Package 'underdisp'

November 4, 2026

Type Package

Title Diagnostics and Models for Underdispersed Count Data

Version 0.2.0

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, double Poisson, hyper-Poisson, and exponentially weighted 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 (>= 3.0.0), knitr, rmarkdown, broom, modelsummary, texreg, gamlss.dist, rmutil, glmmTMB, DGLMExtPois, ewp

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 2026-11-04 18:40:02 UTC

Contents

alpha_confint . . . 3
ame . . . 4
augment.cpb . . . 6
calibrate_alpha . . . 7
ceiling_exceedance . . . 9
compare_dispersion . . . 10
compare_models . . . 11
compois-distribution . . . 13
confint.cpb . . . 14
confint.underdisp . . . 14
count_reg . . . 16
cpb . . . 19
cpb-distribution . . . 23
cpb_fe . . . 24
cv_score . . . 28
dispersion_profile . . . 29
dispersion_test . . . 31
doublepois-distribution . . . 33
ewp-distribution . . . 34
first_difference . . . 36
first_difference.gec . . . 37
first_difference.hurdle_cpb . . . 39
fitted.underdisp . . . 41
gammacount-distribution . . . 42
gec . . . 43
gec-distribution . . . 44
gec_fe . . . 45
genpois-distribution . . . 47
glance.cpb . . . 49
hurdle_count . . . 50

hurdle_cpb . . . 51
hurdle_gec . . . 53
hyperpois-distribution . . . 55
implied_ceiling . . . 56
irr . . . 57
irr.count_reg . . . 58
mundlak . . . 59
peacekeeping . . . 60
pit_hist . . . 61
predict.count_reg . . . 62
predict.cpb . . . 63
predict.cpb_fe . . . 65
predict.gec . . . 66
predict.gec_fe . . . 67
predict.hurdle_count . . . 68
predict.hurdle_cpb . . . 69
predict.hurdle_gec . . . 70
predict.zi_count . . . 71
predict.zi_cpb . . . 72
predict.zi_gec . . . 73
rcpb . . . 74
residuals.count_reg . . . 75
rhurdle_cpb . . . 75
rootogram . . . 76
rzicpb . . . 77
score . . . 78
simulate.underdisp . . . 79
summary.cpb . . . 80
tidy.cpb . . . 81
ud_screen . . . 82
vcov.cpb_fe . . . 85
zi_count . . . 85
zi_cpb . . . 87
zi_gec . . . 90
zi_test . . . 92

Index **94**

`alpha_confint` *Profile-likelihood interval for the dispersion parameter alpha*

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). On King's support it covers between 0.940 and 0.956 in simulations from the CPB. On the floor support it covers about 0.90 to 0.94, with most misses on the upper side, because the estimate of `alpha` is biased toward zero there (`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 does not fall to the cut on the lower side: on King's support, whose trace runs down to `alpha = 1e-6`, the lower limit is then 0; on the floor support, whose trace stops at `alpha = 0.005`, it is that 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)
```

---

ame *Average marginal effects*

Description

The average, over the rows of `newdata`, of the change in a predicted quantity when one variable changes: its derivative (a numeric variable, with `from` and `to` left out) or the contrast between two of its values (`from` and `to`, for a variable of any type). Every row keeps its own values of the other variables, and the variable enters through the model's formula, so interactions and transformations of it (`x:z`, `I(x^2)`, `log(x)`) are handled. `first_difference()` instead moves one model-matrix column at a single reference profile.

Usage

```
ame(
  object,
  variable,
  newdata,
  from = NULL,
  to = NULL,
  type = "response",
  eps = NULL,
```

```
  level = 0.95,
  offset = NULL,
  weights = NULL,
  B = 0,
  cores = 1L
)
```

Arguments

`object`	A fitted model of any class in the package.
`variable`	The name of a variable of the model, a column of `newdata`.
`newdata`	The rows to average over, usually the data the model was fit to.
`from, to`	Two values of `variable` for a contrast. Leave both out for the derivative of a numeric variable.
`type`	The quantity, as the `type` of the class's `predict()` method. The default `"response"` is the expected count (for a zero-truncated fit, such as `cpb()`'s default, the conditional mean E(Y | Y >= 1)); the two-part models also give their equation-specific quantities.
`eps`	The step of the central difference for the derivative; by default 1e-4 times the range of `variable` in `newdata`.
`level`	Confidence level.
`offset`	Passed to `predict()`.
`weights`	Optional weights for the average over the rows: a numeric vector or the name of a column of `newdata`, such as the frequency weights of a fit to tabulated data (the average then equals the one over the expanded rows). By default the rows count equally.
`B`	Refit-bootstrap replicates for a percentile interval; `0` (default) uses the fit's own inference (see Details).
`cores`	Worker processes for the refits; see `cpb()`.

Details

The interval follows the rule of `first_difference()`. With stored bootstrap draws, the average is computed at every draw, jointly over the coefficients, the dispersion parameter and a fixed-effects fit's reference unit (the draws of a fixed-effects fit recentered on its estimate), and the percentile interval is taken. Otherwise it uses the delta method over the fit's covariance, and otherwise there is none. A row at a fixed-effects unit's own effect, or at its own zero-equation dummy, depends on a parameter the draws do not carry, so an average over such rows, the sample's average partial effect, has no interval of this kind (`B` below gives it one). Without the unit column in `newdata` every row enters at the reference unit, the mean of the identified units' effects on the log scale (a typical, geometric-mean unit), whose draws the fit carries: that is the average effect for such a unit over the rows' covariates, not the sample's average partial effect, and the two can differ substantially when the unit effects are dispersed.

A fixed-effects count family (`count_reg()` with `fe`) carries its unit dummies in its covariance, so its average partial effect has a delta-method interval. The slope of a fit on the floor support (`support = "floor"`) is refused, since its mean jumps at integer ceilings; a contrast is available.

`B > 0` gives the average a refit-bootstrap interval instead, for any class: the model is refit on `B` resampled copies of `newdata`, which must then be the data the model was fit to, resampling whole clusters of the fit's `cluster`, else whole units of its first fixed-effects column, else rows (a unit drawn twice becomes two units). Each replicate averages over its own rows at its own unit effects, and the percentile interval is taken. Resampling the units treats them as a sample, so this is an interval for a fixed-effects fit's average partial effect in the population of units they represent, which carries the variability of which units are drawn: with few units, or a few with large effects, it is wide and asymmetric. Each refit is the fit's own call with its data replaced and its standard errors off (on King's support, every replicate of a jackknife-corrected fit is corrected without re-running the validity gate, as in `cpb_fe()`'s own bootstrap, and none is when the gate refused the fit's correction); the call's other arguments are evaluated where `ame()` is called, as `update()` does, and an offset or weights given to the fit as one value per row follow the resampled rows. The run takes about `B` times one fit (see `cores`).

Value

A data frame of class `"ud_ame"` with `term`, `contrast` (`"dY/dx"`, or `"to - from"`), `type`, `estimate`, `lower`, `upper`, `method` (the interval's source) and `n`, the rows averaged (a row with a missing value is left out).

See Also

`first_difference()`, `predict.cpb()` (`se.fit`, `interval`).

Examples

```
set.seed(1); n <- 400; x <- rnorm(n); g <- factor(sample(c("a", "b"), n, TRUE))
d <- data.frame(y = rcpb(n, exp(1 + 0.4 * x + 0.3 * (g == "b")), 0.5), x = x, g = g)
fit <- cpb(y ~ x * g, data = d, truncated = FALSE, se = "bootstrap", B = 100)
ame(fit, "x", d)
ame(fit, "g", d, from = "a", to = "b")

## a fixed-effects fit's average partial effect over its own units
pd <- do.call(rbind, lapply(1:30, function(i) {
  x <- rnorm(8); data.frame(unit = i, x = x, y = rcpb(8, exp(rnorm(1, 0, 0.5) + 0.4 * x), 0.5))
}))
ffit <- cpb_fe(y ~ x, data = pd, fe = "unit")
ame(ffit, "x", pd, B = 100)
```

`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, data = NULL, ...)
```

Arguments

x	A "cpb" object.
data	Optional: the data frame the model was fit to. Its rows the fit used (those without a missing model variable) are returned with the new columns appended.
...	Unused.

Value

A data frame with fitted values, response residuals, and the implied per-observation ceiling (appended to data when it is given).

---

calibrate_alpha *Calibrate the interval for the CPB dispersion parameter*

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. It is the remedy for floor-support fits (support = "floor"). On King's support, the default, the first-order interval covers close to its level, and the calibration is an optional refinement.

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 cpb() fit, or a hurdle_cpb() 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 stats::simulate().
...	Unused.

Details

The first-order interval cuts the profile log-likelihood `qchisq(level, 1) / 2` below its maximum on both sides. On King's support that cut is close to calibrated, because the probabilities change continuously with the parameters. The simulations from the CPB cover nine designs (n from 200 to 1,000, true `alpha` 0.3, 0.5 and 0.7). There the first-order 95% interval covers between 0.940 and 0.956, 0.947 on average, and more of its misses fall above the upper limit than below the lower one. With `B = 39` replicates, the calibrated interval covers 0.948 on average in the same simulations. It is about 12% longer, and its misses are more nearly balanced.

On the floor support the cut is not calibrated. The support `0, ..., floor(lambda / (1 - alpha))` moves with the parameters, the log-ceiling coefficients behave like endpoint parameters, and `alpha-hat` inherits a bias toward zero from them. Across the same nine designs the first-order 95% interval covers between 0.90 and 0.94, and most of its misses have the true `alpha` above the upper limit.

The calibration simulates `B` responses from the fitted model (the fitted rates, `alpha`-hat, the same truncation and offset), refits each one, and records the signed root `r* = sign(alpha* - alpha-hat) sqrt(2 (l* - l*_p(al` The interval is then `{alpha : q_lo <= r(alpha) <= 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. On the floor support, in the same simulations, the calibrated 95% interval covers 0.92 to 0.96, with its misses balanced between the two sides. On the floor support, 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 `alpha` 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 `alpha`), 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
```

---

ceiling_exceedance *Held-out ceiling exceedance of a CPB fit*

Description

Whether counts respect the CPB's ceiling can be checked, but not in sample: its likelihood is minus infinity for any count above the bound, so every feasible fit respects the bound on the data it was fit to. The check is one-sided: it detects a ceiling that is too low, where held-out counts reach it, and has no power against one that is too high (a fitted ceiling above a true bound is never exceeded), nor where counts stay well below the ceiling (see `implied_ceiling()`). `ceiling_exceedance()` takes held-out observations and reports how many of their counts exceed the largest count the fitted model allows at their covariates (`max_count` of `implied_ceiling()`); for a two-part model the bound is the count component's, and a fixed-effects unit seen in training enters at its own effect (an unseen one at the mean effect), as in `score()`. A breached ceiling also zeroes the held-out likelihood, which `score()` records with its 1e-12 floor.

Usage

```
ceiling_exceedance(fit, newdata, weights = NULL, offset = NULL)
```

Arguments

`fit`	A `"cpb"`, `"cpb_fe"`, `"hurdle_cpb"`, or `"zi_cpb"` object.
`newdata`	Held-out observations, including the response.
`weights`	Optional frequency weights for the rows of `newdata`: a column name of `newdata` or a numeric vector with one value per row.
`offset`	Optional offset (log scale) of the count for the `newdata` rows; by default a fit made with an offset reads it from the same column of `newdata` (see `predict.cpb()`).

Value

A one-row data frame with `n` (held-out rows, weighted), `exceed` (rows above the bound, weighted), and `share`.

See Also

`implied_ceiling()`, `score()`

Examples

```
set.seed(7); x <- rnorm(400); y <- rcpb(400, exp(1.4 + 0.3 * x), 0.5)
d <- data.frame(y = y, x = x)
fit <- cpb(y ~ x, d[1:300, ], truncated = FALSE, se = "none")
ceiling_exceedance(fit, d[301:400, ])
```

`compare_dispersion` *Compare count-model dispersion across the model family*

Description

Fits the Poisson and negative binomial defaults alongside the underdispersed models—the soft-tail Conway–Maxwell–Poisson (fit natively, no external dependency), the hard-ceiling CPB, and the GEC, which spans both directions—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 hard ceiling (CPB) is a question for the data's fit and calibration, not an assumption; the scoring rules are the calibration counterpart to the information criteria. Either family's better fit describes the distribution, not the process behind it. The CPB's bound can be checked only on held-out counts, with `ceiling_exceedance()`, and only against a ceiling that is too low: on the data it was fit to every feasible CPB respects its bound, since its likelihood forbids exceedance.

Usage

