

Package: evinf (via r-universe)

September 24, 2026

Type Package

Title Inference with Extreme Value Inflated Count Data

Version 1.0.0

Description Allows users to model and draw inferences from extreme value inflated count data, and to evaluate these models and compare them to their non extreme-value inflated counterparts. The package is built to be compatible with standard model-presentation tools such as the 'broom', 'generics', and 'modelsummary' packages.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Imports dplyr, Rcpp, RcppArmadillo, future, furrr, parallelly, magrittr, tibble, tidyr, purrr, MASS, pscl, Rdpack, rlang, methods, stats, utils, generics, progressr

LinkingTo Rcpp, RcppArmadillo

Suggests testthat (>= 3.0.0), modelsummary, ggplot2, marginaleffects (>= 0.22.0), insight, broom, covr, withr, emmeans, estimability

RoxygenNote 7.3.3

RdMacros Rdpack

Depends R (>= 4.1.0)

URL <https://github.com/Doktorandahl/evinf>

BugReports <https://github.com/Doktorandahl/evinf/issues>

Config/testthat/edition 3

Config/pak/sysreqs libicu-dev

Repository <https://doktorandahl.r-universe.dev>

Date/Publication 2026-09-24 19:13:49 UTC

RemoteUrl <https://github.com/doktorandahl/evinf>

RemoteRef HEAD

RemoteSha cd16088beb74ca3b6d833abef6c3853a865311c3

Contents

add_bootstraps	3
augment.evinb	5
augment.evzinb	5
c_profile	6
check_evinf	7
classify_states	8
coef.evzinb	9
coefficient_extractor	9
compare_fit	11
compare_models	12
confint.evzinb	14
evinb	15
evinf-s3-accessors	19
evinf-s3-predict	20
evinf_bench_data	22
evinf_control	23
evinf_family	26
evzinb	27
evzinbcomp-tidiers	31
failed_bootstraps	32
genevzinb	32
genevzinb2	33
glance.evinb	33
glance.evzinb	34
glance.zinbboot	35
glance.zipboot	36
gm_evzinb	37
gof_map_evinf	38
hks	39
lr_test	40
marginal_effects	42
oob_evaluation	45
plot.evzinb	46
plot_c_profile	47
predict.evinb	48
predict.evzinb	51
predict.nbboot	55
predict.poissonboot	56
predict.zinbboot	57
predict.zipboot	58
predict_grid	59
print.evinb	60
print.evzinb	61
print.evzinbcomp	61
print.summary.evzinb	62
revinb_fit	63

revzinb_fit	63
state_table	64
summary.evinb	65
summary.evzinb	66
tidy.evinb	68
tidy.evzinb	69
tidy.nbboot	71
tidy.poissonboot	72
tidy.zinbboot	74
tidy.zipboot	75
update.evzinb	76
vcov.evzinb	77
Index	79

add_bootstraps	<i>Add more bootstrap replicates to a fitted model</i>
----------------	--

Description

Fits *n* additional bootstrap replicates with the same block structure and appends them, so you don't have to refit everything when the original number turns out to be too few.

Usage

```
add_bootstraps(  
  object,  
  n,  
  boot_seed = NULL,  
  multicore = NULL,  
  ncores = NULL,  
  verbose = FALSE  
)
```

Arguments

object	A fitted evzinb / evinb model (with bootstraps).
n	Number of additional bootstrap replicates.
boot_seed	RNG seed for this batch. Each batch needs its own seed: the bootstrap stream is fully determined by the seed, so reusing one that the model (or an earlier add_bootstraps() call) already used would silently duplicate those draws and is an error. When NULL a fresh seed is drawn and recorded in object\$boot_seeds.
multicore	Bootstrap parallelisation shortcut. The default (NULL) respects whatever future plan is currently set (see the Parallel processing section). TRUE sets a temporary multisession plan for the duration of the call; FALSE forces sequential execution. The previous plan is always restored on exit.

ncores	Number of workers when multicore = TRUE. Default (NULL) is one less than the number of available cores. Ignored when multicore is NULL or FALSE.
verbose	Show a progress bar.

Value

The model with n more replicates in `object$bootstraps` (names continue `bootstrap_<k>`).

Parallel processing

Bootstrap fits (and the per-bootstrap work in [add_bootstraps](#), [lr_test](#), [compare_models](#), [predict.evzinb](#) and [marginal_effects](#)) are dispatched with **furrr** on top of a **future** plan. Set the plan once for your session and leave multicore at its default:

```
future::plan(future::multisession, workers = 8)
progressr::handlers(global = TRUE) # opt in to progress bars
model <- evzinb(y ~ x1 + x2 + x3, data = genevzinb2, n_bootstraps = 1000)
```

Any future backend works (`multisession`, `cluster`, `callr`, a HPC batchtools plan, ...). As a convenience `multicore = TRUE` sets a temporary `multisession` plan for the single call and restores the previous plan on exit. Large models may need `options(future.globals.maxSize = <bytes>)` to raise the default limit on the data shipped to each worker.

Reproducibility

The bootstrap **resample indices** (`object$bootstraps[[i]]$boot_id`) are fully determined by `boot_seed` and are independent of the **future** backend, the number of workers, and chunking — a fixed seed always draws the same resamples.

