optim()</code> .

Details

The mean is modelled log-linearly, $\lambda_i = \exp(x_i'\beta)$. Because the support $0, \dots, \lfloor \lambda_i/(1-\alpha) \rfloor$ depends on the parameters, the log-likelihood is discontinuous: whenever a parameter move carries an observation's ceiling across an integer its normalizing constant jumps, by about $(1-\alpha)^k$ at ceiling k . The maximizer is found by a deterministic multistart, BFGS from the Poisson solution at each value of `alpha.start` polished by Nelder-Mead with feasibility-repaired restarts, on covariates scaled to unit standard deviation (the coefficients are mapped back, so the fit does not depend on the covariates' units), and the reported maximum is the maximum of the fit's own profile in `alpha`: the profile is traced by continuation on both sides of the estimate (the slopes re-maximized at each step from the neighbouring solution) until it has dropped four log-likelihood units, and a trace point above the multistart's value restarts the fit from there. The same trace gives the profile interval of `confint.cpb()`. Two parameterizations of the same design (say, treatment and sum contrasts) can still settle on different teeth of this surface, with log-likelihoods differing by the order of the jumps; unit intercepts belong in the fixed-effects estimator `cpb_fe()`, which solves each of them exactly (a pooled fit with many dummy columns can stop short of it). The numerical Hessian is unreliable on such a surface, so inference uses a cold-multistart bootstrap for the coefficients (validated to nominal coverage) and a profile-likelihood interval for α (see `alpha_confint()`; first-order by default, calibrated by parametric bootstrap after `calibrate_alpha()`).

Runtime: a fit with `se = "none"` takes a few seconds at 500 rows and about fifteen at 2,000 on one core, the nine starts, the profile trace, and the scan in `alpha` included. The bootstrap replicates are maximized by the multistart without the profile trace (about a third of a second each at 500 rows), so a replicate can sit on a different tooth from the point estimate by the order of the jumps; cores runs them in parallel.

Value

An object of class "cpb": a list with `coefficients`, `alpha`, `bootstrap.se.beta/vcov/ci.beta`, `loglik`, `loglik.null`, `fitted.values`, `ceiling` (the observation-specific implied ceiling), the model frame pieces, and the original call.

References

King, G. (1989). Variance specification in event count models. *American Journal of Political Science*, 33(3), 762-784.

See Also

`ud_screen()`, `confint.cpb()`, `predict.cpb()`, `implied_ceiling()`, `dispersion_test()`

Examples

```
set.seed(1)
n <- 200; x <- rnorm(n)
N <- pmax(round(exp(1.6 + 0.5 * x) / 0.5), 1)
y <- rbinom(n, N, 0.5) # underdispersed (var/mean approx 0.5)
d <- data.frame(y = y, x = x)
fit <- cpb(y ~ x, data = d[d$y > 0, ], se = "none")
summary(fit)
```

cpb-distribution *Continuous parameter binomial distribution functions*

Description

Density, distribution function, and quantile function for the continuous parameter binomial (CPB), and its zero-truncated form. The CPB has a hard ceiling at $\lfloor \lambda / (1 - \alpha) \rfloor$; probability above it is zero. Complements the simulator [rcpb\(\)](#).

Usage

```
dcpb(x, lambda, alpha, truncated = FALSE, log = FALSE)
```

```
pcpb(q, lambda, alpha, truncated = FALSE, lower.tail = TRUE, log.p = FALSE)
```

```
qcpb(p, lambda, alpha, truncated = FALSE, lower.tail = TRUE, log.p = FALSE)
```

Arguments

x, q	Vector of quantiles (non-negative integers).
lambda	Mean parameter (scalar or vector, recycled).
alpha	Shape parameter in (0, 1).
truncated	If TRUE, use the zero-truncated CPB. Where the implied ceiling $\lambda / (1 - \alpha)$ is below 1 the CPB puts all its mass on 0, and its zero-truncated form is the point mass at 1, the zero-truncated distribution for every ceiling in $[1, 2)$.
log, log.p	If TRUE, probabilities are given as log.
lower.tail	If TRUE (default), probabilities are $P(X \leq x)$.
p	Vector of probabilities.

Value

dcpb a density, pcpb a distribution function, qcpb a quantile.

See Also

[rcpb\(\)](#)

Examples

```
dcpb(0:5, lambda = 3, alpha = 0.5)
pcpb(3, lambda = 3, alpha = 0.5)
qcpb(0.9, lambda = 3, alpha = 0.5)
```

Description

Fits the CPB with a full set of unit fixed effects by concentrating (profiling) out the unit intercepts. Each unit's intercept is solved by a one-dimensional inner maximization, so the outer optimizer handles only the covariate coefficients and the dispersion parameter. This avoids an explicit unit-dummy design matrix and scales to thousands of units, where dummy-based fitting exhausts memory. The concentrated log-likelihood equals the full-dummy log-likelihood to the tolerance of the inner search, which is solved exactly. Each unit's objective is a saw-tooth in its intercept: whenever the intercept crosses $\log(k(1 - \alpha)) - o_{it}$ for an integer k the support of observation t gains the point k , its normalizing constant grows, and the objective drops by about $(1 - \alpha)^k$; between such breakpoints it is smooth and unimodal. Its supremum is therefore attained at an interior critical point of one tooth or as the left limit at a breakpoint, and the inner search evaluates every breakpoint within a window of the envelope's peak (located on a coarse grid around the unit's Poisson intercept) and refines inside whole teeth by golden section; every evaluation of the concentrated likelihood searches each unit afresh, so the objective is a function of the parameters alone. Solving the inner problem to its supremum makes the concentrated likelihood continuous in alpha (with kinks where a unit's maximizing tooth switches) but not in the slopes: where a unit's supremum sits on its feasibility floor (an observation whose count equals its ceiling) the intercept cannot retreat, so when another observation's breakpoint crosses that floor the concentrated likelihood drops by about $(1 - \alpha)^k$, a cliff in the slopes on whose edge the maximizer can rest with a non-zero one-sided gradient. The outer optimizer is therefore a deterministic multistart: BFGS with the analytic envelope gradient from the Poisson slopes and two dispersion starts, a Nelder-Mead polish, and two perturbed restarts; alternative local optima differ by the order of the jumps (a few hundredths of a log-likelihood unit on a 12-unit panel), far inside the bootstrap variability of the estimates.

Usage

```
cpb_fe(
  formula,
  data,
  fe,
  truncated = FALSE,
  se = c("none", "bootstrap"),
  B = 500,
  cluster = NULL,
  offset = NULL,
  weights = NULL,
  cores = 1L,
  max.support = NULL,
  inner_it = 30L,
  maxit = 3000L,
  reltol = 1e-07,
  bias_correct = c("none", "jackknife")
)
```

Arguments

formula	A model formula for the covariates only. Do not include the unit factor; supply it via fe. The intercept is absorbed by the fixed effects.
data	A data frame.
fe	Name of the column holding the unit identifier.
truncated	Logical; fit the zero-truncated CPB if TRUE. Note the sibling default differs: <code>cpb_fe()</code> defaults to FALSE (whole-panel fits including zeros), while <code>cpb()</code> defaults to TRUE (positives-only fits). State truncated explicitly when moving a specification between the two.
se	Inference for the covariate coefficients: "none" (default) or "bootstrap", a pairs/cluster bootstrap.
B	Bootstrap resamples when <code>se = "bootstrap"</code> .
cluster	Cluster for the bootstrap. NULL (default) resamples the fixed-effects units themselves (unit-clustered inference, the natural panel default); a column name or vector resamples those clusters while preserving the unit fixed effects. Duplicated blocks are relabeled so the concentrated likelihood treats them as distinct units.
offset	Optional offset on the log-mean scale (an exposure): a numeric vector or the name of a column in data.
weights	Optional frequency weights (a numeric vector or a column name); see <code>cpb()</code> .
cores	Worker processes for the bootstrap; see <code>cpb()</code> .
max.support	Guard on the maximum evaluated support; NULL (default) sets $\max(500, 10 * \max(y))$. See <code>cpb()</code> .
inner_it	Golden-section iterations refining each unit's inner maximization after the grid scans (see Details).
maxit, reltol	Outer optimizer controls.
bias_correct	"none" (default) or "jackknife", the split-panel jackknife correction for the $1/T$ incidental-parameters bias (see Details).

Details

Short panels: the dispersion parameter α and the fixed effects are subject to the incidental-parameters bias of nonlinear fixed-effects estimation. The bias in α is downward and of order $1/T$, where T is the per-unit number of observations. It is small for T at least about 30 and should be treated cautiously for short panels. The covariate coefficients are not materially affected. `bias_correct = "jackknife"` removes the leading $1/T$ term by the split-panel (half-panel) jackknife: the model is refit on the first and second temporal halves of every unit's series (rows are split in the order supplied, which should be temporal order) and the corrected estimate is $2 * \text{full} - \text{mean}(\text{halves})$. In the package's fixed-effects bias Monte Carlo (150 units, $\alpha = 0.5$) the α bias at $T = 6/10/20/40$ falls from $-0.153/-0.100/-0.057/-0.033$ to $-0.046/-0.024/-0.013/-0.009$. With correction on, coefficients and α are the corrected estimates (the maximum-likelihood values are kept in `$uncorrected`), the unit effects and fitted values are re-concentrated at the corrected parameters, and `logLik/AIC` continue to refer to the maximum-likelihood fit.

Validity gate: the split-panel identity requires the two half-panels to estimate the same pseudo-true parameter (time-homogeneity; Dhaene & Jochmans 2015, Review of Economic Studies). On trending or time-heterogeneous panels the correction is invalid, so the function REFUSES it – returning the uncorrected maximum-likelihood fit with a warning naming the failed check – whenever the corrected dispersion leaves its feasible space or the two half-panel dispersion estimates disagree beyond what the 1/T bias can explain. A refusal is diagnostic information about the panel, not an error. The gate is deliberately powered over sized: in the package’s calibration Monte Carlo it refuses about 7 percent of genuinely time-homogeneous panels (a conservative nuisance; the returned fit is exactly the ordinary maximum-likelihood estimate) while catching 98 percent of dispersion regime changes and 100 percent of smooth unmodeled trends – the cases where an uncaught correction would be silently wrong.

One set of fixed effects is concentrated out. For two-way (e.g. unit and time) fixed effects, add `factor(time)` to formula (entered as dummies), or use `count_reg()`, which supports two-way fixed effects natively.

Value

An object of class "cpb_fe" with coefficients, alpha, loglik, fe (the estimated unit intercepts), fitted.values, and the implied per-observation ceiling.

See Also

`cpb()`, `ud_screen()`

Examples

```
set.seed(1)
d <- do.call(rbind, lapply(1:25, function(i) {
  x <- rnorm(12); lam <- exp(rnorm(1, 0, 0.5) + 0.5 * x)
  N <- pmax(round(lam / 0.5), 1)
  data.frame(unit = i, x = x, y = rbinom(12, N, 0.5))
}))
fit <- cpb_fe(y ~ x, data = d, fe = "unit")
fit
```

cv_score

Cross-validated proper scores for a count model

Description

K-fold cross-validated log score and RPS: the data are split into k folds, the model is refit on each training set via `fitfun`, and its predicted distribution is scored on the held-out fold. Because the score is genuinely out of sample it penalizes over-parameterization, so it is the honest criterion for comparing models that differ in the number of parameters (for example a hurdle with per-unit participation fixed effects against a zero-inflated model). Per-observation held-out scores are returned so two models fit on the *same* folds can be compared with a paired, cluster-robust test.

Usage

```
cv_score(
  fitfun,
  data,
  k = 5,
  kmax = NULL,
  folds = NULL,
  cores = 1L,
  weights = NULL
)
```

Arguments

fitfun	A function of one argument (a training data frame) returning a fitted underdisp model, e.g. <code>function(d) hurdle_cpb(y ~ x, d, participation = ~ z, fe = "unit", se = "none")</code> .
data	The full data frame.
k	Number of folds (default 5).
kmax	Highest count to evaluate; if NULL, taken from a fit on the full data (one extra fit).
folds	Optional integer vector of length <code>nrow(data)</code> giving a fixed fold assignment (so two models can be scored on identical folds). If NULL, folds are drawn at random.
cores	Worker processes for the fold refits (default 1); see cpb() .
weights	Optional frequency weights for the held-out scores: a column name of data or a numeric vector with one value per row. The mean scores are then weighted means, so a row of weight <code>w</code> counts as <code>w</code> held-out observations (the folds assign whole rows). The fold fits see the weights only through <code>fitfun</code> , so pass them there as well.