```
compare_dispersion(
  formula,
  data,
  max.support = NULL,
  hurdle = FALSE,
  zi = FALSE,
  support = c("ceiling", "floor")
)
```

Arguments

`formula`	A model formula.
`data`	A data frame.
`max.support`	Passed to the CPB and GEC fits; the default `NULL` lets each set its own (see `cpb()`); on the floor support it is 500.
`hurdle`	If `TRUE` and the data contain zeros, also fit and score a `hurdle_cpb()`.
`zi`	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`.
`support`	The CPB's support for the CPB-family fits, `"ceiling"` (King's CPB, the default) or `"floor"`; see `cpb()`.

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`, `zero_fit`, and `converged` 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), `gec_delta` (the GEC's dispersion, `NA` if its fit failed), 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: the CPB's support cannot hold the counts, which another family, or a hurdle or zero-inflated model, may fit).

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` *Compare fitted underdispersed-count models*

Description

Compares fitted models from this package — `cpb`, `cpb_fe`, `hurdle_cpb`, `zi_cpb`, `gec`, `gec_fe`, `hurdle_gec`, `zi_gec`, `count_reg`, `hurdle_count`, and `zi_count`, 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 of the classes above, optionally named. With frequency weights, BIC uses the weight total (`nobs()`).

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, and the flags `truncated` (a zero-truncated likelihood, conditional on `Y >= 1`) and `separated` (a zero-inflation or participation equation the data do not bound, whose log-likelihood is a supremum along a divergent ridge; see `zi_cpb()`, Separation). The function warns when the models mix truncated and untruncated likelihoods, whose log-likelihoods are of different events, or when a model is separated.

Degrees of freedom

Each model's `df` is its own (`logLik()`'s). The hurdle and zero-inflated fits do not count the parameters of a unit with no positive count: with unit fixed effects in the participation or zero equation (`part_fe`, `zero_fe`) or in the count equation (`fe`; dummies written into a formula are covariates to the fit and are counted), such a unit's dummy and its count effect sit at their limits, where every one of its rows has probability one, so it adds nothing to the likelihood. As fixest leaves such units out of a fixed-effects fit, these parameters are left out of `df`, so that a hurdle and a zero-inflated model fit to the same panel are charged alike. The fits record the units (`part_zero_units` in the hurdles; `zero_units` and `zero_fe_zero_units` in `zi_cpb()`; `fe_zero_units` and `zero_fe_zero_units` in `zi_count()`; `zero_fe_zero_units` in `zi_gec()`), and their print methods give the number. In new data (`predict()`, `score()`, `cv_score()`) the hurdles give such a unit's rows the participation probability at that limit, zero, as the zero-inflated fits give them a count effect at its limit. A unit whose every count is positive has its hurdle participation dummy at the other limit, plus infinity; that dummy is counted, as the zero-inflated fits count their zero-equation dummy for the unit, and in new data the hurdles give the unit's rows the participation probability one (the fits record these units in `part_one_units`). The zero-inflated fits give a `zero_fe` unit's rows the structural-zero probability at its limit in the same way: one for a unit with no positive count, zero for a unit whose every count is positive (`zero_fe_one_units`). CPB fits on the floor support (`support = "floor"`) count these parameters, and in new data evaluate these units at the coefficients the fit stopped at.

The rule is the two-part fits'. The single-equation fixed-effects fits (`cpb_fe()`, `gec_fe()`, `count_reg(fe = )`) count every unit's effect, a unit with no positive count included; set beside a two-part fit to the same panel, such a fit is charged two AIC points per unit with no positive count that the two-part fit is not, and `compare_models()` says so in a message.

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 `lambda` (or mean `mu`) and dispersion `nu` (`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

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

Value

`dcompois` a density, `pcompois` a CDF, `qcompois` a quantile, `rcompois` a numeric vector of count draws. A rate whose normalizing sum the 100,000-term cap would truncate (a mean of order 10^5 or more) gives `NA`.

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. In simulations from King's CPB (n = 300) the 95% interval for a slope covers 0.91 at alpha 0.3, 0.95 at 0.5 and 0.94 at 0.7, and on the floor support 0.95. The interval for `alpha` is a profile-likelihood interval. It is first-order by default: on King's support it covers close to its level in simulations from the CPB, and on the floor support about 0.90 to 0.94, with most misses on the upper side (see `calibrate_alpha()`). It is 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. Coefficient intervals need `se = "bootstrap"` at fit time; the interval for `"alpha"` does not.
`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 *Matched count regressions: Poisson, negative binomial, COM-Poisson, generalized Poisson, gamma-count, double Poisson, hyper-Poisson, exponentially weighted Poisson*

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", "hyperpois", "ewp3", "ewp2"),
  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"`, `"hyperpois"`, `"ewp3"`, `"ewp2"` (see Families).
`truncated`	Logical; if `TRUE`, fit the zero-truncated form (requires all `Y >= 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 `cpb_fe()`). 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. $\log(\text{exposure})$ 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 `nobs()` returns the weight total. Survey (probability) weights call for `se = "robust"` or `se = "cluster"`.

`se` Standard errors: `"analytic"` (inverse information, default), `"robust"` (heteroskedasticity-consistent sandwich), `"cluster"` (cluster-robust; needs `cluster`), or `"none"`. A robust or clustered covariance protects inference about the parameters of the best-fitting member of the family; the quantities that rest on the whole distribution (predicted probabilities, the mean of a truncated or two-part model) are still those of the fitted family. King and Roberts (2015) read a large gap between the robust and the model-based standard errors as evidence of a misspecification to fix; `summary()` reports their ratio. For the Poisson the coefficients are consistent for the conditional mean whenever $E(Y \mid x) = \exp(x'\beta)$, whatever the distribution (Gourieroux, Monfort and Trognon 1984; Wooldridge 1999), and the robust or clustered errors are then valid for them: the gap shows that the variance is not the Poisson's, not that the mean is wrong. That makes the robust Poisson the benchmark for mean effects, against which the families' own mean effects (`ame()`) can be compared; a family that fits the distribution better estimates them consistently only if it is the true model.

`cluster` Optional cluster identifier (a column name in `data` or a vector aligned to its rows) for `se = "cluster"`; supplying it selects `se = "cluster"` unless `se` is given explicitly.

Value

An object of class `"count_reg"`. The component `$theta` holds the dispersion parameter on its natural scale (the negative-binomial size, the COM-Poisson ν, the generalized-Poisson λ, the gamma-count α, the double-Poisson θ, the hyper-Poisson γ, the EWP's (β_1, β_2) or β; `NA` for the Poisson); `$fitted.values` and `$residuals` are on the mean scale (the conditional mean E(Y | Y >= 1) for a zero-truncated fit), as `fitted()` and `residuals()` return them, and `$mu` is the family's natural parameter `exp(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, $\log \mathrm{E}(Y) = x'\beta$, 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 $\mu = \exp(x'\beta)$, the mean for $\lambda \geq 0$, with $\mathrm{Var}/\mathrm{Mean} = 1/(1-\lambda)^2$; $\lambda < 0$ is underdispersion, on a finite support that the pmf is renormalized over, and the mean then departs from μ where that support is short (small μ at strong underdispersion: about 1.19μ at $\lambda = -0.6$, $\mu = 0.5$); the exact mean is reported (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).
- `"hyperpois"`: the hyper-Poisson of Bardwell and Crow (1964) in the mean parameterization of Sáez-Castillo and Conde-Sánchez (2013), mean $\mu = \exp(x'\beta)$, $\gamma < 1$ underdispersed, with one γ for all observations: their constant-dispersion model hP(μ_i, γ). Their model with covariates in the dispersion, hP(μ_i, γ_i), is fit by `DGLMExtPois::glm.hP()`. Its underdispersion is bounded: as $\gamma \to 0$ the variance falls only to $\mu - 1$ for $\mu \geq 1$ and $\mu(1-\mu)$ below, so Var/Mean is at least 0.5 at a mean of 2 and 0.9 at 10 (see hyperpois-distribution).
- `"ewp3"` (alias `"ewp"`): the three-parameter exponentially weighted Poisson of Ridout and Besbeas (2004), a Poisson with centrality $\lambda = \exp(x'\beta)$ reweighted by a two-sided exponential in the distance from it; `$theta` holds the two dispersion parameters (β_1, β_2), one per tail, positive values underdispersed (see ewp-distribution). `"ewp2"` is its two-parameter form, $\beta_1 = \beta_2 = \beta$. The coefficients are effects on the log centrality, not the log mean; the exact mean is reported.

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, the hyper-Poisson and the EWP evaluate a normalizing sum per observation (and the mean parameterizations solve a root per observation), so they take seconds at a few hundred rows and minutes at several thousand. The EWP's sum lengthens with a heavy upper tail (a negative upper weight), and its standard errors sum over each observation's support. Rows that repeat (the same count, covariates and offset) are evaluated once with their total weight, so an intercept-only or few-category design costs as many evaluations as it has distinct rows. `se = "robust"` and `"cluster"` add numerical scores at the same cost per parameter.

References

King, G. and Roberts, M. E. (2015). How robust standard errors expose methodological problems they do not fix, and what to do about it. *Political Analysis*, 23(2), 159-179. 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. Shmueli, G., Minka, T. P., Kadane, J. B., Borle, S. and Boatwright, P. (2005). A useful distribution for fitting discrete data: revival of the Conway–Maxwell–Poisson distribution. *Journal of the Royal Statistical Society C*, 54(1), 127-142. Sellers, K. F. and Shmueli, G. (2010). A flexible regression model for count data. *Annals of Applied Statistics*, 4(2), 943-961. Bardwell, G. E. and Crow, E. L. (1964). A two-parameter family of hyper-Poisson distributions. *Journal of the American Statistical Association*, 59(305), 133-141. Sáez-Castillo, A. J. and Conde-Sánchez, A. (2013). A hyper-Poisson regression model for overdispersed and underdispersed count data. *Computational Statistics &*

Data Analysis, 61, 148-157. Ridout, M. S. and Besbeas, P. (2004). An empirical model for underdispersed count data. *Statistical Modelling*, 4(1), 77-89. Gourieroux, C., Monfort, A. and Trognon, A. (1984). Pseudo maximum likelihood methods: applications to Poisson models. *Econometrica*, 52(3), 701-720. Wooldridge, J. M. (1999). Distribution-free estimation of some nonlinear panel data models. *Journal of Econometrics*, 90(1), 77-97.

See Also

cpb(), gec(), compare_models(), dispersion_test(), dispersion_profile(), hurdle_count(), zi_count()

Examples

```
set.seed(1); n <- 400; x <- rnorm(n)
y <- rgammacount(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 = "gammacount")
m
compare_models(poisson = count_reg(y ~ x, d, family = "poisson"), gammacount = m)
```

---

cpb *Fit a Continuous Parameter Binomial (CPB) regression*

Description

Fits the underdispersed continuous parameter binomial model of King (1989): a binomial count with a continuous number of trials, $\lambda_i/(1-\alpha)$, and success probability $1-\alpha$, so that the conditional variance is close to a fraction of the conditional mean, $\mathrm{Var}(Y \mid x) \approx \alpha\, \mathrm{E}(Y \mid x)$ with $0 < \alpha < 1$, and each observation has an endogenous ceiling, $\lceil \lambda_i/(1-\alpha) \rceil$. 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,
```

```
  support = c("ceiling", "floor")
)
```

Arguments

formula	A model formula.
data	A data frame.
truncated	Logical; if TRUE (default) fit the zero-truncated CPB (requires all Y >= 1); if FALSE fit the untruncated CPB on Y >= 0. **Note the sibling default differs:** cpb_fe() defaults to truncated = FALSE, because its typical call fits a whole panel including zeros, while cpb()'s typical call fits the positive counts of a zero-inflated outcome. State truncated 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 (a Hessian-based sandwich is unreliable here; see Details). Robust errors protect inference about the best-fitting CPB's parameters. The mean effects computed from them are the data's only when the CPB is the data's process: on underdispersed counts from another process (binomial counts with varying trials, a COM-Poisson) the CPB's average marginal effects can be off by several percent at any sample size, while the robust Poisson's (count_reg() with se = "robust") are consistent whenever the mean is log-linear; the two fits' ame() side by side is the check. Nor do robust errors protect the CPB's distributional claims (its ceiling, its predicted probabilities) when the CPB is wrong: King and Roberts (2015) read a gap between robust and model-based errors as a sign of misspecification to fix. ceiling_exceedance(), rootogram(), pit_hist() and compare_models() check those claims.
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 nobs() returns the weight total.
cores	Number of worker processes for the bootstrap (default 1). With cores > 1 the replicates run on a socket cluster whose random streams are seeded from the calling session, so set.seed() 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 evaluated support. On King's support the pmf is summed over the points within 36 log units of its mode, and a distribution with more such points than max.support is treated as infeasible (the guard caps the work for

very large rates, binding at 500 where $\alpha\lambda$ exceeds about 880; it does not bind as `alpha` approaches 1). On the floor support parameter values whose implied ceiling exceeds it are infeasible (they lie in the `alpha` near 1 region where that support explodes), and a fit whose largest ceiling reaches the guard is flagged with a warning and in `$support_binding`, because `alpha` is then bounded by the guard rather than by the data. The default `NULL` sets it to `max(500, 10 * max(y))`.