The **fitted coefficients** are reproducible only to within floating-point noise. BLAS operations are not bit-reproducible across processes, so a sequential run and a `multisession` run of the same seed can differ in the last few digits; and because the EM log-likelihood can have close local optima, an occasional replicate converges to a different one under a different backend. For replication material, record the future plan and `utils::sessionInfo()` alongside `boot_seed`, and produce the canonical set of estimates under a single fixed plan.

Examples

```
data(genevzinb2)
model <- evzinb(y ~ x1 + x2 + x3, data = genevzinb2, n_bootstraps = 5)
model <- add_bootstraps(model, 5)
```

augment.evinb	<i>Augment data with fitted values, residuals and state classification</i>
---------------	--

Description

Augment data with fitted values, residuals and state classification

Usage

```
## S3 method for class 'evinb'
augment(x, data = NULL, newdata = NULL, seed = NULL, ...)
```

Arguments

x	A fitted evinb model.
data, newdata	Data to augment. newdata takes precedence over data when both are given (the broom convention); both default to NULL, in which case the model's own estimation data is used.
seed	Optional seed for the randomized quantile residual (.resid); see residuals(type = "quantile").
...	Unused.

Value

See [augment.evzinb](#); evinb has no zero state, so .prob_zero/.post_zero are not produced.

Examples

```
data(genevzinb2)
model <- evinb(y~x1+x2+x3, data=genevzinb2, n_bootstraps = 5)
augment(model)
```

augment.evzinb	<i>Augment data with fitted values, residuals and state classification</i>
----------------	--

Description

Augment data with fitted values, residuals and state classification

Usage

```
## S3 method for class 'evzinb'
augment(x, data = NULL, newdata = NULL, seed = NULL, ...)
```

Arguments

x	A fitted evzinb model.
data, newdata	Data to augment. newdata takes precedence over data when both are given (the broom convention); both default to NULL, in which case the model's own estimation data is used.
seed	Optional seed for the randomized quantile residual (.resid); see residuals(type = "quantile").
...	Unused.

Value

data/newdata (or the estimation data) with columns appended: .fitted (the predict(type = "harmonic") point prediction), .prob_zero/.prob_count/.prob_evi (prior state probabilities – augment.evinb() has no zero state, so .prob_zero is dropped there) and .state (the MAP prior state, from [classify_states\(\)](#)). When the response column is present, three more columns are added: .resid (a seeded randomized quantile residual, residuals(type = "quantile")), .post_zero/.post_count/.post_evi (posterior state probabilities) and .post_state (the MAP posterior state).

Examples

```
data(genevzinb2)
model <- evzinb(y~x1+x2+x3, data=genevzinb2, n_bootstraps = 5)
augment(model)
```

c_profile

Log-likelihood profile over the candidate values of C_{EV} **Description**

Returns the log-likelihood evaluated at each candidate value of C_{EV} from the final EM iteration. Use it to judge whether C_{EV} is well identified and whether the candidate range (c.lim) is wide enough.

Usage

```
c_profile(object)
```

Arguments

object	A fitted evzinb / evinb object.
--------	---------------------------------

Value

A tibble with columns c and loglik.

Examples

```
data(genevzinb2)
model <- evzinb(y ~ x1 + x2 + x3, data = genevzinb2, n_bootstraps = 5)
c_profile(model)
```

check_evinf

Diagnostic report for a fitted evzinb / evinb model

Description

A single report gathering the convergence and health checks otherwise spread across `print()`, `glance()` and `failed_bootstraps()`: EM and C_{EV} -profile convergence, whether C_{EV} landed on a candidate-grid boundary (full sample and across bootstrap replicates), the smallest fitted Pareto shape against `alpha_pl_floor`, failed and degenerate bootstrap replicates, start agreement (see `n_starts` in [evinf_control](#)), and, for the block bootstrap schemes, the out-of-bag row fraction.

Usage

```
check_evinf(object)
```

Arguments

`object` A fitted evzinb / evinb model.

Value

A classed tibble ("evinf_check") with columns `check`, `status` ("ok", "note" or "warning") and `detail` (a one-line description, with a suggestion for anything not "ok").

Examples

```
data(genevzinb2)
model <- evzinb(y ~ x1 + x2 + x3, data = genevzinb2, n_bootstraps = 5)
check_evinf(model)
```

classify_states	<i>Prior and posterior state classification</i>
-----------------	---

Description

For each observation, returns the prior state probabilities (from the inflation components) and, for the estimation data, the posterior state probabilities (responsibilities), together with the maximum-a-posteriori state under each.

Usage

```
classify_states(
  object,
  rule = c("map", "threshold"),
  threshold = 0.5,
  newdata = NULL
)
```

Arguments

object	A fitted evzinb / evinb model.
rule	"map" (assign to the most probable state) or "threshold" (assign only when the top probability exceeds threshold, otherwise NA).
threshold	Threshold for rule = "threshold".
newdata	Optional data. Prior probabilities are always available; posterior probabilities need the response column and are NA otherwise.

Value

A tibble with prior_zero/prior_count/prior_evi, posterior_*, map_prior, map_posterior (factors with levels zero, count, evi) and y.