Value

An object of class "cv_score": a list with the mean held-out logscore and rps, the per-observation vectors `logscore_i/rps_i`, and the folds used.

See Also

[score\(\)](#)

Examples

```
set.seed(1)
d <- data.frame(y = rcpb(500, lambda = 3, alpha = 0.5))
cv <- cv_score(function(tr) cpb(y ~ 1, tr, truncated = FALSE, se = "none"), d, k = 5)
cv$logscore
```

dispersion_profile *Dispersion profile: variance-to-mean ratio against the fitted mean*

Description

Bins the observations by the fitted mean of the first model, computes the empirical conditional variance-to-mean ratio in each bin (the mean of the Pearson contributions $(y_i - \hat{\mu}_i)^2 / \hat{\mu}_i$), and sets it against the ratio each fitted family implies at those means. The families differ in how their dispersion moves with the mean: the CPB and the Katz family hold it essentially constant (at α and δ ; the implied curves are the exact moments of the fitted pmf, so the renormalized CPB's ratio dips a little below α where the ceiling is only two or three), the COM-Poisson, gamma-count, and double Poisson approach a constant only as the mean grows and rise toward the Poisson at small means, and the negative binomial's ratio rises linearly. The profile therefore shows which mechanism the data follow, where a family's variance function fails, and whether the observed underdispersion is confined to a range of means. Zero-truncated fits are profiled on their conditional moments. With frequency weights (those of the first model) the bins hold equal weight rather than equal numbers of rows, each row kept whole, every bin statistic is a weighted mean, and n is the bin's weight total, so a weighted fit is profiled as the data it stands for.

Usage

```
dispersion_profile(
  object,
  ...,
  bins = 10,
  plot = TRUE,
  ylim = NULL,
  main = "Dispersion profile",
  xlab = "Fitted mean",
  ylab = "Conditional Var / Mean"
)

## S3 method for class 'dispersion_profile'
plot(
  x,
  ylim = NULL,
  main = "Dispersion profile",
  xlab = "Fitted mean",
  ylab = "Conditional Var / Mean",
  ...
)
```

Arguments

object	A fitted single-equation model (cpb, cpb_fe, gec, gec_fe, or count_reg); its fitted means define the bins and the empirical ratio.
--------	--

...	Further fitted models on the same data, whose implied ratios are added as columns (name them for readable labels).
bins	Number of equal-frequency bins by fitted mean (default 10).
plot	Draw the profile (default TRUE); the table is returned invisibly either way.
ylim, main, xlab, ylab	Plot settings.
x	A "dispersion_profile" table.

Value

A data frame of class "dispersion_profile" with one row per bin: bin, n, mean_fitted, mean_y, ratio_empirical, and one ratio_<model> column per fitted model.

See Also

[dispersion_test\(\)](#), [compare_models\(\)](#), [rootogram\(\)](#)

Examples

```
set.seed(4); n <- 400; x <- rnorm(n)
d <- data.frame(y = rgammacount(n, exp(0.8 + 0.6 * x), alpha = 2.5), x = x)
m_gc <- count_reg(y ~ x, d, family = "gammacount")
m_nb <- count_reg(y ~ x, d, family = "negbin")
m_cp <- cpb(y ~ x, d, truncated = FALSE, se = "none")
dispersion_profile(gammacount = m_gc, negbin = m_nb, cpb = m_cp)
```

dispersion_test	<i>Regression-adjusted test of equidispersion</i>
-----------------	---

Description

Tests whether a fitted count model's dispersion departs from the Poisson, conditional on the co-variates. For a fit from a family with a dispersion parameter (method = "lr", the default), the test is the likelihood-ratio test of that parameter against its Poisson value, with the Poisson refit on the same design (fixed effects included); its p-value comes from the asymptotic distribution or from a parametric bootstrap under the fitted Poisson, by the rule in Details. For a Poisson fit, method = "auxiliary" runs the Cameron–Trivedi (1990) auxiliary regression, which needs no alternative family.

Usage

```
dispersion_test(
  object,
  alternative = c("two.sided", "under", "over"),
  method = c("lr", "auxiliary"),
  B = NULL,
  cores = 1L
)
```

Arguments

object	A fitted cpb, cpb_fe, gec, gec_fe, or count_reg model (for method = "auxiliary", a count_reg fit with family = "poisson").
alternative	"two.sided" (default where the null is interior), "under" (underdispersion), or "over" (overdispersion).
method	"lr" (likelihood ratio against the Poisson refit; default) or "auxiliary" (the Cameron–Trivedi auxiliary regression for a Poisson fit).
B	Parametric-bootstrap replicates for method = "lr": NULL (the default) applies the calibration rule in Details (199 replicates when the bootstrap is needed), 0 requests the asymptotic distribution, and a positive whole number forces the bootstrap.
cores	Worker processes for the bootstrap replicates.

Details

The Poisson is nested in every family at one value of the dispersion parameter: $\alpha = 1$ (CPB), $\delta = 1$ (GEC), $\nu = 1$ (both COM-Poisson parameterizations), $\lambda = 0$ (generalized Poisson), $\alpha = 1$ (gamma-count), $\theta = 1$ (double Poisson), and $1/\theta = 0$ (negative binomial). When that value is interior to the parameter space the likelihood-ratio statistic is asymptotically χ_1^2 under the null; a directional alternative ("under" or "over") uses the signed root $r = \pm\sqrt{LR}$, positive when the estimate lies on the underdispersed side of the Poisson value, which is asymptotically standard normal. When the Poisson value is on the boundary (the CPB, whose α lives in $(0, 1)$; the negative binomial, whose overdispersion parameter is non-negative) the asymptotic null distribution is the $\frac{1}{2}\chi_0^2 + \frac{1}{2}\chi_1^2$ mixture (Self and Liang 1987; Andrews 2001) and the test is one-sided by construction ("under" for the CPB, "over" for the negative binomial). The CPB qualifies although its support moves with its parameters: near $\alpha = 1$ its ceiling $\lambda / (1 - \alpha)$ diverges, the log-likelihood is regular on that side, and its score there is the classical dispersion score $-[(y - \lambda)^2 - y]/(2\lambda)$; the zero-truncated model reaches the same mixture through its efficient score. Because the Poisson is the CPB's limit as α approaches 1, the statistic cannot be negative in principle; when the max. support guard stops α short of that limit and the fitted log-likelihood lands just below the Poisson's, the statistic takes its boundary value 0. Any other negative statistic means the optimizer fell short, and no p-value is reported.

Calibration. Where the Poisson value is on the boundary of the family's parameter space (the CPB, the negative binomial), or the fit has unit fixed effects, the p-value is by default a parametric bootstrap with 199 replicates. The one-sided boundary statistic is skewed toward rejection in finite samples, and unit intercepts bias the dispersion estimate toward underdispersion by an amount of order $1/T$; in both cases the asymptotic distribution over-rejects. Where the Poisson value is interior, the asymptotic distribution is used unless the estimated mean parameters shift it too far: to first order they move the signed root toward underdispersion by $p/\sqrt{2n}$ standard deviations for p mean parameters on n observations (the weight total), the bias Dean and Lawless (1989) remove from the score test, and the bootstrap takes over when that shift pushes the first-order size of a 5\% simulates responses from the Poisson (zero-truncated Poisson) fitted to the same design, offset, weights, and unit effects, refits both models to each, and reports the share of simulated statistics at least as extreme as the observed one, counting the observed one, among the replicates whose two fits both succeeded. Simulating at the Poisson value itself keeps the bootstrap valid at the boundary. $B = 0$ requests the asymptotic distribution (with unit fixed effects it returns the statistic without a p-value), and a positive B sets the number of bootstrap replicates. Every replicate refits both models,

so the bootstrap takes about B times as long as the original fit; cores spreads the replicates over worker processes.

Value

An object of class `c("dispersion_test", "htest")` with the statistic, the p-value, the estimated dispersion parameter and its Poisson value, the two log-likelihoods, and the alternative; for method = "lr" also calibration ("asymptotic" or "parametric bootstrap"), B and B_ok (replicates requested and completed), boot (the simulated statistics), first_order_size (the asymptotic size), asymptotic_ok (whether the calibration rule admits the asymptotic distribution), and note where a statistic or p-value needs one.

References

Andrews, D. W. K. (2001). Testing when a parameter is on the boundary of the maintained hypothesis. *Econometrica*, 69(3), 683-734. Cameron, A. C. and Trivedi, P. K. (1990). Regression-based tests for overdispersion in the Poisson model. *Journal of Econometrics*, 46(3), 347-364. Dean, C. and Lawless, J. F. (1989). Tests for detecting overdispersion in Poisson regression models. *Journal of the American Statistical Association*, 84(406), 467-472. Self, S. G. and Liang, K.-Y. (1987). Asymptotic properties of maximum likelihood estimators and likelihood ratio tests under nonstandard conditions. *Journal of the American Statistical Association*, 82(398), 605-610.

See Also

`count_reg()`, `dispersion_profile()`, `zi_test()`

Examples

```
set.seed(2); n <- 400; x <- rnorm(n)
d <- data.frame(y = rgammacount(n, exp(1 + 0.4 * x), alpha = 2), x = x)
dispersion_test(count_reg(y ~ x, d, family = "gammacount"), alternative = "under")
dispersion_test(count_reg(y ~ x, d, family = "poisson"), method = "auxiliary")
```

doublepois-distribution

Double Poisson distribution functions

Description

Density, distribution, quantile, and random generation for Efron's (1986) double Poisson distribution with location μ and dispersion θ , normalized by the exact sum rather than Efron's closed-form approximation. $\theta > 1$ gives underdispersion, $\theta = 1$ is the Poisson, $\theta < 1$ overdispersion; the variance-to-mean ratio is approximately $1/\theta$, and the mean is close to (but not exactly) μ . The exact mean and variance are what `count_reg(family = "doublepois")` reports through `predict()` and `fitted()`.

Usage

```
ddoublepois(x, mu, theta, log = FALSE)

pdoublepois(q, mu, theta, lower.tail = TRUE, log.p = FALSE)

qdoublepois(p, mu, theta, lower.tail = TRUE, log.p = FALSE)

rdoublepois(n, mu, theta)
```

Arguments

x, q	Vector of quantiles (non-negative integers).
mu	Location parameter (scalar or vector, recycled).
theta	Dispersion parameter (scalar, positive).
log, log.p	Return log probabilities.
lower.tail	If TRUE (default), $P(X \leq x)$.
p	Vector of probabilities.
n	Number of draws.

Value

ddoublepois a density, pdoublepois a CDF, qdoublepois a quantile, rdoublepois a numeric vector of count draws.

References

Efron, B. (1986). Double exponential families and their use in generalized linear regression. *Journal of the American Statistical Association*, 81(395), 709-721.

See Also

[count_reg\(\)](#)

Examples

```
ddoublepois(0:5, mu = 3, theta = 2)
sum(ddoublepois(0:60, mu = 3, theta = 2))
```

first_difference	<i>First difference for a CPB fit</i>
------------------	---------------------------------------

Description

The effect on a quantity of interest of moving one covariate from one value to another, holding the other covariates at their means, with a percentile interval from the model's bootstrap draws when the fit carries them.

Usage

```
first_difference(object, ...)

## S3 method for class 'cpb'
first_difference(
  object,
  variable,
  from,
  to,
  quantity = c("mean", "ceiling", "prob"),
  y = NULL,
  level = 0.95,
  ...
)
```

Arguments

object	A "cpb" object. With se = "bootstrap" at fit time the difference carries a bootstrap percentile interval; otherwise the point estimate is returned with method = "none".
...	Further arguments passed to methods; unknown arguments error.
variable	Name of a model-matrix column to vary.
from, to	The two values of variable to contrast.
quantity	"mean" (E(Y); for a zero-truncated fit the conditional mean $E(Y \mid Y \geq 1)$), "ceiling" ($\lambda/(1-\alpha)$), or "prob" ($P(Y = y)$).
y	The count value for quantity = "prob".
level	Confidence level (default 0.95).

Value

A "ud_fd" data frame – the package-wide first-difference contract (columns component, from, to, diff, lower, upper, method) – with one row for the requested quantity and a bootstrap percentile interval on the difference (method = "bootstrap (stored)"). Every first_difference() method in the package returns this same shape.

Examples

```
set.seed(1); x <- rnorm(250)
N <- pmax(round(exp(1.6 + 0.5 * x) / 0.5), 1); y <- rbinom(250, N, 0.5)
fit <- cpb(y ~ x, data = data.frame(y = y, x = x)[y > 0, ], se = "bootstrap", B = 60)
first_difference(fit, "x", from = -1, to = 1, quantity = "mean")
```

first_difference.pec *First differences for the matched count families*

Description

The discrete-change effect on the expected count $E(Y)$ as variable moves from *from* to *to*, holding the other covariates at their sample means. For *count_reg* this is a single number with a delta-method interval; for the two-part models it is decomposed exactly into extensive (participation / non-structural-zero) and intensive (count) channels that sum to the total.