`maxit, reltol` Optimizer controls passed to `stats::optim()`.

`support` The CPB's support. `"ceiling"` (the default) is King's (1989) CPB as Winkelmann, Signorino and King (1995) normalize it, on $0, \ldots, \lceil \lambda_i/(1-\alpha) \rceil$; `"floor"` is the package's definition before version 0.2.0, on $0, \ldots, \lfloor \lambda_i/(1-\alpha) \rfloor$, kept so that earlier results reproduce. The two coincide when the ceiling is an integer.

Details

The rate is modeled log-linearly, $\lambda_i = \exp(x_i'\beta)$. The probabilities follow the Katz recursion $P(Y = k+1)/P(Y = k) = (\lambda_i - k(1-\alpha))/(\alpha(k+1))$ on the support $0, \ldots, \lceil \lambda_i/(1-\alpha) \rceil$, King's normalization as Winkelmann, Signorino and King (1995) give it. The CPB is therefore the generalized event count model of `gec()` at $\delta = \alpha$, and the two functions return the same fit wherever the GEC's estimate of δ is below one, given the same `truncated` (`cpb()` defaults to the zero-truncated model, `gec()` to the untruncated one). Where $\lambda_i/(1-\alpha)$ is an integer the distribution is the binomial with that many trials, with mean λ_i and variance $\alpha\lambda_i$; elsewhere its mean is at least λ_i. In King's notation `alpha` is σ^2, the variance-to-mean ratio at an integer ceiling; Winkelmann, Signorino and King write the ceiling $\lambda_i/(1-\alpha)$ as α and $1-\alpha$ as p. Because the support moves with the parameters, the log-likelihood is continuous with kinks wherever an observation's ceiling crosses an integer, and it tends to the Poisson log-likelihood as $\alpha \to 1$; a fit that reaches that limit ($\hat{\alpha} > 0.999$) warns, and `dispersion_test()` gives it the boundary value of the test. The maximizer is found by a deterministic multistart, Nelder-Mead from the Poisson solution at `alpha.start` and seven further dispersions, the best polished by Nelder-Mead with feasibility-repaired restarts and a final BFGS pass, 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 neighboring 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()`. 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 likelihood is not regular at the crossings. Just above an integer the top support point carries probability proportional to the ceiling's distance ϵ past it, so the information an observation carries about its rate grows like $1/\epsilon$ (about $0.28/\epsilon$ for $\log\lambda$ at $\alpha = 0.5$ and a ceiling of $3+\epsilon$, about $1.8/\epsilon$ at $\alpha = 0.3$; King 1989, footnote 7, notes that asymptotic normality needs non-integer ceilings). With continuous covariates the ceilings' fractional parts are spread and the expected information per observation is unbounded, so standard errors from the numerical Hessian are unreliable (at small `alpha` they fall well below the sampling variability of the estimates), and inference uses a cold-multistart bootstrap for the coefficients and a profile-likelihood interval for α. In simulations from King's CPB at n = 300, the bootstrap's 95% interval for a slope covers 0.91 at alpha 0.3, 0.95 at 0.5 and 0.94 at 0.7 (0.92 at alpha 0.3 with n = 1,000); on the floor support it covers 0.95. The interval

for α is first-order by default and calibrated by parametric bootstrap after `calibrate_alpha()` (see `alpha_confint()`).

`support = "floor"` gives the package's definition before version 0.2.0, on $0, \ldots, \lfloor \lambda_i/(1-\alpha) \rfloor$, whose mean is at most λ_i. Its 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 fit then adds a scan of the breakpoints in `alpha` next to the estimate; two parameterizations of the same design (say, treatment and sum contrasts) can settle on different teeth of the surface, with log-likelihoods differing by the order of the jumps; and ceilings above `max.support` are infeasible, so `alpha` can be bounded by that guard rather than by the data.

Runtime: a fit with `se = "none"` takes about a second at 500 rows and four at 2,000 on one core, the multistart and the profile trace included (about four and fifteen on the floor support). The bootstrap replicates are maximized by the multistart without the profile trace (about 0.1 seconds each at 500 rows; on the floor support a replicate can therefore 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: From restrictive assumptions to a generalized estimator. *American Journal of Political Science*, 33(3), 762-784. Winkelmann, R., Signorino, C. S. and King, G. (1995). A correction for an underdispersed event count probability distribution. *Political Analysis*, 5, 215-228. King, G. and Roberts, M. E. (2015). How robust standard errors expose methodological problems they do not fix, and what to do about it. *Political Analysis*, 23(2), 159-179.

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: King's CPB (the default) puts no probability above $\lceil \lambda/(1-\alpha) \rceil$, the floor support (`support = "floor"`, the definition before version 0.2.0) none above $\lfloor \lambda/(1-\alpha) \rfloor$. Complements the simulator `rcpb()`.

Usage

```
dcpb(
  x,
  lambda,
  alpha,
  truncated = FALSE,
  log = FALSE,
  support = c("ceiling", "floor")
)

pcpb(
  q,
  lambda,
  alpha,
  truncated = FALSE,
  lower.tail = TRUE,
  log.p = FALSE,
  support = c("ceiling", "floor")
)

qcpb(
  p,
  lambda,
  alpha,
  truncated = FALSE,
  lower.tail = TRUE,
  log.p = FALSE,
  support = c("ceiling", "floor")
)
```

Arguments

`x`, `q`	Vector of quantiles (non-negative integers).
`lambda`	Rate parameter (scalar or vector, recycled). It is the mean where `lambda / (1 - alpha)` is an integer; otherwise the mean lies above it on King's support and below it on the floor support.

alpha	Shape parameter in (0, 1).
truncated	If TRUE, use the zero-truncated CPB. On the floor support, 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); King's support always contains 1.
log, log.p	If TRUE, probabilities are given as log.
support	The CPB's support, "ceiling" (King's CPB, the default) or "floor"; see cpb().
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)
```

---

cpb_fe *CPB regression with high-dimensional unit fixed effects*

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. On King's support (the default) each unit's objective is continuous in its intercept: when the intercept carries observation t's ceiling across an integer the support point that enters carries mass growing from zero, so the crossing is a kink, and the objective falls to minus infinity at the feasibility floor, where some count would exceed $\lceil \lambda_{it}/(1-\alpha) \rceil$. As for the GEC (the same distribution at $\delta = \alpha$; see gec_fe()), the smooth part is unimodal and the kinks only lower the slope, so the inner search is a 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 outer optimizer is a deterministic multistart: BFGS with that gradient from the Poisson slopes and two dispersion starts, a Nelder-Mead polish, and two perturbed restarts.

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"),
  time = NULL,
  support = c("ceiling", "floor")
)
```

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:** cpb_fe() defaults to FALSE (whole-panel fits including zeros), while cpb() 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 se = "bootstrap".
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 cpb().
cores	Worker processes for the bootstrap; see cpb().
max.support	Guard on the maximum evaluated support; NULL (default) sets max(500, 10 * max(y)). See cpb().

`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).
`time`	Name of the column of `data` that orders each unit's rows in time. Required by `bias_correct = "jackknife"`, which splits each unit's series into its first and second halves (and checks the residuals for a time trend); the fit itself does not depend on it.
`support`	The CPB's support, `"ceiling"` (King's CPB, the default) or `"floor"` (the definition before version 0.2.0); see `cpb()`.

Details

On the floor support (`support = "floor"`, the definition before version 0.2.0) 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, which the perturbed restarts of the outer multistart are there to escape; 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.

Short panels: the dispersion parameter `alpha` and the fixed effects are subject to the incidental-parameters bias of nonlinear fixed-effects estimation. The bias in `alpha` 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 carry a smaller bias of the same order, which can matter in very short panels (a few positive counts per unit, as in a zero-truncated intensity). `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 (each unit's rows ordered by `time`, which the correction requires) and the corrected estimate is `2 * full - mean(halves)`. In the package's fixed-effects bias Monte Carlo (150 units, alpha = 0.5) the alpha bias at T = 6/10/20/40 falls from -0.122/-0.071/-0.036/-0.018 (T times the bias is -0.71 to -0.73 throughout) to +0.008/+0.004/+0.000/-0.001. On the floor support the bias is larger (-0.153/-0.100/-0.057/-0.033), and the correction leaves 23 to 30 percent of it. With correction on, `coefficients` and `alpha` 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. The covariate coefficients are corrected too: unlike the Poisson's, whose slopes carry no first-order bias, the CPB's can carry one (see Short panels above). The correction leaves the estimator's asymptotic variance unchanged

(Dhaene and Jochmans 2015) but adds variance in short panels. In the package's simulations (40 units, alpha = 0.5, a slope with no bias to remove) the corrected slope's standard deviation was 19 percent above the uncorrected slope's at T = 6 and 13 percent at T = 12, while alpha's bias fell from -0.13 and -0.06 to +0.011 and +0.004; where the slopes' own bias is small next to their standard error, the uncorrected fit (`bias_correct = "none"`) estimates them more precisely. With `se = "bootstrap"` each replicate is corrected the same way (its two half-panel refits; the validity gate below is a check on the data, made once), so the standard errors include that variance and the bootstrap takes about three times as long. In the same simulations the corrected slope's 95% interval covered 0.91 at T = 6 and 0.94 at T = 12 (0.85 and 0.89 with uncorrected replicates). On the floor support the replicates are not corrected, and the interval is centered on the corrected estimate.

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 full fit's residuals trend with time (correlation with the within-unit time index, $p < 0.01$), the two half-panel dispersion estimates disagree by more than 2.5 times the noise of five placebo splits of the units (and by more than 0.05 of the estimate's scale), or the corrected dispersion leaves its feasible space. 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 9 to 13 percent of genuinely time-homogeneous panels (a conservative nuisance; the returned fit is exactly the ordinary maximum-likelihood estimate) while catching 98.5 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 out of sample it penalizes over-parameterization, so it is the appropriate 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. Each fold is scored as score() scores newdata: a unit, or a level of any factor, that the training folds lack enters at the reference unit, and a unit a two-part fit holds at a limit is scored at that limit.

Usage

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

Arguments

fitfun	A function of one argument (a training data frame) returning a fitted underdisp model, e.g. function(d) hurdle_cpb(y ~ x, d, participation = ~ z, fe = "unit", se = "none").
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 nrow(data) 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 w counts as w held-out observations (the folds assign whole rows). The fold fits see the weights only through fitfun, so pass them there as well.

`offset` Optional offset (log scale) of the held-out rows: a column name of `data` or a numeric vector with one value per row, split with the folds. It is needed when `fitfun` gives its fit the offset as a vector. The fold fits see the offset only through `fitfun`.