Examples

```
data(genevzinb2)
model <- evzinb(y ~ x1 + x2 + x3, data = genevzinb2, n_bootstraps = 5)
classify_states(model)
```

coef.evzinb	<i>Coefficients of an evzinb / evinb model</i>
-------------	--

Description

Coefficients of an evzinb / evinb model

Usage

```
## S3 method for class 'evzinb'
coef(object, component = "all", ...)

## S3 method for class 'evinb'
coef(object, component = "all", ...)
```

Arguments

object	A fitted evzinb / evinb model.
component	One of "all" (the default), "count", "zero", "evi", "pareto". Deprecated aliases "nb", "zi", "evinf" are accepted, with a warning; they will be removed in a future release.
...	Unused.

Value

For "all", a named numeric vector <component>_<term>, alpha_nb, c_ev; for a single component the plain named vector.

coefficient_extractor	<i>Bootstrap coefficient extractor</i>
-----------------------	--

Description

Bootstrap coefficient extractor

Usage

```
coefficient_extractor(object, ...)

## S3 method for class 'evzinb'
coefficient_extractor(
  object,
  component = c("all", "count", "zero", "evi", "pareto"),
  exclude_degenerate = TRUE,
  ...)
```

```

)

## S3 method for class 'evinb'
coefficient_extractor(
  object,
  component = c("all", "count", "evi", "pareto"),
  exclude_degenerate = TRUE,
  ...
)

## S3 method for class 'zinbboot'
coefficient_extractor(object, component = c("all", "count", "zero"), ...)

## S3 method for class 'nbboot'
coefficient_extractor(object, ...)

```

Arguments

<code>object</code>	a fitted model with bootstraps of class <code>evzinb</code> , <code>evinb</code> , <code>nbboot</code> , or <code>zinbboot</code>
<code>...</code>	Arguments passed to methods, in particular <code>component</code> (not for <code>nbboot</code>): one of "count", "zero", "evi", "pareto" or "all". The pre-0.9.4 names "nb", "zi" and "evinf" are still accepted with a deprecation warning.
<code>component</code>	Which component should be extracted
<code>exclude_degenerate</code>	Drop bootstrap replicates flagged degenerate (default TRUE); see evinf_control .

Value

A tibble with coefficient values, one row per bootstrap and component

Examples

```

data(genevzinb2)
model <- evzinb(y~x1+x2+x3,data=genevzinb2, n_bootstraps = 5)
coefficient_extractor(model, component = 'all')

data(genevzinb2)
model <- evzinb(y~x1+x2+x3,data=genevzinb2, n_bootstraps = 5)
zinb_comp <- compare_models(model)
coefficient_extractor(zinb_comp$zinb)

data(genevzinb2)
model <- evzinb(y~x1+x2+x3,data=genevzinb2, n_bootstraps = 5)
zinb_comp <- compare_models(model)
coefficient_extractor(zinb_comp$nb)

```

compare_fit

*Compare the extreme-value model against its plainer counterparts***Description**

For each compared model (nb, zinb, and any winsorized / razorized variants) and each metric, computes the paired bootstrap difference `evinf - compared` (so a negative median favours the extreme-value model, matching Table B3 of the ISQ appendix), the proportion of bootstraps in which the `evinf` model is better, and the number of bootstrap pairs where both fits succeeded.

Usage

```
compare_fit(
  comp,
  metrics = c("aic", "bic", "rmse", "rmsle"),
  exclude_degenerate = TRUE,
  ...
)

## S3 method for class 'evzinbcomp'
plot(
  x,
  metrics = c("aic", "bic", "rmse", "rmsle"),
  exclude_degenerate = TRUE,
  ...
)
```

Arguments

<code>comp</code>	An <code>evzinbcomp</code> object from <code>compare_models()</code> .
<code>metrics</code>	Metrics to show.
<code>exclude_degenerate</code>	Drop bootstrap replicates of the <code>evinf</code> model flagged degenerate (default TRUE); see the <code>alpha_floor</code> argument of <code>evinf_control</code> . Excluded replicates are treated as missing rather than dropped, so the pairing with the compared models' replicates is preserved.
<code>...</code>	Unused.
<code>x</code>	An <code>evzinbcomp</code> object.

Details

The `aic` / `bic` rows are NA for a `*_razor` or `*_winsor` slot (round8 0.5): a razorised fit is estimated on fewer observations than the `evinf` model, and a winsorised fit is estimated on a different outcome, so their AIC/BIC are not on the same scale as the `evinf` model's and a difference between them is not meaningful. The `RMSE` / `RMSLE` rows for those slots remain comparable, because out-of-bag error is always computed against the raw (un-winsorised, un-razorised) outcome.

Value

A tibble of class `evinf_compare_fit` with columns `model`, `metric`, `median_difference`, `prop_evinf_better`, `n_pairs`. `median_difference / prop_evinf_better` are NA for the `aic / bic` rows of a `*_razor` or `*_winsor` slot (see Details); `n_pairs` is left as computed on the returned object (`print()` blanks it for those rows instead, since a pair count next to an NA metric reads like a bug).