Usage

```
## S3 method for class 'pec'
first_difference(object, variable, from, to, level = 0.95, ...)

## S3 method for class 'pec_fe'
first_difference(object, variable, from, to, level = 0.95, ...)

## S3 method for class 'cpb_fe'
first_difference(object, variable, from, to, level = 0.95, ...)

## S3 method for class 'hurdle_pec'
first_difference(
  object,
  variable,
  from,
  to,
  level = 0.95,
  stage = c("both", "participation", "intensity", "zero", "count"),
  ...
)

## S3 method for class 'zi_pec'
first_difference(
  object,
  variable,
  from,
  to,
  level = 0.95,
  stage = c("both", "participation", "intensity", "zero", "count"),
```

```

    ...
)

## S3 method for class 'count_reg'
first_difference(object, variable, from, to, level = 0.95, ...)

## S3 method for class 'hurdle_count'
first_difference(
  object,
  variable,
  from,
  to,
  level = 0.95,
  stage = c("both", "participation", "intensity", "zero", "count"),
  ...
)

## S3 method for class 'zi_count'
first_difference(
  object,
  variable,
  from,
  to,
  level = 0.95,
  stage = c("both", "participation", "intensity", "zero", "count"),
  ...
)

```

Arguments

object	A fitted count_reg, gec, gec_fe, cpb_fe, hurdle_count, zi_count, hurdle_gec, or zi_gec model.
variable	Name of the covariate to change (must be in the model).
from, to	The two values of variable.
level	Confidence level for the delta-method intervals (all methods on this page that can compute one).
...	Unused; unknown arguments error.
stage	For the two-part models, which equation(s) the change is applied to. "both" (default) moves the variable wherever it appears; "intensity"/"count" or "participation"/"zero" moves it in only that equation, holding it at the reference in the other. For a variable in only one equation all options coincide; for a variable in both, "both" gives the total effect and the single-stage options the partial effect through that margin. All component rows are always returned.

Value

A "ud_fd" data frame – the package-wide first-difference contract shared by every first_difference() method: columns component, from, to, diff, lower, upper, method. Single-equation fits return

one row (component = "mean"); two-part fits return one row per margin level (binary-stage probability, count-stage mean, marginal mean). method records the uncertainty source per row ("delta", a bootstrap label, or "none" with NA bounds).

first_difference.hurdle_cpb

Extensive/intensive first-difference decomposition for a hurdle- or ZI-CPB

Description

The effect of moving variable from one value to another, decomposed into the change in the participation (extensive) margin, the intensity (intensive) margin, and the marginal expected count, holding other covariates at their means. This is the quantity the single-equation defaults cannot separate.

Usage

```
## S3 method for class 'hurdle_cpb'
first_difference(
  object,
  variable,
  from,
  to,
  B = 0,
  level = 0.95,
  data = NULL,
  stage = c("both", "participation", "intensity", "zero", "count"),
  cores = 1L,
  ...
)

## S3 method for class 'zi_cpb'
first_difference(
  object,
  variable,
  from,
  to,
  B = 0,
  level = 0.95,
  data = NULL,
  stage = c("both", "participation", "intensity", "zero", "count"),
  cores = 1L,
  ...
)
```

Arguments

object	A "hurdle_cpb" or "zi_cpb" object.
variable	Name of a model-matrix column to vary (in either margin).
from, to	The two values to contrast.
B	Bootstrap replicates for percentile intervals; 0 (default) returns point estimates only. For zi_cpb the bootstrap is slow (each replicate refits the mixture).
level	Confidence level.
data	The original data frame; required when B > 0.
stage	Which equation(s) the covariate moves in: "both" (default), the binary stage only ("participation"/"zero"), or the count stage only ("intensity"/"count"), holding the covariate at its reference in the other equation. All three component rows are always returned.
cores	Worker processes for the bootstrap refits (default 1); see cpb() .
...	Unused; unknown arguments error.

Value

A "ud_fd" data frame (the package-wide first-difference contract): columns component, from, to, diff, lower, upper, method, one row per margin level.

fitted.underdisp	<i>Fitted means for the two-part and fixed-effects classes</i>
------------------	--

Description

Completes [stats::fitted\(\)](#) across the family, returning the same quantity as `predict(object, type = "response")` on the estimation data: the mean for the single-equation fits and the marginal expected count (zeros included) for the two-part fits. `gec_fe` inherits `fitted.gec`.

Usage

```
## S3 method for class 'cpb_fe'
fitted(object, ...)

## S3 method for class 'hurdle_cpb'
fitted(object, ...)

## S3 method for class 'hurdle_gec'
fitted(object, ...)

## S3 method for class 'hurdle_count'
fitted(object, ...)

## S3 method for class 'zi_cpb'
```

```
fitted(object, ...)

## S3 method for class 'zi_gec'
fitted(object, ...)

## S3 method for class 'zi_count'
fitted(object, ...)
```

Arguments

<code>object</code>	A fitted model from this package.
<code>...</code>	Unused.

Value

A numeric vector, one element per estimation observation.

gammacount-distribution

Gamma-count distribution functions

Description

Density, distribution, quantile, and random generation for Winkelmann's (1995) gamma-count distribution: the number of events in a unit interval when the waiting times between events are independent Gamma variables with shape α and rate $\alpha * \mu$, so that μ is the long-run event rate. $\alpha > 1$ (waiting times more regular than exponential) gives underdispersion, $\alpha = 1$ is the Poisson, $\alpha < 1$ overdispersion; the variance-to-mean ratio is approximately $1/\alpha$. The exact mean and variance are finite sums of incomplete-gamma terms and are what `count_reg(family = "gammacount")` reports through `predict()` and `fitted()`.

Usage

```
dgammacount(x, mu, alpha, log = FALSE)

pgammacount(q, mu, alpha, lower.tail = TRUE, log.p = FALSE)

qgammacount(p, mu, alpha, lower.tail = TRUE, log.p = FALSE)

rgammacount(n, mu, alpha)
```

Arguments

<code>x, q</code>	Vector of quantiles (non-negative integers).
<code>mu</code>	Rate parameter (scalar or vector, recycled).
<code>alpha</code>	Dispersion parameter (scalar, positive).

log, log.p	Return log probabilities.
lower.tail	If TRUE (default), $P(X \leq x)$.
p	Vector of probabilities.
n	Number of draws.

Value

dgammaaccount a density, pgammaaccount a CDF, qgammaaccount a quantile, rgammaaccount a numeric vector of count draws.

References

Winkelmann, R. (1995). Duration dependence and dispersion in count-data models. *Journal of Business & Economic Statistics*, 13(4), 467-474.

See Also

[count_reg\(\)](#)

Examples

```
dgammaaccount(0:5, mu = 3, alpha = 2)
var(rgammaaccount(2000, mu = 3, alpha = 2)) / 3
```

gec

Generalized event count (Katz-family) regression

Description

Fits King's generalized event count model, a Katz-family count regression whose single dispersion parameter δ (the variance-to-mean ratio) is estimated freely and spans underdispersion ($\delta < 1$, a finite-support member; the continuous parameter binomial is this cell), equidispersion ($\delta = 1$, Poisson), and overdispersion ($\delta > 1$, the negative binomial). Unlike [cpb\(\)](#), which fixes the direction of dispersion to under, `gec()` lets the data choose. The Katz recursion delivers exact first and second moments—the Winkelmann–Signorino–King correction realized directly on an unbounded support—and the likelihood is evaluated in C++. For $\delta < 1$ the support is finite and the renormalized distribution's mean and variance equal $\exp(x'b)$ and $\delta * \exp(x'b)$ only when $\exp(x'b)/(1 - \delta)$ is an integer, so $\exp(x'b)$ is the rate parameter of the recursion, the fitted mean is computed exactly from the pmf, and δ is the Katz dispersion parameter (the variance-to-mean ratio on an unbounded support). The optimizer works on covariates scaled to unit standard deviation and maps the coefficients back, so the fit does not depend on the covariates' units.

Usage

```

gec(
  formula,
  data,
  truncated = FALSE,
  se = c("none", "bootstrap"),
  B = 500,
  cluster = NULL,
  offset = NULL,
  weights = NULL,
  cores = 1L,
  max.support = 500,
  maxit = 20000,
  reltol = 1e-08
)

```

Arguments

<code>formula</code>	A model formula.
<code>data</code>	A data frame.
<code>truncated</code>	Logical; if TRUE, fit the zero-truncated GEC (all $Y \geq 1$), the intensity model of hurdle_gec() .
<code>se</code>	Coefficient inference: "none" (default; fast) or "bootstrap".
<code>B</code>	Bootstrap resamples when <code>se = "bootstrap"</code> .
<code>cluster</code>	Optional cluster identifier (a column name in data or a vector) for a cluster/block bootstrap; see cpb() .
<code>offset</code>	Optional offset on the log-mean scale (an exposure): a numeric vector or the name of a column in data.
<code>weights</code>	Optional frequency weights (a numeric vector or a column name); see cpb() .
<code>cores</code>	Worker processes for the bootstrap; see cpb() .
<code>max.support</code>	Guard on the maximum evaluated support.
<code>maxit, reltol</code>	Optimizer controls.

Value

An object of class "gec" with coefficients, delta (the estimated dispersion), loglik, bootstrap standard errors, and bookkeeping.

See Also

[cpb\(\)](#), [compare_dispersion\(\)](#), [dispersion_test\(\)](#)

Examples

```

set.seed(1); x <- rnorm(400)
y <- rpois(400, exp(1 + 0.5 * x))
gec(y ~ x, data = data.frame(y = y, x = x), se = "none")

```

gec-distribution *Generalized event count (Katz) distribution functions*

Description

Density, distribution, quantile, and random generation for the King (1989) generalized event count model with rate λ and Katz dispersion δ ($\delta < 1$ underdispersed, $\delta = 1$ Poisson, $\delta > 1$ overdispersed). On an unbounded support ($\delta \geq 1$) the mean is λ and the variance-to-mean ratio is δ exactly; for $\delta < 1$ the support is finite and the renormalized distribution's mean and variance equal those values only when $\lambda/(1 - \delta)$ is an integer (the exact moments are finite sums of the pmf, as `count_reg()`'s siblings report them). Consistent with the estimator `gec()`.

Usage

```
dgec(x, lambda, delta, max.support = 500, log = FALSE)

pgec(q, lambda, delta, max.support = 500, lower.tail = TRUE, log.p = FALSE)

qgec(p, lambda, delta, max.support = 500, lower.tail = TRUE, log.p = FALSE)

rgec(n, lambda, delta, max.support = 500)
```

Arguments

<code>x, q</code>	Vector of quantiles (non-negative integers).
<code>lambda</code>	Rate parameter (scalar or vector, recycled).
<code>delta</code>	Katz dispersion parameter (scalar).
<code>max.support</code>	Guard on the evaluated support.
<code>log, log.p</code>	Return log probabilities.
<code>lower.tail</code>	If TRUE (default), $P(X \leq x)$.
<code>p</code>	Vector of probabilities.
<code>n</code>	Number of draws.

Value

`dgec` a density, `pgec` a CDF, `qgec` a quantile, `rgec` a numeric vector of count draws.

See Also

`gec()`

Examples

```
dgec(0:5, lambda = 3, delta = 0.7)
var(rgec(2000, lambda = 3, delta = 0.7)) / mean(rgec(2000, lambda = 3, delta = 0.7))
```

gec_fe

*GEC (Katz-family) regression with high-dimensional unit fixed effects***Description**

Fits `gec()` with a full set of unit fixed effects, concentrating (profiling) out the unit intercepts by a one-dimensional inner maximization per unit, so the outer optimizer handles only the covariate coefficients and the free dispersion parameter `delta`. This is the `cpb_fe()` concentration generalized to the whole Katz family: the panel can be under-, equi-, or overdispersed and the direction is estimated, not presumed. For `delta < 1` the Katz support is finite, but the likelihood is continuous across an integer ceiling (the entering support point's mass grows from zero) and only kinks there, so each unit's objective is unimodal in its intercept and is maximized by golden section on a bracket around the unit's Poisson intercept, expanded while the maximum sits at an edge; a maximum at a kink is tracked in the analytic gradient of the concentrated likelihood, which the outer BFGS uses. The feasibility floor is exact: every count must lie in $0 \dots \text{ceiling}(\mu/(1-\text{delta}))$. Runtime: about twice that of `cpb_fe()` on the same panel (the Katz recursion runs the whole support), so a 2,600-row panel in 146 units takes about two minutes.