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 `alpha` and `delta`; the implied curves are the exact moments of the fitted pmf, so the renormalized CPB's ratio rises above `alpha` where the implied ceiling is small: by up to about a third at `alpha = 0.3` where $\lambda/(1-\alpha)$ lies between one and three, and toward one, the Poisson's value, as the rate falls toward zero; on the floor support it dips below `alpha` instead), 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 hyper-Poisson's cannot fall below $1-1/\mu$ ($1-\mu$ below a mean of one), so it approaches one as the mean grows, and the EWP's follows its two tail weights. The profile therefore shows which variance function the data follow (a fitted distribution does not by itself identify the process behind it), where a family's 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` *Regression-adjusted test of equidispersion*

Description

Tests whether a fitted count model's dispersion departs from the Poisson, conditional on the covariates. 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), `gamma = 1` (hyper-Poisson), `beta1 =`

beta2 = 0 or beta = 0 (the EWP), and 1/theta = 0 (negative binomial). When that value is interior to the parameter space the likelihood-ratio statistic is asymptotically χ^2_1 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 alpha lives in (0, 1); the negative binomial, whose overdispersion parameter is non-negative) the asymptotic null distribution is the $\frac{1}{2}\chi^2_0 + \frac{1}{2}\chi^2_1$ 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 alpha approaches 1, the statistic cannot be negative in principle; when the fit reaches that limit (King's CPB, whose likelihood approaches the Poisson's continuously) or the floor support's max.support guard stops alpha short of it, 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, which does not depend on the direction in which the asymptotic reference errs. For the CPB it errs toward rejection: in the package's simulations from the Poisson the asymptotic boundary mixture rejected 7 to 10 percent of 5% tests with 60 to 200 observations, and 54 to 92 percent with unit fixed effects, whose intercepts bias the dispersion estimate toward underdispersion by an amount of order 1/T. The same rule bootstraps the negative binomial's boundary test. 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% test above 6%. The shift is derived for one tested parameter, so the EWP3's joint test of its two tails takes the bootstrap by default: in the package's simulations from the Poisson its asymptotic χ^2_2 rejected 6.4 to 7.5 percent of 5% tests with one or two mean parameters on 200 to 500 observations, and 8.0 percent with ten on 400. The bootstrap 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 5% test's first-order 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 `mu` and dispersion `theta`, 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) `mu`. 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))
```

---

ewp-distribution *Exponentially weighted Poisson distribution functions*

Description

Density, distribution, quantile, and random generation for the exponentially weighted Poisson of Ridout and Besbeas (2004), in its three-parameter form (EWP3) and, with beta2 = beta1, its two-parameter form (EWP2): a Poisson with centrality lambda reweighted by a two-sided exponential in the distance from it,

$$P(Y = y) \propto w_y e^{-\lambda} \lambda^y / y!, \quad w_y = e^{-\beta_1(\lambda - y)} \ (y \leq \lambda), \quad w_y = e^{-\beta_2(y - \lambda)} \ (y > \lambda).$$

beta1 and beta2 greater than zero thin the lower and upper tails (underdispersion), values below zero thicken them, and beta1 = beta2 = 0 is the Poisson. lambda is not the mean; the normalizer and the exact mean and variance are sums over the support (no truncation of the normalizing sum at a fixed count), and the exact mean is what count_reg(family = "ewp3") and count_reg(family = "ewp2") report through predict() and fitted().

Usage

```
dewp(x, lambda, beta1, beta2 = beta1, log = FALSE)

pewp(q, lambda, beta1, beta2 = beta1, lower.tail = TRUE, log.p = FALSE)

qewp(p, lambda, beta1, beta2 = beta1, lower.tail = TRUE, log.p = FALSE)

rewp(n, lambda, beta1, beta2 = beta1)
```

Arguments

x, q	Vector of quantiles (non-negative integers).
lambda	Centrality parameter (scalar or vector, recycled).
beta1, beta2	Lower- and upper-tail dispersion parameters (scalars); beta2 defaults to beta1, the two-parameter EWP.
log, log.p	Return log probabilities.
lower.tail	If TRUE (default), $P(X \leq x)$.
p	Vector of probabilities.
n	Number of draws.

Value

dewp a density, pewp a CDF, qewp a quantile, rewp a numeric vector of count draws.

References

Ridout, M. S. and Besbeas, P. (2004). An empirical model for underdispersed count data. *Statistical Modelling*, 4(1), 77-89.

See Also

count_reg()

Examples

```
dewp(0:6, lambda = 3, beta1 = 1, beta2 = 1)
var(rewp(2000, lambda = 3, beta1 = 1, beta2 = 1))
```

`first_difference` *First difference for a CPB fit*

Description

The effect on a quantity of interest of moving one covariate `from` one value `to` another, holding the other covariates at their means (or at the values `at` gives 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,
  at = NULL,
  ...
)
```

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 | Y >= 1)), `"ceiling"` (lambda/(1-alpha)), or `"prob"` (P(Y = y)).
`y`	The count value for `quantity = "prob"`.
`level`	Confidence level (default 0.95).
`at`	Optional named numeric vector fixing other model-matrix columns at given values instead of their means, e.g. `at = c(z = 1, regionB = 0)` (a factor's dummies are columns, so a reference level sets them to 0).

Details

Every `first_difference()` method in the package takes its interval by one rule. When the fit carries bootstrap draws (`se = "bootstrap"` at fit time), the quantity is computed at every draw, jointly over the coefficients and the dispersion parameter (and, for a fixed-effects fit, the reference unit's effect), and the percentile interval is taken; the draws of a fixed-effects fit are first recentered on the reported estimate, because its replicates share the incidental-parameters bias. Otherwise the interval is the delta method's over the fit's full covariance, the dispersion parameter included (the matched count families' analytic, robust or clustered covariance; a hurdle's two margins, which are independent). A fit with neither gets the point estimate with `method = "none"`.

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 = 100)
first_difference(fit, "x", from = -1, to = 1, quantity = "mean")
```

`first_difference.gec` *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 (or at the values `at` gives them). The interval follows the package's one rule (see `first_difference()`): percentile intervals from the fit's stored bootstrap draws for the GEC, fixed-effects CPB and zero-inflated GEC fits, the delta method over the full covariance, the dispersion parameter included, for the matched count families and the hurdles' margins. For `count_reg` this is a single number; for the two-part models it is reported on each margin: the binary stage's probability (participation, or the structural-zero probability), the count stage's mean, and the marginal mean, their product, each at `from` and `to`. The first two are different quantities and their changes do not add up to the marginal change, which is $\Delta(pm) = \Delta p\, m_0 + p_0 \Delta m + \Delta p\, \Delta m$.

Usage

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

## S3 method for class 'gec_fe'
```

```
first_difference(object, variable, from, to, level = 0.95, at = NULL, ...)

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

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

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

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

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

## S3 method for class 'zi_count'
first_difference(
  object,
  variable,
```

```
  from,
  to,
  level = 0.95,
  stage = c("both", "participation", "intensity", "zero", "count"),
  at = NULL,
  ...
)
```

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 of the intervals.
at	Optional named numeric vector fixing other model-matrix columns at given values instead of their means (see first_difference()); in a two-part model a column of both equations is fixed in both.
...	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 on the participation (extensive) margin, the intensity (intensive) margin, and the marginal expected count, holding other covariates at their means (or at the values at gives them). The two margins are reported side by side; their changes do not add up to the marginal change (see first_difference.count). With B = 0 the intervals follow the package's one rule (see first_difference()): the stored bootstrap draws of a zi_cpb fit, and for a hurdle the delta method over its two margins' covariances (the participation model's, and the intensity's stored bootstrap draws). B > 0 refits the whole model on resampled data instead.

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,
  at = NULL,
  ...
)

## 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,
  at = NULL,
  ...
)
```

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	Refit-bootstrap replicates for percentile intervals; 0 (default) uses the fit's own inference instead (see Details). For zi_cpb the refit bootstrap is slow (each replicate refits the mixture).
level	Confidence level.
data	The original data frame; required when B > 0. Each replicate refits the model with the fit's own settings (fixed effects, link, offset, weights, support, and the like), resampling whole fixed-effects units (or the fit's clusters) when it has them; an offset or weights given to the fit as a vector must then be a column of data instead.
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()`.
`at`	Optional named numeric vector fixing other model-matrix columns at given values instead of their means (see `first_difference()`).
`...`	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` *Fitted means for the two-part and fixed-effects classes*

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

`object`	A fitted model from this package.
`...`	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 `alpha` and rate `alpha * mu`, so that `mu` 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

`x`, `q`	Vector of quantiles (non-negative integers).
`mu`	Rate parameter (scalar or vector, recycled).
`alpha`	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

`dgammacount` a density, `pgammacount` a CDF, `qgammacount` a quantile, `rgammacount` 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

```
dgammacount(0:5, mu = 3, alpha = 2)
var(rgammacount(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 `delta` (the variance-to-mean ratio) is estimated freely and spans underdispersion (`delta` < 1, a finite-support member, the continuous parameter binomial of `cpb()` at `alpha = delta`), 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; wherever its estimate of `delta` is below one the two return the same fit. The Katz recursion delivers exact first and second moments, summed over the support (finite for `delta` < 1, where they are the correction of Winkelmann, Signorino and King), 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 `delta` 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 = NULL,
  maxit = 20000,
  reltol = 1e-08
)
```

Arguments

formula	A model formula.
data	A data frame.
truncated	Logical; if TRUE, fit the zero-truncated GEC (all Y >= 1), the intensity model of hurdle_gec().
se	Coefficient inference: "none" (default; fast) or "bootstrap".
B	Bootstrap resamples when se = "bootstrap".
cluster	Optional cluster identifier (a column name in data or a vector) for a cluster/block bootstrap; see cpb().
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 evaluated support: the GEC's pmf is evaluated up to that many points (on an overdispersed fit, the guard binds where the tail beyond it carries mass; see dgec()). The default NULL sets it to max(500, 10 * max(y)), as for cpb().
maxit, reltol	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 lambda and Katz dispersion delta (< 1 underdispersed, = 1 Poisson, > 1 overdispersed). On an unbounded support (delta >= 1) the mean is lambda and the variance-to-mean ratio is delta 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

x, q	Vector of quantiles (non-negative integers).
lambda	Rate parameter (scalar or vector, recycled).
delta	Katz dispersion parameter (scalar).
max.support	Guard on the evaluated support.
log, log.p	Return log probabilities.
lower.tail	If TRUE (default), $P(X \leq x)$.
p	Vector of probabilities.
n	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..ceiling(mu/(1-delta))`. Runtime: about one and a half times that of `cpb_fe()` on the same panel (the Katz recursion runs the whole support): on the positive counts of the bundled peacekeeping panel, 2,602 rows in 146 units, `gec_fe()` takes about two and a half minutes and `cpb_fe()` about one and a half, and the two return the same fit.

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"),
  time = NULL
)
```

Arguments

`formula`	A model formula for the covariates only (no unit factor, no intercept; the fixed effects absorb it).
`data`	A data frame.
`fe`	Name of the column holding the unit identifier.
`se`	Inference for the covariate coefficients: `"none"` (default) or `"bootstrap"`, a pairs/cluster bootstrap over units.
`B`	Bootstrap resamples when `se = "bootstrap"`.
`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 evaluated support; `NULL` (default) sets `max(500, 10 * max(y))`. See `gec()`.
`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).
`time`	Name of the column of `data` that orders each unit's rows in time; required by `bias_correct = "jackknife"` (see `cpb_fe()`).

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 carry a smaller bias of the same order. `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 * full - mean(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). As there, the covariate coefficients are corrected too, and with `se = "bootstrap"` each replicate is corrected the same way, so the standard errors include the correction's own variance.