Examples

```
data(genevzinb2)
model <- evzinb(y ~ x1 + x2 + x3, data = genevzinb2, n_bootstraps = 10)
compare_fit(compare_models(model))
```

`compare_models`

Function to compare evzinb or evinb models with zinb and nb models

Description

Function to compare evzinb or evinb models with zinb and nb models

Usage

```
compare_models(
  object,
  nb_comparison = TRUE,
  zinb_comparison = TRUE,
  poisson_comparison = identical((object$family %||% evinf_family())$count, "poisson"),
  zip_comparison = identical((object$family %||% evinf_family())$count, "poisson"),
  winsorize = FALSE,
  razorize = FALSE,
  cutoff_value = 10,
  init_theta = NULL,
  multicore = NULL,
  ncores = NULL
)
```

Arguments

<code>object</code>	A fitted evzinb or evinb model object
<code>nb_comparison</code>	Should comparison be made with a negative binomial model?
<code>zinb_comparison</code>	Should comparisons be made with the zinb model? Not available for evinb objects (there is no zero-inflation component); it defaults to FALSE for those, with a message, and errors if set to TRUE explicitly.

poisson_comparison	Should comparison be made with a Poisson model? Defaults to TRUE when object was itself fitted with family = "poisson", FALSE otherwise (round9 E.3); user-overridable either way.
zip_comparison	Should comparisons be made with a zero-inflated Poisson (ZIP) model? Same default as poisson_comparison. Not available for evinb objects, with the same defaults-to-FALSE-with-a-message / errors-if-TRUE behaviour as zinb_comparison.
winsorize	Should winsorizing be done in the comparisons?
razorize	Should razorizing (trimming) be done in the comparisons?
cutoff_value	Integer: Which observation should be used as a basis for winsorizing/razorizing. E.g. 10 means that everything larger than the 10th observation will be winsorized/razorised
init_theta	Optional initial value for theta in the NB specification
multicore	Bootstrap parallelisation shortcut. The default (NULL) respects whatever future plan is currently set (see the Parallel processing section). TRUE sets a temporary multisession plan for the duration of the call; FALSE forces sequential execution. The previous plan is always restored on exit.
ncores	Number of workers when multicore = TRUE. Default (NULL) is one less than the number of available cores. Ignored when multicore is NULL or FALSE.

Value

An object of class `evzinbcomp`: a list whose first element `model` is the original `evzinb`/`evinb` model (also available as `evzinb` for backwards compatibility), followed by the compared `nb` / `zinb` models (and their winsorized/razorized variants when requested).

Parallel processing

Bootstrap fits (and the per-bootstrap work in [add_bootstraps](#), [lr_test](#), [compare_models](#), [predict.evzinb](#) and [marginal_effects](#)) are dispatched with **furrr** on top of a **future** plan. Set the plan once for your session and leave `multicore` at its default:

```
future::plan(future::multisession, workers = 8)
progressr::handlers(global = TRUE) # opt in to progress bars
model <- evzinb(y ~ x1 + x2 + x3, data = genevzinb2, n_bootstraps = 1000)
```

Any future backend works (`multisession`, `cluster`, `callr`, a HPC batchtools plan, ...). As a convenience `multicore = TRUE` sets a temporary `multisession` plan for the single call and restores the previous plan on exit. Large models may need `options(future.globals.maxSize = <bytes>)` to raise the default limit on the data shipped to each worker.

Reproducibility

The bootstrap **resample indices** (`object$bootstraps[[i]]$boot_id`) are fully determined by `boot_seed` and are independent of the **future** backend, the number of workers, and chunking — a fixed seed always draws the same resamples.

The **fitted coefficients** are reproducible only to within floating-point noise. BLAS operations are not bit-reproducible across processes, so a sequential run and a multisession run of the same seed can differ in the last few digits; and because the EM log-likelihood can have close local optima, an occasional replicate converges to a different one under a different backend. For replication material, record the future plan and `utils::sessionInfo()` alongside `boot_seed`, and produce the canonical set of estimates under a single fixed plan.

Examples

```
data(genevzinb2)
model <- evzinb(y~x1+x2+x3,data=genevzinb2, n_bootstraps = 5)
compare_models(model)

data(hks)
hks_mod <- evzinb(osvAll ~ troopLag + lnpop + brv_AllLag_log + osvAllLagDum,
                  formula_pareto = ~ loglp(troopLag),
                  data = hks, n_bootstraps = 2, multicore = FALSE)
cmp <- compare_models(hks_mod)
compare_fit(cmp)
```

confint.evzinb

Confidence intervals for an evzinb / evinb model

Description

Confidence intervals for an evzinb / evinb model

Usage

```
## S3 method for class 'evzinb'
confint(
  object,
  parm,
  level = 0.95,
  type = c("percentile", "approx"),
  exclude_degenerate = TRUE,
  ...
)

## S3 method for class 'evinb'
confint(
  object,
  parm,
  level = 0.95,
  type = c("percentile", "approx"),
```

```

    exclude_degenerate = TRUE,
    ...
  )

```

Arguments

<code>object</code>	A fitted model with bootstraps.
<code>parm</code>	Which parameters (names as in <code>coef(object)</code>); default all.
<code>level</code>	Confidence level.
<code>type</code>	"percentile" (bootstrap percentile intervals, the default) or "approx" (estimate $\pm qnorm() * \sqrt{\text{diag(vcov)}}$).
<code>exclude_degenerate</code>	Drop bootstrap replicates flagged degenerate (default TRUE); see evinf_control .
<code>...</code>	Unused.