Usage

```
gec_fe(
  formula,
  data,
  fe,
  se = c("none", "bootstrap"),
  B = 500,
  cluster = NULL,
  offset = NULL,
  weights = NULL,
  cores = 1L,
  max.support = NULL,
  inner_it = 30L,
  maxit = 3000L,
  reltol = 1e-07,
  bias_correct = c("none", "jackknife")
)
```

Arguments

<code>formula</code>	A model formula for the covariates only (no unit factor, no intercept; the fixed effects absorb it).
<code>data</code>	A data frame.
<code>fe</code>	Name of the column holding the unit identifier.
<code>se</code>	Inference for the covariate coefficients: "none" (default) or "bootstrap", a pairs/cluster bootstrap over units.
<code>B</code>	Bootstrap resamples when <code>se = "bootstrap"</code> .

cluster	Cluster for the bootstrap; NULL (default) resamples the fixed-effects units. See cpb_fe() .
offset	Optional offset on the log-mean scale (an exposure): a numeric vector or the name of a column in data.
weights	Optional frequency weights (a numeric vector or a column name); see cpb() .
cores	Worker processes for the bootstrap; see cpb() .
max.support	Guard on the maximum evaluated support.
inner_it	Golden-section iterations for each unit's inner maximization.
maxit, reltol	Outer optimizer controls.
bias_correct	"none" (default) or "jackknife", the split-panel jackknife correction for the 1/T incidental-parameters bias (see Details).

Details

Limitation: unlike [cpb_fe\(\)](#), [gec_fe\(\)](#) has no truncated argument – a concentrated zero-truncated GEC is not currently implemented. For a zero-truncated GEC with fixed effects, enter the unit factor as dummies in [gec\(\)](#)'s formula (feasible for moderate unit counts).

Short panels: delta and the fixed effects carry the incidental-parameters bias of nonlinear fixed-effects estimation, of order 1/T; treat delta cautiously below about $T = 30$. The covariate coefficients are not materially affected. `bias_correct = "jackknife"` removes the leading 1/T term by the split-panel jackknife exactly as in [cpb_fe\(\)](#): refit on each unit's temporal halves and report $2 * \text{full} - \text{mean}(\text{halves})$, with the unit effects and fitted values re-concentrated at the corrected parameters and logLik/AIC kept at the maximum-likelihood fit (uncorrected estimates in `$uncorrected`). The same time-homogeneity validity gate applies: on panels where the two halves do not estimate a common parameter the correction is REFUSED with a warning and the maximum-likelihood fit is returned (see [cpb_fe\(\)](#), Details).

Value

An object of class `c("gec_fe", "gec")`.

See Also

[gec\(\)](#), [cpb_fe\(\)](#)

Examples

```
set.seed(5)
d <- do.call(rbind, lapply(1:30, function(i) {
  x <- rnorm(10); N <- pmax(round(exp(1 + rnorm(1, 0, 0.4) + 0.3 * x) / 0.5), 1)
  data.frame(unit = i, x = x, y = rbinom(10, N, 0.5))
}))
gec_fe(y ~ x, data = d, fe = "unit") # delta ~ 0.5: underdispersed
```

genpois-distribution *Generalized Poisson distribution functions*

Description

Density, distribution, quantile, and random generation for the Consul-Jain generalized Poisson distribution in its constant-dispersion form: mean μ and dispersion λ in $(-1, 1)$, with $P(Y = y) = \theta(\theta + \lambda y)^{y-1} e^{-\theta - \lambda y} / y!$ and $\theta = \mu(1 - \lambda)$, so that the variance-to-mean ratio is $1/(1 - \lambda)^2$. $\lambda < 0$ gives underdispersion, $\lambda = 0$ the Poisson, $\lambda > 0$ overdispersion. For $\lambda < 0$ the support is finite (the terms are positive only while $\theta + \lambda y > 0$) and the pmf is renormalized on that support, so it is a proper distribution and the exact mean and variance (which then differ from μ and $\mu/(1-\lambda)^2$ only by the negligible mass beyond the support) are what `count_reg(family = "genpois")` reports through `predict()` and `fitted()`.

Usage

```
dgenpois(x, mu, lambda, log = FALSE)

pgenpois(q, mu, lambda, lower.tail = TRUE, log.p = FALSE)

qgenpois(p, mu, lambda, lower.tail = TRUE, log.p = FALSE)

rgenpois(n, mu, lambda)
```

Arguments

<code>x, q</code>	Vector of quantiles (non-negative integers).
<code>mu</code>	Mean parameter (scalar or vector, recycled).
<code>lambda</code>	Dispersion parameter in $(-1, 1)$ (scalar).
<code>log, log.p</code>	Return log probabilities.
<code>lower.tail</code>	If TRUE (default), $P(X \leq x)$.
<code>p</code>	Vector of probabilities.
<code>n</code>	Number of draws.

Value

`dgenpois` a density, `pgenpois` a CDF, `qgenpois` a quantile, `rgenpois` a numeric vector of count draws.

References

Consul, P. C. and Jain, G. C. (1973). A generalization of the Poisson distribution. *Technometrics*, 15(4), 791-799. Consul, P. C. and Famoye, F. (2006). *Lagrangian Probability Distributions*. Birkhauser.

See Also[count_reg\(\)](#)**Examples**

```
dgenpois(0:5, mu = 3, lambda = -0.3)
var(rngenpois(2000, mu = 3, lambda = -0.3)) / 3    # about 1/(1.3)^2
```

glance.cpb

*Glance at a CPB fit (broom method)***Description**

Glance at a CPB fit (broom method)

Usage

```
## S3 method for class 'cpb'
glance(x, ...)

## S3 method for class 'cpb_fe'
glance(x, ...)

## S3 method for class 'hurdle_cpb'
glance(x, ...)

## S3 method for class 'zi_cpb'
glance(x, ...)

## S3 method for class 'gec'
glance(x, ...)

## S3 method for class 'hurdle_gec'
glance(x, ...)

## S3 method for class 'zi_gec'
glance(x, ...)

## S3 method for class 'count_reg'
glance(x, ...)

## S3 method for class 'zi_count'
glance(x, ...)

## S3 method for class 'hurdle_count'
glance(x, ...)
```

Arguments

x A "cpb" object.
... Unused.

Value

A one-row data frame of fit statistics.

hurdle_count	<i>Hurdle count regression for the matched count families</i>
--------------	---

Description

Fits a participation model joined to a zero-truncated intensity from any of the package's count families (see `count_reg()`, Families), the count analogue of `hurdle_cpb()`. Returned as a "hurdle_count" object that `compare_models()` and `score()` accept, so a hurdle-CPB and a hurdle gamma-count can be compared on one footing.

Usage

```
hurdle_count(  
  formula,  
  data,  
  family = c("poisson", "negbin", "compois", "mpcmp", "genpois", "gammacount",  
             "doublepois"),  
  participation = NULL,  
  fe = NULL,  
  part_fe = NULL,  
  link = c("logit", "probit", "cloglog"),  
  offset = NULL,  
  weights = NULL,  
  se = c("analytic", "robust", "cluster", "none"),  
  cluster = NULL  
)
```

Arguments

formula Intensity formula ($y \sim x$).
data A data frame.
family The intensity family; see `count_reg()`.
participation Optional one-sided formula for the participation model; defaults to the intensity right-hand side.
fe, part_fe Optional fixed-effects column names for the intensity and participation models (entered as factor dummies).

link	Link for the participation model: "logit" (default), "probit", or "cloglog". Positive participation coefficients raise the probability of a <i>positive count</i> (participation).
offset	Optional offset for the intensity, on the linear-predictor (log) scale: a numeric vector or the name of a column in data.
weights	Optional frequency weights (a numeric vector or a column name), applied to both margins; see count_reg() .
se, cluster	Standard-error type and optional cluster for the intensity; see count_reg() .

Value

An object of class "hurdle_count".

See Also

[hurdle_cpb\(\)](#), [zi_count\(\)](#), [compare_models\(\)](#)

Examples

```
set.seed(1); n <- 300; x <- rnorm(n); z <- rnorm(n)
y <- ifelse(rbinom(n, 1, plogis(0.4 + 0.8 * z)) == 1, rpois(n, exp(1 + 0.3 * x)) + 1L, 0L)
hurdle_count(y ~ x, data.frame(y = y, x = x, z = z), family = "poisson",
             participation = ~ z, se = "none")
```

hurdle_cpb

*Hurdle continuous parameter binomial regression***Description**

Fits a participation (hurdle) model joined to a zero-truncated CPB for the positive counts. Because the hurdle log-likelihood factorizes, the two parts are fit separately: a logistic regression of participation on all units, and a zero-truncated CPB ([cpb\(\)](#)) on the positive counts. The result is the natural model for a bounded, underdispersed participation process — most units at zero, participants carrying a tight count.

Usage

```
hurdle_cpb(
  formula,
  data,
  participation = NULL,
  fe = NULL,
  part_fe = NULL,
  link = c("logit", "probit", "cloglog"),
  offset = NULL,
  cluster = NULL,
  se = c("none", "bootstrap"),
```

```

    B = 500,
    weights = NULL,
    cores = 1L,
    ...
  )

```

Arguments

<code>formula</code>	Intensity model formula ($y \sim x$).
<code>data</code>	A data frame.
<code>participation</code>	Optional one-sided formula ($\sim z$) for the participation (hurdle) model; defaults to the intensity model's right-hand side.
<code>fe</code>	Optional column name for unit fixed effects in the intensity model, absorbed by a concentrated likelihood (<code>cpb_fe()</code>). This is how the within-unit underdispersion is recovered.
<code>part_fe</code>	Optional column name for fixed effects in the participation model (added as factors). Units with no within-unit variation in participation are uninformative for a fixed-effects logit.
<code>link</code>	Link for the participation model: "logit" (default), "probit", or "cloglog". Positive participation coefficients raise the probability of a positive count.
<code>offset</code>	Optional offset on the log-mean scale for the intensity (an exposure): a numeric vector or the name of a column in data.
<code>cluster</code>	Optional cluster identifier (a column name in data or a vector) for cluster-robust bootstrap inference on the intensity coefficients; see <code>cpb()</code> . Cluster-robust errors for the participation logit are available directly via <code>sandwich::vcovCL(fit\$participation, cluster = ...)</code> .
<code>se</code>	Inference for the intensity coefficients: "none" or "bootstrap".
<code>B</code>	Bootstrap replicates when <code>se = "bootstrap"</code> .
<code>weights</code>	Optional frequency weights (a numeric vector or a column name), applied to both margins; see <code>cpb()</code> .
<code>cores</code>	Worker processes for the intensity bootstrap; see <code>cpb()</code> .
<code>...</code>	Passed to <code>cpb()</code> or <code>cpb_fe()</code> .

Value

An object of class "hurdle_cpb" with elements `participation` (a `glm`), `intensity` (a `cpb` or `cpb_fe`), and bookkeeping.

Examples

```

set.seed(1)
n <- 800; x <- rnorm(n); z <- rnorm(n)
y <- rhurdle_cpb(n, lambda = exp(1.3 + 0.5 * x), alpha = 0.5,
               p = plogis(-0.2 + 0.8 * z))
fit <- hurdle_cpb(y ~ x, data = data.frame(y = y, x = x, z = z),
                 participation = ~ z)
fit

```

hurdle_gec

*Hurdle GEC (Katz-family) regression***Description**

Joins a participation (hurdle) logit to a zero-truncated GEC intensity on the positive counts. Unlike [hurdle_cpb\(\)](#), whose intensity is fixed to the underdispersed CPB, the GEC intensity estimates its own dispersion direction.

Usage

```
hurdle_gec(
  formula,
  data,
  participation = NULL,
  part_fe = NULL,
  link = c("logit", "probit", "cloglog"),
  offset = NULL,
  cluster = NULL,
  se = c("none", "bootstrap"),
  B = 500,
  weights = NULL,
  cores = 1L,
  max.support = 500
)
```

Arguments

formula	Intensity formula, $y \sim x$.
data	A data frame.
participation	One-sided formula for the participation logit; defaults to the intensity's right-hand side.
part_fe	Optional column name for fixed effects in the participation model (added as factor dummies).
link	Link for the participation model: "logit" (default), "probit", or "cloglog".
offset	Optional offset (log scale) for the intensity: a numeric vector or a column name in data.
cluster	Optional cluster (column name or vector) for the intensity's cluster bootstrap.
se	Intensity inference: "none" (default) or "bootstrap".
B	Bootstrap resamples.
weights	Optional frequency weights (a numeric vector or a column name), applied to both margins; see cpb() .
cores	Worker processes for the intensity bootstrap; see cpb() .
max.support	Guard on the maximum evaluated support.