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 `mu` and dispersion `lambda` 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 `mu` 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

`x`, `q`	Vector of quantiles (non-negative integers).
`mu`	Mean parameter (scalar or vector, recycled).
`lambda`	Dispersion parameter in (-1, 1) (scalar).
`log`, `log.p`	Return log probabilities.
`lower.tail`	If `TRUE` (default), $P(X \leq x)$.
`p`	Vector of probabilities.
`n`	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(rgenpois(2000, mu = 3, lambda = -0.3)) / 3   # about 1/(1.3)^2
```

`glance.cpb` *Glance at a fitted model (broom method)*

Description

One row of fit statistics for every model class of the package: `logLik`, `AIC`, `BIC`, `df` and `nobs` (the weight total of a weighted fit), the dispersion parameter as `dispersion` with its name (`dispersion_parameter`) and the direction its estimate points (`dispersion_direction`; the families' parameters point different ways, so the CPB's `alpha` below 1 and the gamma-count's `alpha` above 1 are both underdispersion), `coef_scale`, the scale of the count equation's coefficients (see `tidy.cpb()`), and the class's own columns (`alpha` for the CPB family, `family` for the matched count families, `n.units` for the fixed-effects fits).

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 fitted model from this package.
...	Unused.

Value

A one-row data frame of fit statistics.

---

hurdle_count *Hurdle count regression for the matched count families*

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", "hyperpois", "ewp3", "ewp2"),
  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 ~ 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). The participation slopes carry the dummy logit's incidental-parameters bias in short panels (see hurdle_cpb()). The participation dummy of a unit that never participates is not counted in the degrees of freedom (see compare_models(), Degrees of freedom). In new data such a unit, and one whose every count is positive, get the participation probability at their limits (zero and one; with one part_fe column), and a unit either equation has not seen (the intensity never sees a unit with no positive count) enters that equation at its reference unit, as in hurdle_cpb(); the participation equation is checked for separation (part_separation).
link	Link for the participation model: "logit" (default), "probit", or "cloglog". Positive participation coefficients raise the probability of a *positive count* (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; see count_reg(). They apply to both margins: the participation equation's standard errors are model-based, robust or cluster-robust with the intensity's.

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

formula	Intensity model formula (y ~ x).
data	A data frame.
participation	Optional one-sided formula (~ z) for the participation (hurdle) model; defaults to the intensity model's right-hand side.
fe	Optional column name for unit fixed effects in the intensity model, absorbed by a concentrated likelihood (cpb_fe()). This is how the within-unit underdispersion is recovered.
part_fe	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, and the dummy logit's coefficients carry the incidental-parameters bias of order 1/T (in short panels the slopes are inflated, roughly toward twice their value at T = 2); the split-panel jackknife (bias_correct) corrects only the intensity. The hurdle's likelihood separates its two parts, so for the logit link a conditional logit on the same data (survival::clogit()) gives participation slopes free of that bias, and an analytical correction (the **bife** package, for the logit and the probit) reduces it to second order; the hurdle's predictions and quantities of interest use the dummy logit's unit effects. On King's support the dummy of a unit that never participates, at minus infinity, is not counted in the degrees of freedom (see compare_models(), Degrees of freedom). In new data (predict.hurdle_cpb(), score(), cv_score()) such a unit's rows get the participation probability at that limit, zero, and those of a unit whose every count is positive (part_one_units) one; a unit the fit has not seen, or every row of newdata without the part_fe column, enters at the reference unit of the units the data identify. The participation equation is checked for separation by zi_cpb()'s rule (Separation; component part_separation). On the floor support these dummies are counted in the degrees of freedom, new rows of these units are evaluated at the dummies' fitted values, and an unseen unit enters at the mean of all the unit dummies.

link	Link for the participation model: "logit" (default), "probit", or "cloglog". Positive participation coefficients raise the probability of a positive count.
offset	Optional offset on the log-mean scale for the intensity (an exposure): a numeric vector or the name of a column in data.
cluster	Optional cluster identifier (a column name in data or a vector). The participation equation's standard errors are then cluster-robust (the sandwich, with the G/(G - 1) factor), and with se = "bootstrap" the intensity's bootstrap resamples whole clusters; see cpb().
se	Inference for the intensity coefficients: "none" or "bootstrap".
B	Bootstrap replicates when se = "bootstrap".
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().
...	Passed to cpb() or cpb_fe(), including support (King's CPB, "ceiling", by default; see cpb()).

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 = NULL
)
```

Arguments

formula	Intensity formula, y ~ 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), whose slopes carry the dummy logit's incidental-parameters bias in short panels (see hurdle_cpb()). The dummy of a unit that never participates is not counted in the degrees of freedom (see compare_models(), Degrees of freedom). In new data such a unit, and one whose every count is positive, get the participation probability at their limits (zero and one), and an unseen unit enters at the reference unit, as in hurdle_cpb(); the participation equation is checked for separation (part_separation).
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): cluster-robust standard errors for the participation equation, and the cluster bootstrap of the intensity with se = "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 evaluated support of the intensity; NULL (default) sets max(500, 10 * max(y)). See gec().

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)
```

---

hyperpois-distribution

Hyper-Poisson distribution functions

Description

Density, distribution, quantile, and random generation for the hyper-Poisson distribution of Bardwell and Crow (1964) in the mean parameterization of Saez-Castillo and Conde-Sanchez (2013): mean mu and dispersion gamma > 0, with $P(Y = y) = \lambda^y/((\gamma)_y\, {}_1F_1(1;\gamma;\lambda))$, $(\gamma)_y = \Gamma(\gamma + y)/\Gamma(\gamma)$, and the rate λ the unique solution of $E(Y) = \mu$. gamma < 1 gives underdispersion, gamma = 1 the Poisson, gamma > 1 overdispersion. The underdispersion is bounded: as gamma -> 0 the distribution tends to 1 + Poisson(mu - 1) for mu > 1 and to a Bernoulli(mu) below, so the variance is at least mu - 1 (mu >= 1) or mu (1 - mu) (mu < 1), and Var/Mean approaches one as the mean grows (0.5 at a mean of 2, 0.9 at 10). The normalizer and the moments are exact sums over the support.

Usage

```
dhyperpois(x, mu, gamma, log = FALSE)

phyperpois(q, mu, gamma, lower.tail = TRUE, log.p = FALSE)

qhyperpois(p, mu, gamma, lower.tail = TRUE, log.p = FALSE)

rhyperpois(n, mu, gamma)
```

Arguments

x, q	Vector of quantiles (non-negative integers).
mu	Mean (scalar or vector, recycled).
gamma	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

dhyperpois a density, phyperpois a CDF, qhyperpois a quantile, rhyperpois a numeric vector of count draws.

References

Bardwell, G. E. and Crow, E. L. (1964). A two-parameter family of hyper-Poisson distributions. *Journal of the American Statistical Association*, 59(305), 133-141. Sáez-Castillo, A. J. and Conde-Sánchez, A. (2013). A hyper-Poisson regression model for overdispersed and underdispersed count data. *Computational Statistics & Data Analysis*, 61, 148-157.

See Also

count_reg()

Examples

```
dhyperpois(0:5, mu = 3, gamma = 0.5)
var(rhyperpois(2000, mu = 3, gamma = 0.5)) / 3
```

---

implied_ceiling *Implied ceiling, and the ceiling at the limits of alpha's interval*

Description

Returns the observation- (or profile-) specific ceiling lambda/(1-alpha), and the ceiling at the fitted rate with alpha at the limits of its interval (see alpha_confint(); calibrated when the fit went through calibrate_alpha()). Those two columns hold the rate at its estimate, so they show how far the dispersion's uncertainty alone moves the ceiling and are not a confidence interval for it; first_difference() with quantity = "ceiling" propagates the coefficients' uncertainty too. The ceiling is the CPB's continuous number of trials. No count above its integer ceiling has positive probability on King's support (the default), and none above its integer floor on the floor support (support = "floor"); that largest feasible count is returned as max_count. Whether held-out counts respect the bound is tested by ceiling_exceedance().

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

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

Details

The ceiling is the fitted distribution's support end. With the rate and `alpha` it is fixed by the CPB's variance-mean link, so it is a bound on the counts only when the CPB is the data's process. On counts with a common hard bound from another process (binomial counts out of six trials, say), the fitted ceiling follows the mean and the dispersion instead: in one simulation it ran from 3 where the mean was low to 12 where it was high, below the true bound in the first case and well above it in the second.

Value

A data frame with `lambda`, `ceiling`, `max_count` (the largest count with positive probability), and `ceiling_at_alpha_lower` / `ceiling_at_alpha_upper`, the ceiling at the fitted rate with `alpha` at the lower and upper limits of its interval (for `cpb` fits, and hurdles with a `cpb` intensity).

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

The ratios are $\exp(\beta)$, ratios of the CPB's rate parameter λ. The CPB's mean equals λ only where $\lambda/(1-\alpha)$ is an integer, and a zero-truncated fit (the `cpb()` default) models $E(Y \mid Y \geq 1)$, so a ratio of expected counts can differ from them; `first_difference()` reports the exact mean contrast.

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 = 100)
irr(fit)
```

---

irr.count_reg *Rate and odds ratios for the matched count families*

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(structural zero) in the second), `ratio` (`"IRR"`, `"OR"`, or the parameter's ratio named below), `estimate`, `lower`, `upper`, and `method` (the interval source; `"none"` with `NA` bounds when no covariance is available). Two-part models stack both equations. The count rows are $\exp(\beta)$, the ratio of the family's natural parameter. That is a ratio of means for the Poisson, the negative binomial, the mean-parameterized COM-Poisson (`"mpcmp"`), the hyper-Poisson, and the generalized Poisson with $\lambda \geq 0$ (`"IRR"`). It is not for the others, whose rows name what it is, with a message: the rate-parameterized COM-Poisson (`"compois"`, `"lambda ratio"`; its mean moves roughly as the rate to the power $1/\nu$, so that $\exp(\beta)$ is roughly the mean ratio to the power ν), the EWP (`"centrality ratio"`), the gamma count (`"long-run rate ratio"`), the double Poisson (`"location ratio"`) and the generalized Poisson below zero (`"mu ratio"`). The CPB and the underdispersed GEC keep `"IRR"`, a ratio of their rate, which their mean follows only approximately (the rate renormalized over a finite support). For all of these, `first_difference()` and `ame()` report 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

`formula`	A model formula.
`data`	A data frame.
`unit`	Column name identifying the panel unit.
`suffix`	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% of state-years are zero, so it is also a natural example for the 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
## among these forty states every major power contributes, so majorpower would
## separate the participation equation (hurdle_cpb() warns when it does)
fit <- hurdle_cpb(contributions ~ lgdppc + milper, data = some,
                  participation = ~ democracy + lgdppc, 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 hump means the data are less dispersed than the fitted model predicts (underdispersed relative to it, as a Poisson fit to underdispersed counts shows), and a U shape means they are more dispersed.

Usage

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

Arguments

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

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 *Predictions from a matched count-family fit*

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,
  se.fit = FALSE,
```

```
  interval = c("none", "confidence"),
  level = 0.95,
  ...
)
```

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.
se.fit, interval, level	Standard errors (se.fit = TRUE) and a level confidence interval (interval = "confidence") of the predicted quantity at the rows of newdata; see predict.cpb().
...	Unused.

Value

A numeric vector. With interval = "confidence", a matrix with columns fit, lwr and upr; with se.fit = TRUE, a list of fit (the vector or the matrix), se.fit and method (each row's interval source). Without se.fit, the source is the attribute method.

---

predict.cpb *Predictions from a CPB fit*

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,
  se.fit = FALSE,
  interval = c("none", "confidence"),
  level = 0.95,
  ...
)
```

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 >= 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), the CPB's continuous number of trials; the largest count with positive probability is its ceiling on King's support and its floor on the floor support), 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` rows: a numeric vector or a column name in `newdata`. By default a fit made with an offset column reads that column from `newdata`, and one made with an offset vector requires this argument. For `newdata = NULL` the fit's own offset is used.
`se.fit`, `interval`, `level`	The uncertainty of the predicted quantity at the rows of `newdata`: `se.fit = TRUE` adds standard errors, and `interval = "confidence"` a `level` confidence interval. They follow the rule of `first_difference()`: the fit's stored bootstrap draws (a percentile interval, and the draws' standard deviation), otherwise the delta method over the fit's covariance, otherwise none. They need `newdata`. A row at a fixed-effects unit's own effect, or at its own zero-equation dummy, has none, since the draws do not carry that parameter; a row of a unit the fit has not seen, or `newdata` without the unit column, enters at the reference unit, which they carry. `ame()` gives average marginal effects.
`...`	Unused.

Value

A numeric vector. With `interval = "confidence"`, a matrix with columns `fit`, `lwr` and `upr`; with `se.fit = TRUE`, a list of `fit` (the vector or the matrix), `se.fit` and `method` (each row's interval source). Without `se.fit`, the source is the attribute `method`.