Value

A two-column matrix.

Examples

```

data(genevzinb2)
m <- evzinb(y ~ x1 + x2 + x3, data = genevzinb2, n_bootstraps = 25)
# the Pareto shape (alpha_nb) and the threshold (c_ev) are in coef()/confint()
confint(m, parm = c("count_x1", "alpha_nb", "c_ev"))

```

evinb	<i>Running an extreme value inflated negative binomial model with bootstrapping</i>
-------	---

Description

Running an extreme value inflated negative binomial model with bootstrapping

Usage

```

evinb(
  formula_nb,
  formula_evi = NULL,
  formula_pareto = NULL,
  data,
  bootstrap = TRUE,
  n_bootstraps = 100,
  multicore = NULL,
  ncores = NULL,
  block = NULL,

```

```

weights = NULL,
time = NULL,
bootstrap_scheme = NULL,
block_length = NULL,
family = evinf_family(),
boot_seed = NULL,
start_seed = NULL,
control = evinf_control(),
max.diff.par,
max.no.em.steps,
max.no.em.steps.warmup,
c.lim,
prune.c.range,
max.upd.par.pl.multinomial,
max.upd.par.nb,
max.upd.par.pl,
no.m.bfgs.steps.multinomial,
no.m.bfgs.steps.nb,
no.m.bfgs.steps.pl,
pdf.pl.type,
eta.int,
init.Beta.multinom.PL,
init.Beta.NB,
init.Beta.PL,
init.Alpha.NB,
init.C,
verbose = FALSE
)

```

Arguments

formula_nb	Formula for the negative binomial (count) component of the model. May include an <code>offset()</code> term ($\mu_{NB} = \exp(x'\beta + offset)$).
formula_evi	Formula for the extreme-value inflation component of the model. If <code>NULL</code> taken as the same formula as <code>nb</code> , with any <code>offset()</code> term stripped (an offset applies only where it is written explicitly, never by inheritance). May include its own <code>offset()</code> term: the EVI logit is the log-odds of the extreme-value state against the count state, so an offset there shifts that log-odds, e.g. <code>offset(log(population))</code> for a probability of an extreme event that scales with exposure.
formula_pareto	Formula for the pareto (extreme value) component of the model. If <code>NULL</code> taken as the same formula as <code>nb</code> , offset stripped as above. <code>offset()</code> is not supported here (errors if present): an offset on a shape parameter has no clear reading.
data	Data to run the model on
bootstrap	Should bootstrapping be performed. Needed to obtain standard errors and p-values
n_bootstraps	Number of bootstraps to run. For use of bootstrapped p-values, at least 1,000

	bootstraps are recommended. For approximate p-values, a lower number can be sufficient
multicore	Bootstrap parallelisation shortcut. The default (NULL) respects whatever future plan is currently set (see the Parallel processing section). TRUE sets a temporary multisession plan for the duration of the call; FALSE forces sequential execution. The previous plan is always restored on exit.
ncores	Number of workers when multicore = TRUE. Default (NULL) is one less than the number of available cores. Ignored when multicore is NULL or FALSE.
block	Optional case-identifier column for block bootstrapping, given either as a bare column name (block = id) or a string (block = "id"). Note that the bundled hks data contain no conflict identifier, so the conflict-level cluster bootstrap in Randahl and Vegelius (2024) cannot be reproduced from them directly (see ?hks).
weights	Optional observation weights (round9 D.2), given as a bare column name (weights = wt), a string naming a column (weights = "wt"), or a numeric vector. Must be positive and finite; need not be integers (analytic weights are allowed, not just frequency counts). Frequency-weight semantics: every observation's contribution to the log-likelihood and to the EM/M-step accumulations is multiplied by its weight, and nobs() – and therefore AIC, BIC and the approximate t-based p-values – use sum(weights), not the row count (see sum_weights in glance.evinb). This interpretation of AIC/BIC assumes the weights really are frequency weights (repeat-count equivalents); for analytic weights the information-criterion values are still computed this way but their usual interpretation is weaker. weights does not give design-based standard errors for survey data – a sampling weight changes the point estimate, not the variance under the sampling design; for that, resample primary sampling units with block = instead. The bootstrap resamples rows exactly as without weights and carries each drawn row's weight along (the resampling probabilities themselves are not reweighted).
time	Optional time index for panel/time-series bootstrap resampling (round9 F), given as a bare column name (time = t) or a string (time = "t"); required for bootstrap_scheme %in% c("moving_block", "stationary"). Must be strictly increasing within every block unit's rows as they already appear in the data – this is never sorted for you; sort data by (block, time) first if it isn't already.
bootstrap_scheme	One of "iid" (plain row resampling, the default when block is not given), "cluster" (resample whole block units, the default when block is given – what block = has always done), "moving_block" or "stationary" (block-resample each unit's own time series; see Kunsch 1989 / Politis and Romano 1994). The two block schemes need time; block is optional for them (the whole data is treated as one unit when omitted). With overlapping blocks the out-of-bag set is smaller and more temporally correlated than under iid resampling, so out-of-bag error (oob_evaluation) is optimistic relative to genuine forecasting performance; each bootstrap replicate's realised out-of-bag fraction is stored as \$oob_fraction next to \$boot_id.
block_length	Block length for bootstrap_scheme %in% c("moving_block", "stationary"); NULL (the default) uses ceiling($T^{(1/3)}$) for each unit's own length T (a mes-