Value

An object of class "hurdle_gec".

Limitation

Unlike [hurdle_cpb\(\)](#), [hurdle_gec\(\)](#) has no `fe` argument for a concentrated fixed-effects intensity (no concentrated zero-truncated GEC is implemented). For within-unit intensities, enter the unit factor as dummies in formula (feasible for moderate unit counts), or use [hurdle_cpb\(\)](#).

See Also

[hurdle_cpb\(\)](#), [gec\(\)](#), [zi_gec\(\)](#)

Examples

```
set.seed(3); n <- 300; x <- rnorm(n); z <- rnorm(n)
y <- ifelse(rbinom(n, 1, plogis(0.3 + 0.8 * z)) == 1, rpois(n, exp(1 + 0.3 * x)) + 1L, 0L)
hurdle_gec(y ~ x, data.frame(y = y, x = x, z = z), participation = ~ z)
```

implied_ceiling

Implied ceiling with a profile-likelihood interval

Description

Returns the observation- (or profile-) specific ceiling $\lambda/(1-\alpha)$, with bounds that carry the limits of the interval for α (see [alpha_confint\(\)](#); calibrated when the fit went through [calibrate_alpha\(\)](#)) to the ceiling at the fitted rate. The rate is held at its estimate: coefficient uncertainty in λ is not propagated, so the bounds are not a confidence interval for the ceiling; use [first_difference\(\)](#) with `quantity = "ceiling"` for a fully bootstrapped contrast.

Usage

```
implied_ceiling(object, ...)

## S3 method for class 'cpb'
implied_ceiling(object, newdata = NULL, level = 0.95, ...)

## S3 method for class 'cpb_fe'
implied_ceiling(object, newdata = NULL, level = 0.95, ...)
```

Arguments

<code>object</code>	A "cpb" object.
<code>...</code>	Further arguments passed to methods.
<code>newdata</code>	Optional covariate profiles.
<code>level</code>	Confidence level (default 0.95).

Value

A data frame with lambda, ceiling, and lower/upper bounds.

Examples

```
set.seed(6); x <- rnorm(300)
N <- pmax(round(exp(1.5 + 0.4 * x) / 0.5), 1); y <- rbinom(300, N, 0.5)
fit <- cpb(y ~ x, data.frame(y = y, x = x)[y > 0, ], se = "none")
implied_ceiling(fit, newdata = data.frame(x = c(-1, 0, 1)))
```

irr

*Incidence rate ratios for a CPB fit***Description**

Incidence rate ratios for a CPB fit

Usage

```
irr(object, ...)

## S3 method for class 'cpb'
irr(object, level = 0.95, ...)
```

Arguments

object	A "cpb" object. With se = "bootstrap" at fit time the ratios carry bootstrap percentile intervals; otherwise point estimates are returned with method = "none", as for every other irr() method.
...	Further arguments passed to methods.
level	Confidence level (default 0.95).

Value

A "ud_irr" data frame – the package-wide rate-ratio contract (columns term, equation, ratio, estimate, lower, upper, method) shared by every irr() method; here with bootstrap percentile intervals from the stored draws.

Examples

```
set.seed(8); x <- rnorm(200)
N <- pmax(round(exp(1.5 + 0.4 * x) / 0.5), 1); y <- rbinom(200, N, 0.5)
fit <- cpb(y ~ x, data.frame(y = y, x = x)[y > 0, ], se = "bootstrap", B = 40)
irr(fit)
```

irr.count_reg	<i>Rate and odds ratios for the matched count families</i>
---------------	--

Description

Rate ratios (exp of the count/intensity coefficients, column IRR) and, for the two-part models, odds ratios of the binary stage (column OR), by equation. Element names match the CPB family (intensity for the count component; participation for the hurdle stage, zero for the inflation stage).

Usage

```
## S3 method for class 'count_reg'
irr(object, level = 0.95, ...)

## S3 method for class 'gec'
irr(object, level = 0.95, ...)

## S3 method for class 'gec_fe'
irr(object, level = 0.95, ...)

## S3 method for class 'cpb_fe'
irr(object, level = 0.95, ...)

## S3 method for class 'hurdle_gec'
irr(object, level = 0.95, ...)

## S3 method for class 'zi_gec'
irr(object, level = 0.95, ...)

## S3 method for class 'hurdle_count'
irr(object, level = 0.95, ...)

## S3 method for class 'zi_count'
irr(object, level = 0.95, ...)
```

Arguments

object	A fitted model from this package.
level	Confidence level.
...	Unused.

Value

A "ud_irr" data frame – the package-wide ratio contract shared by every irr() method: columns term, equation ("count", "participation" for a hurdle's binary stage, or "inflation" for a zero-inflated model's structural-zero stage; the two point in opposite directions, an odds ratio

above one raising $P(Y > 0)$ in the first and $P(\text{structural zero})$ in the second), `ratio` ("IRR" or "OR"), `estimate`, `lower`, `upper`, and `method` (the interval source; "none" with NA bounds when no covariance is available). Two-part models stack both equations. The IRR rows are ratios of the rate parameter $\exp(\beta)$; for the CPB and the underdispersed GEC the ratio of fitted means differs from it by the support renormalization, and `first_difference()` reports the exact mean contrast.

mundlak

Mundlak (correlated random effects) device

Description

Augments a data frame with the unit-level means of the time-varying numeric covariates in `formula`, and returns the augmented formula and data. Fitting any of the package's estimators on the result implements the Mundlak / correlated-random-effects specification: the coefficients on the original covariates recover the within-unit (fixed-effects-consistent) effects, while the coefficients on the unit means capture (and test) the correlation between the covariates and the unit effect. It is a lighter, between-variation-preserving alternative to full fixed effects, and it is model-agnostic — the same device feeds `cpb()`, `gec()`, or `count_reg()`.

Usage

```
mundlak(formula, data, unit, suffix = "_mean")
```

Arguments

<code>formula</code>	A model formula.
<code>data</code>	A data frame.
<code>unit</code>	Column name identifying the panel unit.
<code>suffix</code>	Suffix for the added unit-mean columns (default "_mean").

Value

A list with `formula` (the augmented formula), `data` (the augmented data frame), and `added` (the names of the unit-mean columns).

See Also

[count_reg\(\)](#), [cpb\(\)](#)

Examples

```
set.seed(1)
d <- data.frame(y = rpois(200, 3), x = rnorm(200), unit = factor(rep(1:20, each = 10)))
m <- mundlak(y ~ x, d, unit = "unit")
fit <- count_reg(m$formula, m$data, family = "poisson")
```

 peacekeeping

UN peacekeeping contributions, state-years

Description

A state-year panel of contributions to United Nations peacekeeping operations, the archetype of a bounded, underdispersed participation count: most state-years contribute to no operation, and the states that contribute carry a tight count held in a narrow band by a turning-over roster. About 42\hurdle-CPB.

Usage

```
data(peacekeeping)
```

Format

A data frame with 4,448 rows and 9 variables:

iso3 ISO3 country code (the panel unit).

year Calendar year, 1990–2016.

contributions Number of distinct UN peacekeeping operations the state contributes personnel to in that year (the count outcome).

democracy Electoral-democracy index.

lgdppc Log GDP per capita.

lpop Log population.

milper Military personnel, thousands.

majorpower Major-power indicator (0/1).

region World region.

Source

Contribution counts aggregated from the International Peace Institute Providing for Peacekeeping database (<https://www.providingforpeacekeeping.org>); covariates from V-Dem and the Correlates of War National Material Capabilities data. Counts and public covariates only; prepared for illustration.

Examples

```
data(peacekeeping)
table(peacekeeping$contributions == 0)

some <- subset(peacekeeping, iso3 %in% unique(iso3)[1:40]) # forty states keep the example short
fit <- hurdle_cpb(contributions ~ lgdppc + milper, data = some,
                  participation = ~ democracy + majorpower, fe = "iso3")
fit
```

pit_hist

*Non-randomized PIT histogram for a fitted count model***Description**

Draws the non-randomized probability integral transform histogram of Czado, Gneiting, and Held (2009). A well-calibrated model yields a flat histogram at height one (the reference line); a U shape indicates under-dispersion in the predictive distribution and a hump indicates over-dispersion.

Usage

```
pit_hist(
  fit,
  bins = 10,
  main = "PIT histogram",
  xlab = "PIT",
  ylab = "Relative frequency",
  ...
)
```

Arguments

<code>fit</code>	A fitted underdisp count model (any class <code>score()</code> takes); a fit with frequency weights counts each row as many times as its weight.
<code>bins</code>	Number of histogram bins.
<code>main, xlab, ylab</code>	Plot labels.
<code>...</code>	Passed to <code>graphics::barplot()</code> .

Value

Invisibly, the vector of bin heights (normalized so that a calibrated model gives heights near one).

Examples

```
set.seed(1)
y <- rcpb(400, lambda = 3, alpha = 0.5)
fit <- cpb(y ~ 1, data = data.frame(y = y), truncated = FALSE, se = "none")
pit_hist(fit)
```

predict.count_reg	<i>Predictions from a matched count-family fit</i>
-------------------	--

Description

Predictions from a matched count-family fit

Usage

```
## S3 method for class 'count_reg'
predict(
  object,
  newdata = NULL,
  type = c("response", "link", "prob"),
  at = NULL,
  offset = NULL,
  ...
)
```

Arguments

object	A "count_reg" object.
newdata	Optional data frame of covariate profiles.
type	"response" (the mean $E(Y)$; for a zero-truncated fit the conditional mean $E(Y Y > 0)$), "link" (the linear predictor), or "prob" ($P(Y = at)$; for a zero-truncated fit this is $P(Y = at Y > 0)$).
at	Count value for type = "prob".
offset	Optional offset (log scale) for newdata: a numeric vector or a column name in newdata.
...	Unused.

Value

A numeric vector.

predict.cpb	<i>Predictions from a CPB fit</i>
-------------	-----------------------------------

Description

Predictions from a CPB fit

Usage

```
## S3 method for class 'cpb'
predict(
  object,
  newdata = NULL,
  type = c("response", "rate", "link", "ceiling", "prob"),
  at = NULL,
  offset = NULL,
  ...
)
```

Arguments

object	A "cpb" object.
newdata	Optional data frame of new covariate profiles; if omitted, the fitted data are used.
type	One of "response" (the exact mean of the fitted distribution; for a zero-truncated fit the conditional mean $E(Y Y \geq 1)$), "rate" (the CPB rate parameter $\lambda = \exp(x'b)$, which equals the mean only when $\lambda/(1-\alpha)$ is an integer), "link" (the linear predictor, $\log \lambda$), "ceiling" (the implied ceiling $\lambda/(1-\alpha)$), or "prob" (the probability that Y equals at).
at	For type = "prob", the count value(s) y whose probability is returned (length 1, or one per row of the prediction data).
offset	Optional offset (log scale) for the newdata branch: a numeric vector or a column name in newdata. For newdata = NULL the fit's own offset is used.
...	Unused.

Value

A numeric vector.

Examples

```
set.seed(1); x <- rnorm(300)
N <- pmax(round(exp(1.6 + 0.5 * x) / 0.5), 1); y <- rbinom(300, N, 0.5)
fit <- cpb(y ~ x, data = data.frame(y = y, x = x)[y > 0, ], se = "none")
predict(fit, newdata = data.frame(x = 0), type = "ceiling")
predict(fit, newdata = data.frame(x = c(-1, 1)), type = "prob", at = 5)
```

predict.cpb_fe

Predictions from a fixed-effects CPB fit

Description

Predictions from a fixed-effects CPB fit

Usage

```
## S3 method for class 'cpb_fe'
predict(
  object,
  newdata = NULL,
  type = c("response", "rate", "link", "ceiling"),
  ...
)
```

Arguments

object	A "cpb_fe" object.
newdata	Optional covariate profiles. With newdata, predictions use the average unit (the mean fixed effect), since the panel's own unit effects do not apply to new rows.
type	"response" (the mean; for a zero-truncated fit the conditional mean $E(Y Y \geq 1)$), "rate" (the CPB rate λ), "link" (the log rate), or "ceiling".
...	Unused.