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"),
  offset = NULL,
  se.fit = FALSE,
  interval = c("none", "confidence"),
  level = 0.95,
  ...
)
```

Arguments

object	A "cpb_fe" object.
newdata	Optional covariate profiles. A row whose fe column names a unit of the fit enters at that unit's estimated effect (as in score() and ceiling_exceedance()); a row of an unseen unit, or every row when newdata has no fe column, enters at the reference unit, the mean fixed effect of the units whose effect the data identify (see Details of cpb_fe()).
type	"response" (the mean; for a zero-truncated fit the conditional mean E(Y | Y >= 1)), "rate" (the CPB rate lambda), "link" (the log rate), or "ceiling".
offset	Optional offset (log scale) for the newdata rows: a numeric vector or a column name in newdata; by default the fit's own offset column is read from newdata (see predict.cpb()).
se.fit, interval, level	Standard errors (se.fit = TRUE) and a level confidence interval (interval = "confidence") of the predicted quantity at the rows of newdata; see predict.cpb().
...	Unused.

Value

A numeric vector. With interval = "confidence", a matrix with columns fit, lwr and upr; with se.fit = TRUE, a list of fit (the vector or the matrix), se.fit and method (each row's interval source). Without se.fit, the source is the attribute method.

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,
  se.fit = FALSE,
  interval = c("none", "confidence"),
  level = 0.95,
  ...
)
```

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 newdata rows: a numeric vector or a column name in newdata; by default the fit's own offset column is read from newdata (see predict.cpb()).
se.fit, interval, level	Standard errors (se.fit = TRUE) and a level confidence interval (interval = "confidence") of the predicted quantity at the rows of newdata; see predict.cpb().
...	Unused.

Value

A numeric vector. With interval = "confidence", a matrix with columns fit, lwr and upr; with se.fit = TRUE, a list of fit (the vector or the matrix), se.fit and method (each row's interval source). Without se.fit, the source is the attribute method.

predict.gec_fe *Predictions from a fixed-effects GEC fit*

Description

Predictions from a fixed-effects GEC fit

Usage

```
## S3 method for class 'gec_fe'
predict(
  object,
  newdata = NULL,
  type = c("response", "link"),
  offset = NULL,
  se.fit = FALSE,
  interval = c("none", "confidence"),
  level = 0.95,
  ...
)
```

Arguments

object	A "gec_fe" object.
newdata	Optional covariate profiles. A row whose fe column names a unit of the fit enters at that unit's estimated effect (as in score() and ceiling_exceedance()); a row of an unseen unit, or every row when newdata has no fe column, enters at the reference unit, the mean fixed effect of the units whose effect the data identify (see Details of cpb_fe()).
type	"response" (the mean) or "link" (the log rate).
offset	Optional offset (log scale) for the newdata rows; see predict.cpb().
se.fit, interval, level	Standard errors (se.fit = TRUE) and a level confidence interval (interval = "confidence") of the predicted quantity at the rows of newdata; see predict.cpb().
...	Unused.

Value

A numeric vector. With interval = "confidence", a matrix with columns fit, lwr and upr; with se.fit = TRUE, a list of fit (the vector or the matrix), se.fit and method (each row's interval source). Without se.fit, the source is the attribute method.

`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,
  se.fit = FALSE,
  interval = c("none", "confidence"),
  level = 0.95,
  ...
)
```

Arguments

`object`	A `"hurdle_count"` object.
`newdata`	Optional data frame of covariate profiles. A fixed-effects unit an equation has not seen, or every row when its column is absent, enters that equation at the reference unit, and a unit at a limit at that limit (see `fe`, `part_fe` in `hurdle_count()`).
`type`	`"response"` (marginal E(Y), zeros included), `"participation"` ($P(Y > 0)$), or `"intensity"` ($E(Y \mid Y > 0)$).
`offset`	Optional offset (log scale) for the intensity when predicting on `newdata`; a numeric vector or a column name in `newdata`.
`se.fit`, `interval`, `level`	Standard errors (`se.fit = TRUE`) and a `level` confidence interval (`interval = "confidence"`) of the predicted quantity at the rows of `newdata`; see `predict.cpb()`.
`...`	Unused.

Value

A numeric vector. With `interval = "confidence"`, a matrix with columns `fit`, `lwr` and `upr`; with `se.fit = TRUE`, a list of `fit` (the vector or the matrix), `se.fit` and `method` (each row's interval source). Without `se.fit`, the source is the attribute `method`.

`predict.hurdle_cpb` *Predict from a hurdle-CPB fit*

Description

Predict from a hurdle-CPB fit

Usage

```
## S3 method for class 'hurdle_cpb'
predict(
  object,
  newdata = NULL,
  type = c("response", "participation", "intensity"),
  offset = NULL,
  se.fit = FALSE,
  interval = c("none", "confidence"),
  level = 0.95,
  ...
)
```

Arguments

`object`	A `"hurdle_cpb"` object.
`newdata`	Data frame of covariate profiles (the intensity and participation covariates). A row of a fixed-effects unit the fit has not seen, or every row when the `fe` or `part_fe` column is absent, enters that equation at the reference unit, and a unit at a limit (no positive count, or no zero) at that limit (see `part_fe` in `hurdle_cpb()`).
`type`	`"response"` for the marginal expected count E(Y) (zeros included), `"participation"` for P(Y > 0), or `"intensity"` for E(Y | Y > 0).
`offset`	Optional offset (log scale) of the intensity for the `newdata` rows: a numeric vector or a column name in `newdata`; by default the fit's own offset column is read from `newdata` (see `predict.cpb()`).
`se.fit`, `interval`, `level`	Standard errors (`se.fit = TRUE`) and a `level` confidence interval (`interval = "confidence"`) of the predicted quantity at the rows of `newdata`; see `predict.cpb()`.
`...`	Unused.

Value

A numeric vector. With `interval = "confidence"`, a matrix with columns `fit`, `lwr` and `upr`; with `se.fit = TRUE`, a list of `fit` (the vector or the matrix), `se.fit` and `method` (each row's interval source). Without `se.fit`, the source is the attribute `method`.

`predict.hurdle_gec` *Predict from a hurdle-GEC fit*

Description

Predict from a hurdle-GEC fit

Usage

```
## S3 method for class 'hurdle_gec'
predict(
  object,
  newdata = NULL,
  type = c("response", "participation", "intensity"),
  offset = NULL,
  se.fit = FALSE,
  interval = c("none", "confidence"),
  level = 0.95,
  ...
)
```

Arguments

`object`	A `"hurdle_gec"` object.
`newdata`	Optional covariate profiles (intensity and participation). A `part_fe` unit the fit has not seen, or every row when that column is absent, enters at the reference unit, and a unit at a limit at that limit (see `part_fe` in `hurdle_gec()`).
`type`	`"response"` (marginal E(Y)), `"participation"` (P(Y>0)), or `"intensity"` (E(Y | Y>0)).
`offset`	Optional offset (log scale) of the intensity for the `newdata` rows; see `predict.cpb()`.
`se.fit, interval, level`	Standard errors (`se.fit = TRUE`) and a `level` confidence interval (`interval = "confidence"`) of the predicted quantity at the rows of `newdata`; see `predict.cpb()`.
`...`	Unused.

Value

A numeric vector. With `interval = "confidence"`, a matrix with columns `fit`, `lwr` and `upr`; with `se.fit = TRUE`, a list of `fit` (the vector or the matrix), `se.fit` and `method` (each row's interval source). Without `se.fit`, the source is the attribute `method`.

predict.zi_count *Predict from a zero-inflated Poisson/NB fit*

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,
  se.fit = FALSE,
  interval = c("none", "confidence"),
  level = 0.95,
  ...
)
```

Arguments

object	A "zi_count" object.
newdata	Optional data frame of covariate profiles. A fixed-effects unit an equation has not seen, or every row when its column is absent, enters that equation at the reference unit. A zero_fe unit with no positive count gets the structural-zero probability one and one whose every count is positive zero, and an fe unit with no positive count a count mean of zero (see compare_models(), Degrees of freedom).
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) of the count component for the newdata rows; by default the fit's own offset column is read from newdata (see predict.cpb()).
se.fit, interval, level	Standard errors (se.fit = TRUE) and a level confidence interval (interval = "confidence") of the predicted quantity at the rows of newdata; see predict.cpb().
...	Unused.

Value

A numeric vector. With interval = "confidence", a matrix with columns fit, lwr and upr; with se.fit = TRUE, a list of fit (the vector or the matrix), se.fit and method (each row's interval source). Without se.fit, the source is the attribute method.

`predict.zi_cpb` *Predict from a zi_cpb fit*

Description

Predict from a zi_cpb fit

Usage

```
## S3 method for class 'zi_cpb'
predict(
  object,
  newdata = NULL,
  type = c("response", "zero", "intensity", "rate"),
  offset = NULL,
  se.fit = FALSE,
  interval = c("none", "confidence"),
  level = 0.95,
  ...
)
```

Arguments

`object`	A `"zi_cpb"` object.
`newdata`	Data frame of covariate profiles. A `zero_fe` unit the fit has not seen, or every row when that column is absent, enters at the zero equation's reference unit, and a unit at a limit at that limit (see `zero_fe` in `zi_cpb()`); for the count equation's `fe`, see `offset`.
`type`	`"response"` for the marginal expected count E(Y), `"zero"` for the structural-zero probability, `"intensity"` for the mean of the CPB component (on the floor support, its rate lambda), or `"rate"` for the component's rate lambda.
`offset`	Optional offset (log scale) of the count component for the `newdata` rows; see `predict.cpb()`. With `fe`, a row whose `fe` column names a unit of the fit enters at that unit's count effect, a row of an unseen unit (or every row, when `newdata` has no `fe` column) at the reference unit, the mean effect of the units the data identify. A unit left out for having no positive count (see `zi_cpb()`, `fe`) predicts a mean of zero, and a structural-zero probability of `NA` where its own `zero_fe` dummy would be needed.
`se.fit, interval, level`	Standard errors (`se.fit = TRUE`) and a `level` confidence interval (`interval = "confidence"`) of the predicted quantity at the rows of `newdata`; see `predict.cpb()`.
`...`	Unused.

Value

A numeric vector. With `interval = "confidence"`, a matrix with columns `fit`, `lwr` and `upr`; with `se.fit = TRUE`, a list of `fit` (the vector or the matrix), `se.fit` and `method` (each row's interval source). Without `se.fit`, the source is the attribute `method`.

`predict.zi_gec` *Predict from a zi_gec fit*

Description

Predict from a zi_gec fit

Usage

```
## S3 method for class 'zi_gec'
predict(
  object,
  newdata = NULL,
  type = c("response", "zero", "intensity"),
  offset = NULL,
  se.fit = FALSE,
  interval = c("none", "confidence"),
  level = 0.95,
  ...
)
```

Arguments

`object`	A `"zi_gec"` object.
`newdata`	Optional covariate profiles. A `zero_fe` unit the fit has not seen, or every row when that column is absent, enters at the zero equation's reference unit; a unit with no positive count gets the structural-zero probability one, and one whose every count is positive zero (see `compare_models()`, Degrees of freedom).
`type`	`"response"` (marginal E(Y)), `"zero"` (structural-zero probability), or `"intensity"` (the count mean E of the GEC component).
`offset`	Optional offset (log scale) of the count component for the `newdata` rows; see `predict.cpb()`.
`se.fit, interval, level`	Standard errors (`se.fit = TRUE`) and a `level` confidence interval (`interval = "confidence"`) of the predicted quantity at the rows of `newdata`; see `predict.cpb()`.
`...`	Unused.

Value

A numeric vector. With `interval = "confidence"`, a matrix with columns `fit`, `lwr` and `upr`; with `se.fit = TRUE`, a list of `fit` (the vector or the matrix), `se.fit` and `method` (each row's interval source). Without `se.fit`, the source is the attribute `method`.