	sage names the value(s) used, once, at the original fit – not on every bootstrap replicate).
family	An <code>evinf_family()</code> object, or a string as shorthand for its count argument (round9 E.0/E.1/E.2), e.g. <code>family = "poisson"</code> . The default reproduces today's negative-binomial, mixture-zero model exactly. <code>count = "poisson"</code> drops <code>Alpha.NB</code> entirely (not merely fixes it): it is absent from <code>par.all</code> , <code>coef()</code> , <code>vcov()</code> , <code>confint()</code> and <code>tidy()</code> , and shown as absent (not NA) in <code>summary()</code> and <code>glance()</code> . <code>zero = "hurdle"</code> makes the zero state own every zero (rather than competing with the count state for them) and zero-truncates the count state; verified to match <code>pscl::hurdle()</code> 's coefficients and log-likelihood to numerical precision when the extreme-value state is unreachable.
boot_seed	Optional bootstrap seed for reproducibility. When supplied it is used as-is; when NULL a seed is drawn and recorded, so <code>object\$boot_seeds</code> is always populated for a bootstrapped model (see <code>add_bootstraps</code>).
start_seed	Optional seed for the perturbed starts when <code>control\$n_starts > 1</code> (round10 G.1); recorded as <code>object\$start_seed</code> (NULL for the default <code>n_starts = 1</code>). Unused otherwise.
control	An <code>evinf_control()</code> object holding the EM tuning settings.
max.diff.par,	max.no.em.steps, max.no.em.steps.warmup, c.lim,
prune.c.range,	max.upd.par.pl.multinomial, max.upd.par.nb,
max.upd.par.pl,	no.m.bfgs.steps.multinomial, no.m.bfgs.steps.nb,
no.m.bfgs.steps.pl,	pdf.pl.type, eta.int, init.Beta.multinom.PL,
init.Beta.NB, init.Beta.PL, init.Alpha.NB, init.C	
	Deprecated. Pass these through <code>control = evinf_control(...)</code> ; supplying one directly overrides the corresponding control element and emits a warning. See <code>evinf_control</code> . Will be removed in a future release.
verbose	Should progress be printed for the first run of <code>evinb</code>

Value

An object of class 'evinb'

Parallel processing

Bootstrap fits (and the per-bootstrap work in `add_bootstraps`, `lr_test`, `compare_models`, `predict.evinb` and `marginal_effects`) are dispatched with **furrr** on top of a **future** plan. Set the plan once for your session and leave `multicore` at its default:

```
future::plan(future::multisession, workers = 8)
progressr::handlers(global = TRUE) # opt in to progress bars
model <- evzinb(y ~ x1 + x2 + x3, data = genevzinb2, n_bootstraps = 1000)
```

Any future backend works (`multisession`, `cluster`, `callr`, a HPC batchtools plan, ...). As a convenience `multicore = TRUE` sets a temporary `multisession` plan for the single call and restores the previous plan on exit. Large models may need `options(future.globals.maxSize = <bytes>)` to raise the default limit on the data shipped to each worker.

Reproducibility

The bootstrap **resample indices** (`object$bootstraps[[i]]$boot_id`) are fully determined by `boot_seed` and are independent of the **future** backend, the number of workers, and chunking — a fixed seed always draws the same resamples.

The **fitted coefficients** are reproducible only to within floating-point noise. BLAS operations are not bit-reproducible across processes, so a sequential run and a multisession run of the same seed can differ in the last few digits; and because the EM log-likelihood can have close local optima, an occasional replicate converges to a different one under a different backend. For replication material, record the future plan and `utils::sessionInfo()` alongside `boot_seed`, and produce the canonical set of estimates under a single fixed plan.

Examples

```
data(genevzinb2)
model <- evinb(y~x1+x2+x3,data=genevzinb2, n_bootstraps = 5)
```

```
data(hks)
hks_mod <- evinb(
  osvAll ~ troopLag + policeLag + militaryobserversLag + epduration +
  lntpop + brv_AllLag_log + osvAllLagDum + incomp,
  formula_pareto = ~ log1p(troopLag),
  data = hks, n_bootstraps = 2, multicore = FALSE
)
```

evinf-s3-accessors	<i>logLik / nobs / model.frame / formula / terms for evzinb / evinb models</i>
--------------------	--

Description

`logLik / nobs / model.frame / formula / terms` for `evzinb / evinb` models

Usage

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

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

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

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

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

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

## S3 method for class 'evzinb'
formula(x, component = "count", ...)

## S3 method for class 'evinb'
formula(x, component = "count", ...)

## S3 method for class 'evzinb'
terms(x, component = "count", ...)

## S3 method for class 'evinb'
terms(x, component = "count", ...)
```

Arguments

object	A fitted model.
...	Unused.
formula	A fitted model (for the formula method).
x	A fitted model (for the terms method).
component	Which component ("count" by default) for formula() / terms().

Value

As for the corresponding **stats** generic. `logLik()` carries `df = length(object$par.all)` and `nobs`, so `AIC()` / `BIC()` work and match `object$AIC` / `object$BIC`.

evinf-s3-predict	<i>Fitted values, residuals and simulations from an evzinb / evinb model</i>
------------------	--

Description

Fitted values, residuals and simulations from an evzinb / evinb model

Usage

```
## S3 method for class 'evzinb'
fitted(object, type = c("harmonic", "explog", "counts", "pareto_alpha"), ...)