Value

A numeric vector.

predict.gec

Predictions from a generalized event count fit

Description

Predictions from a generalized event count fit

Usage

```
## S3 method for class 'gec'
predict(
  object,
  newdata = NULL,
  type = c("response", "link", "prob"),
  at = NULL,
  offset = NULL,
  ...
)
```

Arguments

object	A "gec" object.
newdata	Optional covariate profiles.
type	"response" (the mean), "link" (the log mean), or "prob" (the probability of the count at).
at	Count value(s) for type = "prob".
offset	Optional offset (log scale) for the count component when predicting on newdata; a numeric vector or a column name in newdata.
...	Unused.

Value

A numeric vector.

predict.hurdle_count *Predict from a hurdle Poisson/NB fit*

Description

Predict from a hurdle Poisson/NB fit

Usage

```
## S3 method for class 'hurdle_count'
predict(
  object,
  newdata = NULL,
  type = c("response", "participation", "intensity"),
  offset = NULL,
  ...
)
```

Arguments

object	A "hurdle_count" object.
newdata	Optional data frame of covariate profiles.
type	"response" (marginal $E(Y)$, zeros included), "participation" ($P(Y > 0)$), or "intensity" ($E(Y Y > 0)$).
offset	Optional offset (log scale) for the intensity when predicting on newdata; a numeric vector or a column name in newdata.
...	Unused.

Value

A numeric vector.

predict.hurdle_cpb	<i>Predict from a hurdle-CPB fit</i>
--------------------	--------------------------------------

Description

Predict from a hurdle-CPB fit

Usage

```
## S3 method for class 'hurdle_cpb'
predict(
  object,
  newdata = NULL,
  type = c("response", "participation", "intensity"),
  ...
)
```

Arguments

object	A "hurdle_cpb" object.
newdata	Data frame of covariate profiles (must contain both the intensity and participation covariates).
type	"response" for the marginal expected count $E(Y)$ (zeros included), "participation" for $P(Y > 0)$, or "intensity" for $E(Y Y > 0)$.
...	Unused.

Value

A numeric vector.

predict.hurdle_gec	<i>Predict from a hurdle-GEC fit</i>
--------------------	--------------------------------------

Description

Predict from a hurdle-GEC fit

Usage

```
## S3 method for class 'hurdle_gec'
predict(
  object,
  newdata = NULL,
  type = c("response", "participation", "intensity"),
  ...
)
```

Arguments

object	A "hurdle_gec" object.
newdata	Optional covariate profiles (intensity and participation).
type	"response" (marginal $E(Y)$), "participation" ($P(Y>0)$), or "intensity" ($E(Y Y>0)$).
...	Unused.

Value

A numeric vector.

predict.zi_count	<i>Predict from a zero-inflated Poisson/NB fit</i>
------------------	--

Description

Predict from a zero-inflated Poisson/NB fit

Usage

```
## S3 method for class 'zi_count'
predict(
  object,
  newdata = NULL,
  type = c("response", "count", "zero", "intensity"),
  offset = NULL,
  ...
)
```

Arguments

object	A "zi_count" object.
newdata	Optional data frame of covariate profiles.
type	"response" (marginal $E(Y)$), "count" (equivalently "intensity"; the count component's mean $E(Y)$ of that component), or "zero" (structural-zero probability). "intensity" is accepted as a synonym for "count" for consistency with predict.zi_cpb() .
offset	Optional offset (log scale) for the count component when predicting on newdata; a numeric vector or a column name in newdata.
...	Unused.

Value

A numeric vector.

predict.zi_cpb	<i>Predict from a zi_cpb fit</i>
----------------	----------------------------------

Description

Predict from a zi_cpb fit

Usage

```
## S3 method for class 'zi_cpb'
predict(object, newdata = NULL, type = c("response", "zero", "intensity"), ...)
```

Arguments

object	A "zi_cpb" object.
newdata	Data frame of covariate profiles.
type	"response" for the marginal expected count $E(Y)$, "zero" for the structural-zero probability, or "intensity" for the CPB mean.
...	Unused.

Value

A numeric vector.

predict.zi_gec	<i>Predict from a zi_gec fit</i>
----------------	----------------------------------

Description

Predict from a zi_gec fit

Usage

```
## S3 method for class 'zi_gec'
predict(object, newdata = NULL, type = c("response", "zero", "intensity"), ...)
```

Arguments

object	A "zi_gec" object.
newdata	Optional covariate profiles.
type	"response" (marginal $E(Y)$), "zero" (structural-zero probability), or "intensity" (the count mean E of the GEC component).
...	Unused.

Value

A numeric vector.

rcpb

Simulate from the continuous parameter binomial

Description

Draws counts from the continuous parameter binomial (CPB) or its zero-truncated variant.

Usage

```
rcpb(n, lambda, alpha, truncated = FALSE)
```

Arguments

n	Number of draws (recycled against lambda).
lambda	Mean parameter; a scalar or a length-n vector.
alpha	Shape/dispersion parameter in (0, 1).
truncated	If TRUE, draw from the zero-truncated CPB.

Details

The support of the CPB is $0, \dots, \text{floor}(\text{lambda} / (1 - \alpha))$. A rate whose ceiling $\text{lambda} / (1 - \alpha)$ is below 1 has the support $\{0\}$, so its zero-truncated distribution does not exist. With `truncated = TRUE` such a draw is set to 1 and a warning reports how many there were: the stated parameters give that count probability zero, so data simulated this way are not data from the model (in a simulation study, keep every ceiling at 1 or above).

Value

An integer vector of counts.

Examples

```
set.seed(1)
table(rcpb(1000, lambda = 3, alpha = 0.5))
```

rhurdle_cpb

Simulate from the hurdle continuous parameter binomial

Description

Simulate from the hurdle continuous parameter binomial

Usage

```
rhurdle_cpb(n, lambda, alpha, p)
```

Arguments

n	Number of units.
lambda	Intensity mean for participants; scalar or length-n.
alpha	CPB shape parameter in (0, 1).
p	Participation probability; scalar or length-n.

Value

An integer vector: 0 for nonparticipants, a zero-truncated CPB draw otherwise.

Examples

```
set.seed(1)
y <- rhurdle_cpb(1000, lambda = 3, alpha = 0.5, p = 0.6)
mean(y == 0)
```

rootogram	<i>Hanging rootogram for a fitted count model</i>
-----------	---

Description

Draws a Tukey hanging rootogram: bars for the observed frequencies hang from the curve of expected frequencies, both on the square-root scale. Bars that hang below the zero line mark counts the model under-predicts; bars that stop short mark counts it over-predicts. A fit with frequency weights counts each row as many times as its weight in both the observed and the expected frequencies.

Usage

```
rootogram(
  fit,
  kmax = NULL,
  main = "Rootogram",
  xlab = "Count",
  ylab = "sqrt(frequency)",
  ...
)
```

Arguments

fit	A fitted underdisp count model (any class <code>score()</code> takes).
kmax	Highest count to display; defaults to the maximum observed count.
main, xlab, ylab	Plot labels.
...	Passed to <code>graphics::plot()</code> .

Value

Invisibly, a data frame of count, observed, and expected frequencies.

Examples

```
set.seed(1)
y <- rcpb(400, lambda = 3, alpha = 0.5)
fit <- cpb(y ~ 1, data = data.frame(y = y), truncated = FALSE, se = "none")
rootogram(fit)
```

 rzicpb

Simulate from the zero-inflated continuous parameter binomial

Description

Simulate from the zero-inflated continuous parameter binomial

Usage

```
rzicpb(n, lambda, alpha, pi)
```

Arguments

n	Number of units.
lambda	Intensity mean; scalar or length-n.
alpha	CPB shape parameter in (0, 1).
pi	Structural-zero probability; scalar or length-n.

Value

An integer vector: a structural zero with probability pi, otherwise an (untruncated) CPB draw, which may itself be zero.

Examples

```
set.seed(1)
y <- rzicpb(1000, lambda = 3, alpha = 0.5, pi = 0.3)
mean(y == 0)
```

score

*Proper scoring rules for a fitted count model***Description**

Computes the mean logarithmic score and the ranked probability score (RPS) of a fitted model's predicted distribution against observed counts. Lower is better for both, and both are proper, so they are the natural way to compare the calibration of competing count models.

Usage

```
score(fit, newdata = NULL, kmax = NULL, weights = NULL)
```

Arguments

fit	A fitted underdisp count model (cpb, cpb_fe, hurdle_cpb, zi_cpb, count_reg, hurdle_count, zi_count, gec, gec_fe, hurdle_gec, or zi_gec).
newdata	Optional data frame of held-out observations (including the response). When supplied, the fitted model's predicted distribution is evaluated on these rows; fixed-effect units unseen in training fall back to the mean fixed effect. Offsets are assumed absent on newdata.
kmax	Highest count to evaluate; defaults to the maximum observed count in the fitting data, raised to the maximum held-out count when newdata contains larger values. Predicted probabilities are floored at 1e-12 in the log score, so an observation outside a hard-ceiling model's support contributes 27.6 to it.
weights	Frequency weights for the rows of newdata: a column name of newdata or a numeric vector with one value per row. The held-out scores are then weighted means, as the in-sample scores of a weighted fit are.

Details

By default the scores are computed **in sample** (against the data the model was fit to, as means weighted by its frequency weights). Supply newdata to score a fitted model on **held-out** observations, or use [cv_score\(\)](#) for a cross-validated score; an in-sample log score equals $-\log\text{Lik}/n$ and does not penalize model complexity, so for comparing models of different size the held-out or cross-validated score is the honest criterion.

Value

A named numeric vector `c(logscore, rps)`.

See Also

[cv_score\(\)](#)

Examples

```
set.seed(1)
y <- rcpb(400, lambda = 3, alpha = 0.5)
fit <- cpb(y ~ 1, data = data.frame(y = y), truncated = FALSE, se = "none")
score(fit)
```

simulate.underdisp	<i>Simulate responses from a fitted underdisp model</i>
--------------------	---

Description

Draws `nsim` replicate response vectors from the fitted model, at the estimated parameters and the estimation data, in the format of `stats::simulate()`. Available for every model class in the package: `cpb`, `cpb_fe`, `gec`, `gec_fe` (through inheritance), `count_reg`, `hurdle_count`, `zi_count`, `hurdle_cpb`, `hurdle_gec`, `zi_cpb`, and `zi_gec`. Two-part models first draw the binary stage (participation or structural zero), then the count stage from its own distribution, so the replicates carry the model's full zero structure.

Usage

```
## S3 method for class 'cpb'
simulate(object, nsim = 1, seed = NULL, ...)

## S3 method for class 'cpb_fe'
simulate(object, nsim = 1, seed = NULL, ...)

## S3 method for class 'gec'
simulate(object, nsim = 1, seed = NULL, ...)

## S3 method for class 'count_reg'
simulate(object, nsim = 1, seed = NULL, ...)

## S3 method for class 'hurdle_cpb'
simulate(object, nsim = 1, seed = NULL, ...)

## S3 method for class 'hurdle_gec'
simulate(object, nsim = 1, seed = NULL, ...)

## S3 method for class 'hurdle_count'
simulate(object, nsim = 1, seed = NULL, ...)

## S3 method for class 'zi_cpb'
simulate(object, nsim = 1, seed = NULL, ...)

## S3 method for class 'zi_gec'
simulate(object, nsim = 1, seed = NULL, ...)
```

```
## S3 method for class 'zi_count'
simulate(object, nsim = 1, seed = NULL, ...)
```

Arguments

object	A fitted model from this package.
nsim	Number of replicate response vectors.
seed	Optional seed, handled as in <code>stats::simulate()</code> : the caller's RNG state is restored on exit when a seed is supplied.
...	Unused.

Details

The main consumer is simulated-residual diagnostics: `DHARMA::createdHARMA(simulatedResponse = as.matrix(simulate(fit, 250)), observedResponse = y, fittedPredictedResponse = fitted(fit), integerResponse = TRUE)` works for any fit in the family.

Value

A data frame with `nsim` integer columns, one row per observation, with a "seed" attribute.

Examples

```
set.seed(1); x <- rnorm(200)
N <- pmax(round(exp(1.4 + 0.4 * x) / 0.5), 1); y <- rbinom(200, N, 0.5)
fit <- cpb(y ~ x, data = data.frame(y = y, x = x)[y > 0, ], se = "none")
sims <- simulate(fit, nsim = 5)
colMeans(sims)
```

summary.cpb	<i>Summarize a CPB fit</i>
-------------	----------------------------

Description

Summarize a CPB fit

Usage

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

Arguments

object	A "cpb" object.
...	Unused.

Value

An object of class "summary.cpb" with the coefficient table, the dispersion parameter and its profile-likelihood interval (first-order, or calibrated when the fit went through `calibrate_alpha()`), the implied ceiling, fit statistics, and the likelihood-ratio test against a (zero-truncated) Poisson.

tidy.cpb

*Tidy a CPB fit (broom method)***Description**

The two-part classes (hurdle_*, zi_*) return one row per coefficient in each equation, with a component column (the **broom** convention for multi-equation models) and the term prefixed by its component, so that **modelsummary** lays the table out without a shape argument.