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, support = c("ceiling", "floor"))
```

Arguments

n	Number of draws (recycled against lambda).
lambda	Rate parameter; a scalar or a length-n vector (see dcpb()).
alpha	Shape/dispersion parameter in (0, 1).
truncated	If TRUE, draw from the zero-truncated CPB.
support	The CPB's support, "ceiling" (King's CPB, the default) or "floor"; see cpb().

Details

King's CPB has the support `0, ..., ceiling(lambda / (1 - alpha))`, which contains 1 for every positive rate. The floor support, `0, ..., floor(lambda / (1 - alpha))`, is {0} for a rate whose ceiling is below 1, 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))
```

residuals.count_reg *Residuals of a fitted model*

Description

Response residuals (y minus the fitted mean) for every model class, and for the single-equation classes Pearson residuals, the response residual divided by the standard deviation of the fitted distribution (its exact variance; the conditional one of a zero-truncated fit).

Usage

```
## S3 method for class 'count_reg'
residuals(object, type = c("response", "pearson"), ...)

## S3 method for class 'gec'
residuals(object, type = c("response", "pearson"), ...)

## S3 method for class 'cpb_fe'
residuals(object, type = c("response", "pearson"), ...)
```

Arguments

object	A fitted model from this package.
type	"response" (default) or, for cpb, cpb_fe, gec, gec_fe and count_reg, "pearson".
...	Unused.

Value

A numeric vector, in the rows of the data.

rhurdle_cpb *Simulate from the hurdle continuous parameter binomial*

Description

Simulate from the hurdle continuous parameter binomial

Usage

```
rhurdle_cpb(n, lambda, alpha, p, support = c("ceiling", "floor"))
```

Arguments

n	Number of units.
lambda	Intensity rate for participants (the CPB's lambda); scalar or length-n.
alpha	CPB shape parameter in (0, 1).
p	Participation probability; scalar or length-n.
support	The CPB's support, "ceiling" (King's CPB, the default) or "floor"; see cpb().

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 *Hanging rootogram for a fitted count model*

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 score() takes).
kmax	Highest count to display; defaults to the maximum observed count.
main, xlab, ylab	Plot labels.
...	Passed to graphics::plot().

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, support = c("ceiling", "floor"))
```

Arguments

`n`	Number of units.
`lambda`	Intensity rate (the CPB's `lambda`); scalar or length-`n`.
`alpha`	CPB shape parameter in (0, 1).
`pi`	Structural-zero probability; scalar or length-`n`.
`support`	The CPB's support, `"ceiling"` (King's CPB, the default) or `"floor"`; see `cpb()`.

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, offset = 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. A fixed-effects unit the fit has not seen, or a level of any factor its training rows lack (a held-out block's own units or years), enters at the reference unit: the effect, or the dummies' average, of the units the data identify, the reference that first_difference() uses (on the floor support, the participation and zero equations of hurdle_cpb() and zi_cpb() enter such a unit at the mean of all their unit dummies). A unit the hurdles or mixtures hold at a limit is scored at that limit: a positive count of a unit with no positive count in the fitted data, and a hurdle's zero of a unit whose every count was positive, are impossible (see compare_models(), Degrees of freedom). A fit made with an offset reads the rows' offset from its own offset column in newdata, or from offset.
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.
offset	Optional offset (log scale) for the rows of newdata: a column name of newdata or a numeric vector with one value per row. A fit whose offset was given as a vector needs it here; one given as a column name reads that column of newdata by default.

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 -logLik/n and

does not penalize model complexity, so models of different size are better compared on the held-out or cross-validated score.

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 *Simulate responses from a fitted underdisp model*

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 `stats::simulate()`: 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 *Summarize a CPB fit*

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 fitted model (broom method)*

Description

One row per coefficient with its estimate, standard error, z statistic and p-value, for every model class of the package. The two-part classes (hurdle_*, zi_*) have 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. A term of a zero-inflation or participation equation that the data do not bound (separation; see zi_cpb()) has no standard error.

Usage

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

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

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

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

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

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

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

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

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

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

Arguments

x	A fitted model from this package.
conf.int	If TRUE, add conf.low/conf.high, the intervals of confint() (NA where the fit carries no inference).
conf.level	Confidence level for the interval.
...	Unused.

Details

The scale column names what each coefficient multiplies: "log mean" for the mean-parameterized count families ("log untruncated mean" for a zero-truncated fit or a hurdle's intensity), the binary equation's link for a participation or zero-inflation row, and otherwise the family's own parameter ("log rate" for the CPB and GEC, "log lambda" for the rate-parameterized COM-Poisson, "log centrality" for the EWP, and so on; see irr.count). Coefficients on different scales are not comparable across the columns of a table; ame() puts every model's effect on the mean.

Value

A data frame with one row per coefficient.

---

ud_screen *Screen a count outcome for underdispersion*

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 (the units at risk of the event, in a hurdle's reading those that participate), 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 it fits a generalized-Poisson soft-tail comparator. (The CPB's bound is tested on held-out counts with ceiling_exceedance(); in sample every feasible fit respects it.)

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,
  support = c("ceiling", "floor")
)
```

Arguments

formula	A model formula.
data	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% quantile of the statistic). The calibrated rule is 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. The bootstrap restores the random-number state, so every other result is the same with or without it.

`support` The CPB's support, `"ceiling"` (King's CPB, the default) or `"floor"`; see `cpb()`.

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 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: From restrictive assumptions to a generalized estimator. *AJPS* 33(3):762-784.

Bagozzi, B. E. (2026). Revisiting Underdispersion in Political Science: Diagnostics and a Typology for Count Data. 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", "hyperpois", "ewp3", "ewp2"),
  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 ~ 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. In either equation the dummy of a unit with no positive count is not counted in the degrees of freedom (see `compare_models()`, Degrees of freedom). In new data (with one column each) such a unit gets the structural-zero probability one through `zero_fe` and a count mean of zero through `fe`, and a unit whose every count is positive the structural-zero probability zero.
`link`	Link for the inflation probability: `"logit"` (default), `"probit"`, or `"cloglog"`. Positive inflation coefficients raise the probability of a *structural zero*.
`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"`; `$fitted.values` is the marginal mean `(1 - pi) E(Y | count component)` that `fitted()` returns, and `$mu` the count component's natural parameter.

Separation

The fit checks its zero-inflation equation for separation as zi_cpb() does (see its Separation section), under its own link. It warns when the data put no useful bound on a covariate coefficient, and stores the check in the component separation. Such a coefficient's standard error is not meaningful either.

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 pi 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. The mixture likelihood's optima differ with the structural share of zeros, so the search starts from points that span it: separate fits (a zero-truncated CPB on the positives and a logit of the zero indicator), the zero-inflated Poisson's coefficients, and a low-inflation start (a constant structural-zero probability of 0.02), followed by a BFGS pass; it is cross-checked against an expectation-maximization climb of the same observed-data likelihood, keeping the better optimum. The separation check (Separation) restarts the final polish from any better point it finds. 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 = NULL,
```