## S3 method for class 'evinb'
fitted(object, type = c("harmonic", "explog", "counts", "pareto_alpha"), ...)
```

```
## S3 method for class 'evzinb'
residuals(
  object,
  type = c("response", "quantile"),
  seed = NULL,
  newdata = NULL,
  ...
)

## S3 method for class 'evinb'
residuals(
  object,
  type = c("response", "quantile"),
  seed = NULL,
  newdata = NULL,
  ...
)

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

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

Arguments

<code>object</code>	A fitted model.
<code>type</code>	For <code>fitted()</code> , one of "harmonic", "explog", "counts", "pareto_alpha". For <code>residuals()</code> , "response" (y — harmonic prediction) or "quantile" (randomized quantile residuals from the mixture CDF).
<code>...</code>	For <code>fitted()</code> , forwarded to <code>predict()</code> (round11 A4: e.g. <code>clamp_alpha_pl</code> for <code>type = "explog"</code>); unused otherwise.
<code>seed</code>	Optional RNG seed for the randomized quantile residuals / <code>simulate()</code> . If given, the caller's RNG state is restored on exit (the seed only affects this call's draws).
<code>newdata</code>	Optional data to simulate for, or (round10 I.2, for <code>residuals()</code>) to compute residuals for instead of the estimation data; the response column (named by the count formula's left-hand side) must be present.
<code>nsim</code>	Number of simulated response vectors.

Value

`fitted()` / `residuals()` return a numeric vector; `simulate()` a data frame with columns `sim_1`, `sim_2`, ..., with a "seed" attribute following the [simulate](#) convention (the given seed, or, if none was given, the RNG state before the draws were made).

evinf_bench_data

Synthetic data from an EVZINB model with known true parameters

Description

Generates n observations from an EVZINB mixture with fixed, hand-chosen true parameters (see Details) – formula_nb = $y \sim x_1 + x_2 + x_3$, formula_zi/formula_evi sharing the same three co-variates, formula_pareto depending on x_1 alone. Meant for benchmarking (inst/bench/bench_evinf.R) and scale testing, not as a realistic applied example – see [genevzinb2](#) for that.

Usage

```
evinf_bench_data(n = 1000, seed = NULL)
```

Arguments

n	Number of observations.
seed	Optional seed; the caller's .Random.seed is left untouched either way (the same convention as simulate()/predict(type = "draws")).

Details

True parameters (on each component's own scale, the same parameterisation coef() returns):

- Zero-state log-odds against the count-state baseline: $-0.5 + 0.8x_1 - 0.5x_2$.
- Evi-state log-odds against the count-state baseline: $-2.5 + 0.5x_1 + 0.3x_3$ (a deliberately small share, matching an extreme-value-inflation process being the rare state).
- Count state (negative binomial, log link): $\mu = \exp(1.5 + 0.4x_1 - 0.3x_2 + 0.2x_3)$, dispersion $\alpha_{NB} = 0.8$.
- Evi state (the discretised Pareto tail the package fits throughout): threshold $C = 50$, shape $\alpha_{PL} = \exp(0.6 + 0.15x_1)$ (kept well above 0 for any simulated x_1 , so the tail never collapses).

Value

A tibble with columns y, x1, x2, x3.

Examples

```
d <- evinf_bench_data(200, seed = 1)

m <- evzinb(y ~ x1 + x2 + x3, data = d, bootstrap = FALSE)
```

evinf_control	<i>Control settings for evzinb() / evinb()</i>
---------------	--

Description

Bundles the ~20 tuning arguments of the EM fitting routine into a single object, in the style of `stats::glm.control()` or `pscl::zeroinfl.control()`. Pass the result as the control argument of `evzinb()` / `evinb()`.

Usage

```
evinf_control(
  max.diff.par = 0.01,
  max.no.em.steps = 500,
  max.no.em.steps.warmup = 5,
  c.lim = NULL,
  prune.c.range = FALSE,
  max.upd.par.zc.multinomial = 0.5,
  max.upd.par.pl.multinomial = 0.5,
  max.upd.par.nb = 0.5,
  max.upd.par.pl = 0.5,
  no.m.bfgs.steps.multinomial = 3,
  no.m.bfgs.steps.nb = 3,
  no.m.bfgs.steps.pl = 3,
  pdf.pl.type = c("approx", "exact"),
  eta.int = c(-1, 1),
  init.Beta.multinom.ZC = NULL,
  init.Beta.multinom.PL = NULL,
  init.Beta.NB = NULL,
  init.Beta.PL = NULL,
  init.Alpha.NB = 0.01,
  init.C = NULL,
  alpha_floor = 0.001,
  coef_limit = 50,
  max.c.iter = 50,
  alpha_pl_floor = 0.01,
  n_starts = 1L,
  start_jitter = 0.5,
  chunk_size = NULL
)
```