Usage

```
## S3 method for class 'cpb'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)
```

```
## S3 method for class 'cpb_fe'
tidy(x, ...)
```

```
## S3 method for class 'hurdle_cpb'
tidy(x, ...)
```

```
## S3 method for class 'zi_cpb'
tidy(x, ...)
```

```
## S3 method for class 'gec'
tidy(x, ...)
```

```
## S3 method for class 'hurdle_gec'
tidy(x, ...)
```

```
## S3 method for class 'zi_gec'
tidy(x, ...)
```

```
## S3 method for class 'count_reg'
tidy(x, ...)
```

```
## S3 method for class 'zi_count'
tidy(x, ...)
```

```
## S3 method for class 'hurdle_count'
tidy(x, ...)
```

Arguments

<code>x</code>	A "cpb" object.
<code>conf.int</code>	If TRUE, add <code>conf.low/conf.high</code> (requires a fit with bootstrap standard errors).
<code>conf.level</code>	Confidence level for the interval.
<code>...</code>	Unused.

Value

A data frame with one row per coefficient.

<code>ud_screen</code>	<i>Screen a count outcome for underdispersion</i>
------------------------	---

Description

Applies the diagnostic sequence developed in Bagozzi (2026): a *marginal* verdict from the conditional Pearson statistic and a through-origin score regression (which gives the direction of dispersion), a negative-binomial-versus-Poisson test for the overdispersion call, and—most importantly—an *at-risk* verdict on the positive counts benchmarked against a **zero-truncated Poisson**. The last step is what separates genuine underdispersion from the artifact of conditioning on $Y > 0$. When the data are not zero-dominated it also fits the CPB and reports a ceiling-exceedance diagnostic, and it fits a generalized-Poisson soft-tail comparator.

Usage

```
ud_screen(
  formula,
  data,
  run_cpb = TRUE,
  cpb_max_n = 3000,
  run_gp = TRUE,
  run_comp = TRUE,
  comp_max_par = 30,
  comp_max_n = 5000,
  ztp_threshold = c("calibrated", "bootstrap"),
  ztp_boot_B = 199L,
  cores = 1L,
  digits = 3,
  nb_boot = NULL
)
```

Arguments

<code>formula</code>	A model formula.
<code>data</code>	A data frame.

run_cpb	Logical; fit the CPB when the data are not zero-dominated (default TRUE).
cpb_max_n	Skip the CPB fit above this sample size (default 3000).
run_gp	Logical; fit the generalized-Poisson comparator (default TRUE); gated by comp_max_par and comp_max_n like the COM-Poisson.
run_comp	Logical; fit the native COM-Poisson comparator (default TRUE). Skipped when the mean model carries more than comp_max_par parameters (the COM-Poisson has no concentrated fixed-effects path, so dummy-heavy screens would be slow) or when n exceeds comp_max_n.
comp_max_par, comp_max_n	Parameter and sample-size gates for the generalized-Poisson and COM-Poisson comparators (defaults 30 and 5000).
ztp_threshold	How to set the at-risk test's underdispersion cutoff. "calibrated" (default) uses the simulation-calibrated rule $1 - 2.27/\sqrt{n_+}$, whose constant is an estimated standard deviation fitted to one calibration grid (no fixed effects, and that grid's rate profile); "bootstrap" calibrates the cutoff on the data at hand by a parametric bootstrap of the fitted zero-truncated Poisson null (simulate ztp_boot_B at-risk panels at the fitted rates, refit, and take the empirical 5% anti-conservative in two separable regimes: when the mean model's parameter count is a nontrivial share of the positive observations (dummy-heavy fixed-effects screens, where the null's center drifts with the parameter share), and when the fitted rates concentrate at small values (roughly λ below 2, and the more severely the smaller the rates), where the null's spread exceeds the fitted constant even without fixed effects (simulated size roughly 0.07–0.10 against the nominal 0.05). The bootstrap absorbs both departures by construction and is the recommended choice in either regime.
ztp_boot_B	Number of parametric-bootstrap replicates (default 199).
cores	Worker processes for the parametric-bootstrap replicates (default 1); see cpb() .
digits	Printing precision.
nb_boot	Whether the NB-vs-Poisson p-value is a parametric bootstrap under the fitted Poisson (ztp_boot_B replicates over cores), the package default at a boundary null (see dispersion_test()). NULL (default) bootstraps it when ztp_threshold = "bootstrap" and the mean model is within comp_max_par and comp_max_n; otherwise the p-value is the asymptotic boundary mixture, which is conservative for this test. The bootstrap restores the random-number state, so every other result is the same with or without it.

Details

Runtime: the marginal and at-risk arms take seconds. `ztp_threshold = "bootstrap"` refits the zero-truncated Poisson `ztp_boot_B` times (use `cores`); the CPB comparator is fit only up to `cpb_max_n` rows, and the generalized-Poisson and COM-Poisson comparators only when the mean model carries at most `comp_max_par` parameters and at most `comp_max_n` rows, beyond which their rows print NA; each is a full model fit (the two soft-tail families have no concentrated fixed-effects path, so a dummy-heavy screen would take minutes).

Value

An object of class "ud_screen" with verdict_marginal, verdict_atrisk, the conditional and at-risk (ZTP-benchmarked) Pearson statistics, the NB-vs-Poisson LR test (p_nb_method records how its p-value was computed), a log-likelihood comparison, (when fit) the CPB alpha and ceiling-exceedance share, and the over-conditioning guard state: atrisk_skipped (TRUE when the mean model nearly saturates the positive counts, so the at-risk statistic is not computed and the printout says why), sat_ratio (the fitted parameter share of the positives), and overconditioned (TRUE when that share reaches 0.10, the region where the calibrated threshold is anti-conservative; the printout then flags the verdict as diagnostic rather than probative and recommends ztp_threshold = "bootstrap").

References

King, G. (1989). Variance specification in event count models. *AJPS* 33(3):762-784.
 Bagozzi, B. E. (2026). Revisiting underdispersion in political science. Companion manuscript.

See Also

[cpb\(\)](#), [compare_dispersion\(\)](#)

Examples

```
set.seed(1); x <- rnorm(250)
N <- pmax(round(exp(1.4 + 0.4 * x) / 0.6), 1); y <- rbinom(250, N, 0.6)
ud_screen(y ~ x, data = data.frame(y = y, x = x))
```

vcov.cpb_fe

Bootstrap covariance for a fixed-effects CPB fit

Description

The covariance of the covariate coefficients from the stored pairs/cluster bootstrap (se = "bootstrap" at fit time); NULL when no bootstrap was run.

Usage

```
## S3 method for class 'cpb_fe'
vcov(object, ...)
```

Arguments

object	A "cpb_fe" object.
...	Unused.

Value

A covariance matrix, or NULL.

zi_count

*Zero-inflated count regression for the matched count families***Description**

Fits a structural-zero mixture whose count component comes from any of the package's count families (see [count_reg\(\)](#), Families), the count analogue of [zi_cpb\(\)](#). The count component uses a log link (on the mean for the mean-parameterized families, on the rate λ for "compois") and the inflation probability a link set by link. Returned as a "zi_count" object that [compare_models\(\)](#) and [score\(\)](#) accept.

Usage

```
zi_count(
  formula,
  data,
  family = c("poisson", "negbin", "compois", "mpcmp", "genpois", "gammacount",
    "doublepois"),
  zero = NULL,
  fe = NULL,
  zero_fe = NULL,
  link = c("logit", "probit", "cloglog"),
  offset = NULL,
  weights = NULL,
  se = c("analytic", "robust", "cluster", "none"),
  cluster = NULL
)
```

Arguments

formula	Count formula ($y \sim x$).
data	A data frame.
family	The count-component family; see count_reg() .
zero	Optional one-sided formula for the inflation (structural-zero) model; defaults to the count right-hand side.
fe	Optional fixed-effects column for the count equation (factor dummies).
zero_fe	Optional fixed-effects column for the inflation equation (factor dummies); zero-equation fixed effects are opt-in.
link	Link for the inflation probability: "logit" (default), "probit", or "cloglog". Positive inflation coefficients raise the probability of a <i>structural zero</i> .
offset	Optional offset for the count component, on the linear-predictor (log) scale: a numeric vector or the name of a column in data.
weights	Optional frequency weights (a numeric vector or a column name); see count_reg() .
se, cluster	Standard-error type and optional cluster; see count_reg() .

Value

An object of class "zi_count"; `$fitted.values` is the marginal mean $(1 - \pi) E(Y \mid \text{count component})$ that `fitted()` returns, and `$mu` the count component's natural parameter.

See Also

[zi_cpb\(\)](#), [hurdle_count\(\)](#), [compare_models\(\)](#)

Examples

```
set.seed(2); n <- 300; x <- rnorm(n); z <- rnorm(n)
y <- ifelse(rbinom(n, 1, plogis(-0.5 + 0.8 * z)) == 1, 0L, rpois(n, exp(1 + 0.3 * x)))
zi_count(y ~ x, data.frame(y = y, x = x, z = z), family = "poisson",
        zero = ~ z, se = "none")
```

 zi_cpb

Zero-inflated continuous parameter binomial regression

Description

Fits the zero-inflated CPB. The model mixes a structural-zero process (a logistic model for the probability π that a unit is a structural zero) with an untruncated CPB for the count, so a zero can arise either structurally or as a sampling zero from the CPB. Unlike [hurdle_cpb\(\)](#), the likelihood does not factorize, so the parameters are estimated by direct joint maximum likelihood, seeded from separate fits (a zero-truncated CPB on the positives and a logit of the zero indicator) and cross-checked against an expectation-maximization climb of the same observed-data likelihood, keeping the better optimum. An EM-only path (`method = "em"`) is retained for comparison.

Usage

```
zi_cpb(
  formula,
  data,
  zero = NULL,
  fe = NULL,
  zero_fe = NULL,
  method = c("ml", "em"),
  se = c("none", "bootstrap"),
  B = 500,
  cluster = NULL,
  max.support = 500,
  maxit = 200,
  tol = 1e-06,
  offset = NULL,
  weights = NULL,
  cores = 1L
)
```

Arguments

formula	Intensity (count) model formula ($y \sim x$).
data	A data frame.
zero	Optional one-sided formula ($\sim z$) for the zero-inflation model; defaults to the intensity model's right-hand side.
fe	Optional column name for unit fixed effects in the intensity (count) component. Under method = "ml" the unit effects enter as plug-in offsets from a first-stage fit; under method = "em" a classification (hard-assignment) step handles the structural zeros.
zero_fe	Optional column name for fixed effects in the zero-inflation equation, entered as factor dummies (opt-in).
method	Estimation method: "ml" (default) is robust direct joint maximum likelihood seeded from separate fits; "em" is expectation- maximization, retained for comparison but prone to a degenerate structural-zero-probability collapse under heavy zero-inflation.
se	Coefficient inference: "none" (default) or "bootstrap".
B	Bootstrap resamples when se = "bootstrap".
cluster	Optional cluster for the bootstrap (a column name or vector); under fixed effects the units are resampled by default. The mixture EM makes the bootstrap costly, so keep B modest.
max.support	Maximum support for the CPB pmf.
maxit	Optimizer iteration budget (scaled internally for the joint maximization; also the EM iteration cap under method = "em").
tol	Relative convergence tolerance on the observed-data log-likelihood.
offset	Optional exposure offset (log scale) for the count component: a numeric vector or the name of a column in data, making the count a rate model. Supported for method = "ml" without fe (the fixed-effects path's plug-in unit effects have no offset handling yet, and errors loudly). The bootstrap carries the offset through every replicate.
weights	Optional frequency weights (a numeric vector or a column name); see cpb() . Supported for method = "ml".
cores	Worker processes for the bootstrap; see cpb() .

Details

The inflation equation uses a logit link. For a probit or cloglog inflation link, use [zi_count\(\)](#) (Poisson/NB/COM-Poisson count), whose link argument covers the binary stage.

Value

An object of class "zi_cpb".

Examples

```
set.seed(1)
n <- 800; x <- rnorm(n); z <- rnorm(n)
y <- rzicpb(n, lambda = exp(1.3 + 0.5 * x), alpha = 0.5, pi = plogis(-0.5 + 0.8 * z))
zi_cpb(y ~ x, data = data.frame(y = y, x = x, z = z), zero = ~ z)
```

 zi_pec

Zero-inflated GEC (Katz-family) regression

Description

A structural-zero mixture with a GEC count component whose dispersion δ is estimated freely. Estimated by robust direct maximum likelihood, seeded from separate fits (a GEC on the positive counts for the count parameters and a logit of the zero indicator for the inflation), exactly as in [zi_cpb\(\)](#); this avoids the degenerate $\pi \rightarrow 0$ basin that coordinate-wise EM falls into.