```
  maxit = 200,
  tol = 1e-06,
  offset = NULL,
  weights = NULL,
  cores = 1L,
  support = c("ceiling", "floor")
)
```

Arguments

formula	Intensity (count) model formula (y ~ x).
data	A data frame.
zero	Optional one-sided formula (~ 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 to the positive counts, and on King's support the fit is then polished on the observed-data likelihood with every unit's effect free, so that it is a joint maximum of the mixture likelihood (on the floor support the fit keeps the plug-in effects, which are not a joint maximum: a unit whose positive counts are all 1 takes an arbitrary point of the first stage's flat set, so the inflation equation and the log-likelihood can depend on the order of the rows); under method = "em" a classification (hard-assignment) step handles the structural zeros before the same polish. A unit with no positive count is left out of the estimation, as fixest leaves such units out of a Poisson fixed-effects fit: at the maximum its count effect is minus infinity, so its zeros are certain whatever its structural-zero probability, and its rows add nothing to the likelihood or the degrees of freedom. Its rows stay in the fit at that limit (the effect at the cpb_fe() sentinel, 30 log units below the unit's largest linear predictor), with fitted means of zero, and the fit records the units in zero_units and their number in n_zero_units. Their structural-zero probability is NA where the zero equation would give them their own dummy (zero_fe), which their zeros cannot identify. On the floor support such units stay in the estimation with a filled-in count effect (the smallest estimated effect minus 2 under "ml", the mean effect under "em").
zero_fe	Optional column name for fixed effects in the zero-inflation equation, entered as factor dummies (opt-in). On King's support the dummy of a unit with no positive count is not counted in the degrees of freedom (see compare_models(), Degrees of freedom), and in new data such a unit's rows get the structural-zero probability one and those of a unit whose every count is positive (zero_fe_one_units) zero; a unit the fit has not seen, or every row when newdata lacks the column, enters at the zero equation's reference unit.
method	Estimation method: "ml" (default) is direct joint maximum likelihood from the starts described above; "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	Guard on the evaluated support of the count component; see cpb(). The default NULL sets max(500, 10 * max(y)) (500 on the floor support). It is kept with the fit for simulate().
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; with fe, an offset is refused with an error. 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().
support	The CPB's support, "ceiling" (King's CPB, the default) or "floor"; 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". Its component separation holds the separation check (see Separation).

Separation

A zero contributes log(pi + (1 - pi) f0) to the likelihood, which rises with pi, and a positive count contributes log(1 - pi) + log f(y), which falls with it. So a coefficient of the zero-inflation equation can grow without bound, and the likelihood with it, under three conditions. The zeros it pushes toward pi = 1 can go there freely, and so can the positives it pushes toward pi = 0. The count component's probability of a zero, f0, covers the zeros it pushes toward pi = 0. Unlike separation in a logistic regression, this needs no covariate that splits the zeros from the positives. So a hurdle_cpb() whose participation logit uses the same covariates can have a finite maximum while the mixture has none. The optimizer then stops somewhere along the divergent path. The zero equation's coefficients there are not estimates, and the log-likelihood approximates a supremum, so likelihood comparisons with the fit (likelihood ratio, AIC, Vuong) lack their usual reference distributions.

After the fit, the profile likelihood of each covariate coefficient in the zero equation is evaluated at the estimate. It is evaluated again at 10 and at 100 standard deviations of its covariate further from zero. Each time, the other zero-equation coefficients are re-maximized with the count component held at its estimate. The fit warns when a coefficient's profile 100 standard deviations out is less than qchisq(0.95, 1) / 2 below its value at the estimate. That coefficient's 95% profile interval then reaches past an effect of 100 on the log-odds per standard deviation, and the data put no useful

bound on it. This covers a coefficient with no finite estimate and one whose finite maximum lies on a near-flat ridge, which a fit cannot tell apart. Holding the count component fixed can only lower the profile, so the full model's interval is at least as wide. Every profile can also reach the no-inflation boundary, where each structural-zero probability is zero and the zero equation drops out. A coefficient whose profile at the estimate lies within `qchisq(0.95, 1) / 2` of that limit is unbounded because the data show little inflation, not by separation, so it is not flagged, and the fit is reported as near the boundary instead (`zi_test()` tests it). The intercept and the `zero_fe` dummies are not checked. Their limits are the no-inflation boundary (`zi_test()`) and the boundaries of single units (a unit with no positive count, whose structural-zero probability goes to one, as in a fixed-effects logit; with `fe`, such a unit is left out of the estimation, see `fe`). The component `separation` of the fit holds:

- `diverging`: the terms the data do not bound;
- `profile`: the profile steps;
- `short_of_max`: how far the best point the check found lies above the fit, with the count component unchanged. That point can come from the re-maximization at the estimate, when the fit is not a stationary point of the zero equation, or from a profile that rises above its value at the estimate, since the mixture likelihood is not concave and a profile can rise before it falls. A positive value means the optimizer stopped at least that far below the maximum;
- `best`: the zero-equation coefficients at that point (`NULL` when there is none);
- `restarts`, `restart_gain`: when the check finds a point at least 0.01 higher, the fit restarts its final polish there, with every parameter it estimates free, keeps the result only if it gains, and checks again, up to three times. These record the restarts kept and their total gain in log-likelihood. The other components describe the final fit;
- `no_inflation`, `inflation_gain`: whether the best point lies within `qchisq(0.95, 1) / 2` of the no-inflation limit (the count component held), and its distance from it;
- `n_extreme`: the number of observations whose structural-zero probability is numerically 0 or 1, outside units that lie wholly at one edge.

`zi_gec()` and `zi_count()` run the same check.

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_gec *Zero-inflated GEC (Katz-family) regression*

Description

A structural-zero mixture with a GEC count component whose dispersion `delta` is estimated freely. Estimated by direct maximum likelihood from several starts, as in `zi_cpb()`: the single-equation GEC with a logit of the zero indicator for the inflation, at four dispersions; the zero-inflated Poisson's coefficients; and a low-inflation start. The best is polished, which avoids the degenerate `pi -> 0` basin that coordinate-wise EM falls into.

Usage

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

Arguments

formula	Count formula, y ~ 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). The dummy of a unit with no positive count is not counted in the degrees of freedom (see compare_models(), Degrees of freedom); in new data such a unit gets the structural-zero probability one, and one whose every count is positive zero.
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 evaluated support of the count component; NULL (default) sets max(500, 10 * max(y)). See gec().
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".

Separation

The fit checks its zero-inflation equation for separation as `zi_cpb()` does (see its Separation section). It warns when the data put no useful bound on a covariate coefficient, and stores the check in the component `separation`.

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 $\frac{1}{2}\chi^2_0 + \frac{1}{2}\chi^2_1$ mixture (Self and Liang 1987), which halves the naive χ^2_1 p-value. The test uses an intercept-only structural-zero probability, so it is a clean single-parameter boundary test. Given two fitted models it refuses the cases where that reference distribution does not hold: a covariate-dependent structural-zero probability (several parameters on their boundary), a fixed-effects `zi_cpb()` (one count effect per unit, whose number grows with the panel, so the statistic has no fixed-dimension boundary reference), a separated zero equation, models fit to different data, a zero-truncated nest, and CPB models on different supports. It is not a test of a hurdle against a mixture, which are not nested. Compare those with `compare_models()`.

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`, `gec`, `zi_cpb`, or `zi_gec`).
`object2`	The second argument: the data frame when `object` is a formula (so `zi_test(y ~ x, mydata)` 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 `zi_cpb` (or `zi_gec`) with intercept-only inflation and the other its untruncated non-inflated nest (`cpb`, or `gec`) on the same data.
`data`	A data frame; an alternative to passing it as `object2`.
`...`	Passed to `cpb()` and `zi_cpb()` 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, 60

alpha_confint, 3
alpha_confint(), *7, 8*, *22*, *56*
ame, 4
ame(), *17*, *20*, *59*, *64*, *82*
augment.cpb, 6

calibrate_alpha, 7
calibrate_alpha(), *3*, *14*, *22*, *56*, *81*
ceiling_exceedance, 9
ceiling_exceedance(), *10*, *20*, *56*, *65*, *67*, *82*
compare_dispersion, 10
compare_dispersion(), *12*, *44*, *84*
compare_models, 11
compare_models(), *16*, *19, 20*, *30*, *50–52*, *54*, *71*, *73*, *78*, *85–88*, *91, 92*
compois-distribution, 13
confint(), *82*
confint.count_reg *(*confint.underdisp*)*, 14
confint.cpb, 14
confint.cpb(), *7*, *15*, *21, 22*
confint.cpb_fe *(*confint.underdisp*)*, 14
confint.gec *(*confint.underdisp*)*, 14
confint.hurdle_count *(*confint.underdisp*)*, 14
confint.hurdle_cpb *(*confint.underdisp*)*, 14
confint.hurdle_gec *(*confint.underdisp*)*, 14
confint.underdisp, 14
confint.zi_count *(*confint.underdisp*)*, 14
confint.zi_cpb *(*confint.underdisp*)*, 14
confint.zi_gec *(*confint.underdisp*)*, 14
count_reg, 16
count_reg(), *5*, *14*, *20*, *27*, *33–35*, *43*, *48*, *50, 51*, *56*, *60*, *85, 86*
cpb, 19
cpb(), *5*, *7*, *10, 11*, *14*, *16*, *19*, *24–28*, *41*, *43, 44*, *46*, *51*, *53, 54*, *60*, *74*, *76, 77*, *84*, *89*, *91, 92*
cpb-distribution, 23
cpb_fe, 24
cpb_fe(), *6*, *12*, *14*, *16*, *20, 21*, *45–47*, *52, 53*, *65*, *67*, *88*
cv_score, 28
cv_score(), *12*, *52*, *78, 79*

dcompois *(*compois-distribution*)*, 13
dcpb *(*cpb-distribution*)*, 23
dcpb(), *74*
ddoublepois *(*doublepois-distribution*)*, 33
dewp *(*ewp-distribution*)*, 34
dgammacount *(*gammacount-distribution*)*, 42
dgec *(*gec-distribution*)*, 44
dgec(), *44*
dgenpois *(*genpois-distribution*)*, 47
dhyperpois *(*hyperpois-distribution*)*, 55
dispersion_profile, 29
dispersion_profile(), *16*, *18, 19*, *33*
dispersion_test, 31
dispersion_test(), *8*, *16*, *18, 19*, *21, 22*, *30*, *44*, *84*
doublepois-distribution, *18*, 33

ewp-distribution, *18*, 34

first_difference, 36
first_difference(), *4–6*, *37*, *39*, *41*, *56, 57*, *59*, *64*, *78*
first_difference.count, *39*
first_difference.count *(*first_difference.gec*)*, 37
first_difference.count_reg *(*first_difference.gec*)*, 37

first_difference.cpb_fe
(first_difference.gec), 37
first_difference.gec, 37
first_difference.gec_fe
(first_difference.gec), 37
first_difference.hurdle_count
(first_difference.gec), 37
first_difference.hurdle_cpb, 39
first_difference.hurdle_gec
(first_difference.gec), 37
first_difference.zi_count
(first_difference.gec), 37
first_difference.zi_cpb
(first_difference.hurdle_cpb), 39
first_difference.zi_gec
(first_difference.gec), 37
fitted.cpb_fe (fitted.underdisp), 41
fitted.hurdle_count (fitted.underdisp), 41
fitted.hurdle_cpb (fitted.underdisp), 41
fitted.hurdle_gec (fitted.underdisp), 41
fitted.underdisp, 41
fitted.zi_count (fitted.underdisp), 41
fitted.zi_cpb (fitted.underdisp), 41
fitted.zi_gec (fitted.underdisp), 41

gammacount-distribution, *18*, 42
gec, 43
gec(), *14*, *16*, *19*, *21*, *44–47*, *54, 55*, *60*, *91, 92*
gec-distribution, 44
gec_fe, 45
gec_fe(), *12*, *14*, *24*
genpois-distribution, *17*, 47
glance.count_reg (glance.cpb), 49
glance.cpb, 49
glance.cpb_fe (glance.cpb), 49
glance.gec (glance.cpb), 49
glance.hurdle_count (glance.cpb), 49
glance.hurdle_cpb (glance.cpb), 49
glance.hurdle_gec (glance.cpb), 49
glance.zi_count (glance.cpb), 49
glance.zi_cpb (glance.cpb), 49
glance.zi_gec (glance.cpb), 49
graphics::barplot(), *62*
graphics::plot(), *76*

hurdle_count, 50
hurdle_count(), *19*, *68*, *87*
hurdle_cpb, 51
hurdle_cpb(), *7*, *10*, *50, 51*, *53–55*, *69*, *78*, *87*, *89*
hurdle_gec, 53
hurdle_gec(), *44*, *70*, *92*
hyperpois-distribution, *18*, 55

implied_ceiling, 56
implied_ceiling(), *7*, *9*, *22*
irr, 57
irr.count, *82*
irr.count (irr.count_reg), 58
irr.count_reg, 58
irr.cpb_fe (irr.count_reg), 58
irr.gec (irr.count_reg), 58
irr.gec_fe (irr.count_reg), 58
irr.hurdle_count (irr.count_reg), 58
irr.hurdle_gec (irr.count_reg), 58
irr.zi_count (irr.count_reg), 58
irr.zi_gec (irr.count_reg), 58

mundlak, 59

nobs(), *11*

pcompois (compois-distribution), 13
pcpb (cpb-distribution), 23
pdoublepois (doublepois-distribution), 33
peacekeeping, 60
pewp (ewp-distribution), 34
pgammacount (gammacount-distribution), 42
pgec (gec-distribution), 44
pgenpois (genpois-distribution), 47
phyperpois (hyperpois-distribution), 55
pit_hist, 61
pit_hist(), *20*
plot.dispersion_profile
(dispersion_profile), 29
predict.count_reg, 62
predict.cpb, 63
predict.cpb(), *6*, *9*, *22*, *63*, *65–73*
predict.cpb_fe, 65
predict.gec, 66
predict.gec_fe, 67
predict.hurdle_count, 68
predict.hurdle_cpb, 69
predict.hurdle_cpb(), *52*

predict.hurdle_gec, 70
predict.zi_count, 71
predict.zi_cpb, 72
predict.zi_cpb(), *71*
predict.zi_gec, 73

qcompois *(*compois-distribution*)*, 13
qcpb *(*cpb-distribution*)*, 23
qdoublepois *(*doublepois-distribution*)*, 33
qewp *(*ewp-distribution*)*, 34
qgammacount *(*gammacount-distribution*)*, 42
qgec *(*gec-distribution*)*, 44
qgenpois *(*genpois-distribution*)*, 47
qhyperpois *(*hyperpois-distribution*)*, 55

rcompois *(*compois-distribution*)*, 13
rcpb, 74
rcpb(), *23*, *24*
rdoublepois *(*doublepois-distribution*)*, 33
residuals.count_reg, 75
residuals.cpb_fe *(*residuals.count_reg*)*, 75
residuals.gec *(*residuals.count_reg*)*, 75
residuals.underdisp *(*residuals.count_reg*)*, 75
rewp *(*ewp-distribution*)*, 34
rgammacount *(*gammacount-distribution*)*, 42
rgec *(*gec-distribution*)*, 44
rgenpois *(*genpois-distribution*)*, 47
rhurdle_cpb, 75
rhyperpois *(*hyperpois-distribution*)*, 55
rootogram, 76
rootogram(), *20*, *30*
rzicpb, 77

score, 78
score(), *9*, *11*, *12*, *16*, *28*, *29*, *50*, *52*, *62*, *65*, *67*, *76*, *85*
simulate(), *89*
simulate.count_reg *(*simulate.underdisp*)*, 79
simulate.cpb *(*simulate.underdisp*)*, 79
simulate.cpb_fe *(*simulate.underdisp*)*, 79
simulate.gec *(*simulate.underdisp*)*, 79
simulate.hurdle_count *(*simulate.underdisp*)*, 79
simulate.hurdle_cpb *(*simulate.underdisp*)*, 79
simulate.hurdle_gec *(*simulate.underdisp*)*, 79
simulate.underdisp, 79
simulate.zi_count *(*simulate.underdisp*)*, 79
simulate.zi_cpb *(*simulate.underdisp*)*, 79
simulate.zi_gec *(*simulate.underdisp*)*, 79
stats::fitted(), *41*
stats::optim(), *21*
stats::simulate(), *7*, *79*, *80*
summary(), *17*
summary.cpb, 80

tidy.count_reg *(*tidy.cpb*)*, 81
tidy.cpb, 81
tidy.cpb(), *49*
tidy.cpb_fe *(*tidy.cpb*)*, 81
tidy.gec *(*tidy.cpb*)*, 81
tidy.hurdle_count *(*tidy.cpb*)*, 81
tidy.hurdle_cpb *(*tidy.cpb*)*, 81
tidy.hurdle_gec *(*tidy.cpb*)*, 81
tidy.zi_count *(*tidy.cpb*)*, 81
tidy.zi_cpb *(*tidy.cpb*)*, 81
tidy.zi_gec *(*tidy.cpb*)*, 81

ud_screen, 82
ud_screen(), *11*, *22*, *27*
update(), *6*

vcov.cpb_fe, 85

zi_count, 85
zi_count(), *12*, *14*, *19*, *51*, *89*, *90*
zi_cpb, 87
zi_cpb(), *10*, *12*, *14*, *52*, *72*, *78*, *81*, *85*, *87*, *90*, *92*
zi_gec, 90
zi_gec(), *12*, *14*, *55*, *90*
zi_test, 92
zi_test(), *11*, *12*, *33*, *90*