Usage

```
zi_pec(
  formula,
  data,
  zero = NULL,
  zero_fe = NULL,
  se = c("none", "bootstrap"),
  B = 500,
  cluster = NULL,
  max.support = 500,
  maxit = 200,
  tol = 1e-06,
  offset = NULL,
  weights = NULL,
  cores = 1L
)
```

Arguments

formula	Count formula, $y \sim x$.
data	A data frame.
zero	One-sided formula for the structural-zero logit; defaults to the count formula's right-hand side.
zero_fe	Optional column name for fixed effects in the zero-inflation equation, entered as factor dummies (opt-in).
se	Coefficient inference: "none" (default) or "bootstrap".
B	Bootstrap resamples.

cluster	Optional cluster (column name or vector) for the bootstrap.
max.support	Guard on the maximum evaluated support.
maxit,tol	Optimizer controls.
offset	Optional exposure offset (log scale) for the count component: a numeric vector or the name of a column in data; the bootstrap carries it through every replicate.
weights	Optional frequency weights (a numeric vector or a column name); see cpb() .
cores	Worker processes for the bootstrap; see cpb() .

Value

An object of class "zi_gec".

See Also

[zi_cpb\(\)](#), [gec\(\)](#), [hurdle_gec\(\)](#)

Examples

```
set.seed(4); n <- 300; x <- rnorm(n); z <- rnorm(n)
y <- ifelse(rbinom(n, 1, plogis(-0.5 + 0.8 * z)) == 1, 0L, rpois(n, exp(1 + 0.3 * x)))
zi_gec(y ~ x, data.frame(y = y, x = x, z = z), zero = ~ z)
```

 zi_test

Boundary-corrected test for zero-inflation (CPB vs ZI-CPB)

Description

A likelihood-ratio test of whether a count needs a structural-zero component. The (untruncated) CPB is nested in the zero-inflated CPB at a structural-zero probability of zero. Because that value lies on the boundary of the parameter space, the LR statistic follows a $\frac{1}{2}\chi_0^2 + \frac{1}{2}\chi_1^2$ mixture (Self and Liang 1987), which halves the naive χ_1^2 p-value. The test uses an intercept-only structural-zero probability, so it is a clean single-parameter boundary test; a covariate-dependent structural-zero model is better compared with information criteria and proper scores via [compare_dispersion\(\)](#).

Usage

```
zi_test(object, object2 = NULL, data = NULL, ...)
```

Arguments

object	Either a model formula — then data is required and the nested pair is fit internally — or a fitted model (cpb, cpb_fe, or zi_cpb).
object2	The second argument: the data frame when object is a formula (so <code>zi_test(y ~ x, mydata)</code> works), or the second fitted model when object is a fit. In the model interface exactly one of the two models must be a zero-inflated <code>zi_cpb</code> and the other its non-inflated nest (cpb or cpb_fe); the two may carry any combination of fixed effects and robust/clustered standard errors.

data	A data frame; an alternative to passing it as object2.
...	Passed to <code>cpb()</code> and <code>zi_cpb()</code> in the formula interface.

Value

An object of class "zi_test" with the two log-likelihoods, the LR statistic, and the boundary-corrected p-value.

Examples

```
set.seed(1); x <- rnorm(500)
y <- rzicpb(500, lambda = exp(1.2 + 0.4 * x), alpha = 0.5, pi = 0.3)
zi_test(y ~ x, data = data.frame(y = y, x = x))
```

Index

* datasets

- peacekeeping, [48](#)

- `alpha_confint`, [3](#)
- `alpha_confint()`, [5](#), [6](#), [16](#), [44](#)
- `augment.cpb`, [4](#)

- `calibrate_alpha`, [5](#)
- `calibrate_alpha()`, [3](#), [10](#), [16](#), [44](#), [63](#)
- `compare_dispersion`, [6](#)
- `compare_dispersion()`, [8](#), [34](#), [66](#), [71](#)
- `compare_models`, [8](#)
- `compare_models()`, [12](#), [14](#), [23](#), [40](#), [41](#), [67](#), [68](#)
- `compois-distribution`, [9](#)
- `confint.count_reg (confint.underdisp)`, [10](#)
- `confint.cpb`, [10](#)
- `confint.cpb()`, [5](#), [11](#), [16](#)
- `confint.cpb_fe (confint.underdisp)`, [10](#)
- `confint.gec (confint.underdisp)`, [10](#)
- `confint.hurdle_count (confint.underdisp)`, [10](#)
- `confint.hurdle_cpb (confint.underdisp)`, [10](#)
- `confint.hurdle_gec (confint.underdisp)`, [10](#)
- `confint.underdisp`, [10](#)
- `confint.zi_count (confint.underdisp)`, [10](#)
- `confint.zi_cpb (confint.underdisp)`, [10](#)
- `confint.zi_gec (confint.underdisp)`, [10](#)
- `count_reg`, [12](#)
- `count_reg()`, [10](#), [20](#), [25](#), [26](#), [33](#), [39–41](#), [47](#), [67](#)
- `cpb`, [14](#)
- `cpb()`, [5](#), [7](#), [10](#), [12](#), [14](#), [19–21](#), [31](#), [33](#), [34](#), [37](#), [41–43](#), [47](#), [65](#), [66](#), [69](#), [71](#), [72](#)
- `cpb-distribution`, [17](#)
- `cpb_fe`, [18](#)
- `cpb_fe()`, [10](#), [12](#), [15](#), [16](#), [36](#), [37](#), [42](#)
- `cv_score`, [20](#)
- `cv_score()`, [60](#)

- `dcompois (compois-distribution)`, [9](#)
- `dcpb (cpb-distribution)`, [17](#)
- `ddoublepois (doublepois-distribution)`, [25](#)
- `dgammaaccount (gammaaccount-distribution)`, [32](#)
- `dgec (gec-distribution)`, [35](#)
- `dgenpois (genpois-distribution)`, [38](#)
- `dispersion_profile`, [22](#)
- `dispersion_profile()`, [12–14](#), [25](#)
- `dispersion_test`, [23](#)
- `dispersion_test()`, [6](#), [12–14](#), [16](#), [23](#), [34](#), [65](#)
- `doublepois-distribution`, [13](#), [25](#)

- `first_difference`, [27](#)
- `first_difference()`, [44](#), [47](#)
- `first_difference.count (first_difference.gec)`, [28](#)
- `first_difference.count_reg (first_difference.gec)`, [28](#)
- `first_difference.cpb_fe (first_difference.gec)`, [28](#)
- `first_difference.gec`, [28](#)
- `first_difference.gec_fe (first_difference.gec)`, [28](#)
- `first_difference.hurdle_count (first_difference.gec)`, [28](#)
- `first_difference.hurdle_cpb`, [30](#)
- `first_difference.hurdle_gec (first_difference.gec)`, [28](#)
- `first_difference.zi_count (first_difference.gec)`, [28](#)
- `first_difference.zi_cpb (first_difference.hurdle_cpb)`, [30](#)
- `first_difference.zi_gec (first_difference.gec)`, [28](#)
- `fitted.cpb_fe (fitted.underdisp)`, [31](#)
- `fitted.hurdle_count (fitted.underdisp)`, [31](#)

- fitted.hurdle_cpb(fitted.underdisp), 31
- fitted.hurdle_gec(fitted.underdisp), 31
- fitted.underdisp, 31
- fitted.zi_count(fitted.underdisp), 31
- fitted.zi_cpb(fitted.underdisp), 31
- fitted.zi_gec(fitted.underdisp), 31

- gammacount-distribution, 13, 32
- gec, 33
- gec(), 10, 12, 14, 35–37, 44, 47, 71
- gec-distribution, 35
- gec_fe, 36
- gec_fe(), 10
- genpois-distribution, 13, 38
- glance.count_reg(glance.cpb), 39
- glance.cpb, 39
- glance.cpb_fe(glance.cpb), 39
- glance.gec(glance.cpb), 39
- glance.hurdle_count(glance.cpb), 39
- glance.hurdle_cpb(glance.cpb), 39
- glance.hurdle_gec(glance.cpb), 39
- glance.zi_count(glance.cpb), 39
- glance.zi_cpb(glance.cpb), 39
- glance.zi_gec(glance.cpb), 39
- graphics::barplot(), 49
- graphics::plot(), 58

- hurdle_count, 40
- hurdle_count(), 14, 68
- hurdle_cpb, 41
- hurdle_cpb(), 5, 7, 40, 41, 43, 44, 68
- hurdle_gec, 43
- hurdle_gec(), 34, 71

- implied_ceiling, 44
- implied_ceiling(), 5, 16
- irr, 45
- irr.count(irr.count_reg), 46
- irr.count_reg, 46
- irr.cpb_fe(irr.count_reg), 46
- irr.gec(irr.count_reg), 46
- irr.gec_fe(irr.count_reg), 46
- irr.hurdle_count(irr.count_reg), 46
- irr.hurdle_gec(irr.count_reg), 46
- irr.zi_count(irr.count_reg), 46
- irr.zi_gec(irr.count_reg), 46

- mundlak, 47
- pcompois(compois-distribution), 9
- pcpb(cpb-distribution), 17
- pdoublepois(doublepois-distribution), 25
- peacekeeping, 48
- pgammacount(gammacount-distribution), 32
- pgec(gec-distribution), 35
- pgenpois(genpois-distribution), 38
- pit_hist, 49
- plot.dispersion_profile
(dispersion_profile), 22
- predict.count_reg, 50
- predict.cpb, 50
- predict.cpb(), 16
- predict.cpb_fe, 51
- predict.gec, 52
- predict.hurdle_count, 53
- predict.hurdle_cpb, 54
- predict.hurdle_gec, 54
- predict.zi_count, 55
- predict.zi_cpb, 56
- predict.zi_cpb(), 55
- predict.zi_gec, 56

- qcompois(compois-distribution), 9
- qcpb(cpb-distribution), 17
- qdoublepois(doublepois-distribution), 25
- qgammacount(gammacount-distribution), 32
- qgec(gec-distribution), 35
- qgenpois(genpois-distribution), 38

- rcompois(compois-distribution), 9
- rcpb, 57
- rcpb(), 17
- rdoublepois(doublepois-distribution), 25
- rgammacount(gammacount-distribution), 32
- rgec(gec-distribution), 35
- rgenpois(genpois-distribution), 38
- rhurdle_cpb, 57
- rootogram, 58
- rootogram(), 23
- rzicpb, 59

- score, 60
- score(), 7, 12, 21, 40, 49, 58, 67

`simulate.count_reg`
 (`simulate.underdisp`), 61
`simulate.cpb` (`simulate.underdisp`), 61
`simulate.cpb_fe` (`simulate.underdisp`), 61
`simulate.gec` (`simulate.underdisp`), 61
`simulate.hurdle_count`
 (`simulate.underdisp`), 61
`simulate.hurdle_cpb`
 (`simulate.underdisp`), 61
`simulate.hurdle_gec`
 (`simulate.underdisp`), 61
`simulate.underdisp`, 61
`simulate.zi_count` (`simulate.underdisp`),
 61
`simulate.zi_cpb` (`simulate.underdisp`), 61
`simulate.zi_gec` (`simulate.underdisp`), 61
`stats::fitted()`, 31
`stats::optim()`, 15
`stats::simulate()`, 5, 61, 62
`summary.cpb`, 62

`tidy.count_reg` (`tidy.cpb`), 63
`tidy.cpb`, 63
`tidy.cpb_fe` (`tidy.cpb`), 63
`tidy.gec` (`tidy.cpb`), 63
`tidy.hurdle_count` (`tidy.cpb`), 63
`tidy.hurdle_cpb` (`tidy.cpb`), 63
`tidy.hurdle_gec` (`tidy.cpb`), 63
`tidy.zi_count` (`tidy.cpb`), 63
`tidy.zi_cpb` (`tidy.cpb`), 63
`tidy.zi_gec` (`tidy.cpb`), 63

`ud_screen`, 64
`ud_screen()`, 7, 16, 20

`vcov.cpb_fe`, 66

`zi_count`, 67
`zi_count()`, 10, 14, 41, 69
`zi_cpb`, 68
`zi_cpb()`, 7, 10, 67, 68, 70–72
`zi_gec`, 70
`zi_gec()`, 10, 44
`zi_test`, 71
`zi_test()`, 8, 25