Arguments

<code>max.diff.par</code>	EM convergence tolerance: the algorithm has converged when the maximum absolute change in the parameter estimates falls below this. The EM log-likelihood can have close local optima; a tighter tolerance reduces (without eliminating) the
---------------------------	--

	chance that two future backends run with the same <code>boot_seed</code> settle in different ones (see the Reproducibility section of ?evzinb).
<code>max.no.em.steps</code>	Maximum number of EM steps.
<code>max.no.em.steps.warmup</code>	Number of EM steps in each warm-up round.
<code>c.lim</code>	NULL or a numeric vector of length 2. The candidate set for C_{EV} is the unique observed response values within this range. NULL (the default) uses a data-driven range (see Details).
<code>prune.c.range</code>	FALSE, or a number in $\setminus[0, 1)$: thin the candidate set to about $\text{length}(c.lim) * (1 - \text{prune.c.range})$ values.
<code>max.upd.par.zc.multinomial,</code> <code>max.upd.par.nb, max.upd.par.pl</code>	<code>max.upd.par.pl.multinomial,</code> Maximum parameter-change step sizes for the zero-inflation, extreme-value inflation, count and Pareto components.
<code>no.m.bfgs.steps.multinomial, no.m.bfgs.steps.nb, no.m.bfgs.steps.pl</code>	Number of BFGS steps per M-step for the multinomial, count and Pareto blocks.
<code>pdf.pl.type</code>	Which Pareto-block derivatives the M-step uses for β_{PL} : "approx" (the default) uses the gradient/Hessian of the <i>continuous</i> Pareto log-density; "exact" uses the gradient/Hessian of the <i>discretised</i> Pareto log-pmf that the likelihood itself always uses (this only changes the Newton step taken each M-step, not what is being maximised). The two typically converge to nearly the same estimates; "exact" can help when "approx"'s steps are poorly scaled for a heavy-tailed fit.
<code>eta.int</code>	Interval for the eta line search, a numeric vector of length 2.
<code>init.Beta.multinom.ZC,</code> <code>init.Beta.PL</code>	<code>init.Beta.multinom.PL,</code> <code>init.Beta.NB,</code> Optional starting values for the component coefficient vectors (NULL starts at zero).
<code>init.Alpha.NB</code>	Starting value for the negative-binomial dispersion.
<code>init.C</code>	NULL or a starting value for C_{EV} within <code>c.lim</code> . NULL uses the median of the candidate set.
<code>alpha_floor, coef_limit</code>	Thresholds for flagging a bootstrap replicate as <i>degenerate</i> (<code>\$degenerate, \$degenerate_reason</code>), so it is excluded from bootstrap summaries by default (see <code>exclude_degenerate</code>) and counted in <code>glance()</code> / <code>failed_bootstraps()</code> . A replicate is <i>degenerate</i> if (in this order): its EM did not converge; any fitted coefficient in a linear predictor (<code>Beta.*</code>) or the negative-binomial dispersion is non-finite or exceeds <code>coef_limit</code> in absolute value; or its smallest fitted Pareto shape is non-finite or below <code>alpha_floor</code> . <code>alpha_floor</code> (default 0.001) is deliberately low — it catches an outright tail collapse (shape near 0), not a merely heavy tail. <code>coef_limit</code> (default 50) is on the linear-predictor scale, where 50 is already extreme.
<code>max.c.iter</code>	Maximum number of outer ECME iterations (each one re-profiles C_{EV} over the candidate grid) per phase (warm-up, convergence). Guards against the profile oscillating between two candidate values forever. If the convergence-phase

	loop hits this cap, the fit's converge is set to FALSE (see <code>\$c_converged</code> to tell this apart from the EM inner loop not converging) and, for a full-sample fit, a <code>warning()</code> names the last two C_{EV} values visited. The warm-up phase is capped independently and recorded in <code>\$c_warmup_capped</code> (with its own <code>warning()</code> for a full-sample fit); it does not affect converge, since warm-up is a short exploratory phase and not settling there is not by itself a sign the fit failed.
<code>alpha_pl_floor</code>	Floor for the fitted Pareto shape α_{PL} (round9 0.1). Some observations' fitted α_{PL} can collapse toward 0 (heaviest possible tail); several downstream quantities involve $\exp(1/\alpha_{PL})$ or $1/\alpha_{PL}$ and silently return Inf or an astronomically large finite number when that happens (<code>predict(type = "harmonic")</code>, <code>predict(type = "explog")</code>, the continuous-Pareto mixture quantile, and the <code>\$fitted</code> tail summaries). Any fitted α_{PL} below <code>alpha_pl_floor</code> is clamped to it before these calculations, with a <code>warning()</code> naming how many observations were clamped; the unclamped values are still what <code>glance()</code>'s <code>min_alpha_pl</code> and <code>print()</code>'s note report, so a collapsed EV shape stays visible.
<code>n_starts</code>	Number of starting points for the full-sample fit (round10 G.1, audit §5.10). 1 (the default) fits once from the default start, exactly as before this argument existed. With <code>n_starts > 1</code>, that default start plus <code>n_starts - 1</code> perturbed starts each run through the EM (in parallel via <code>evinf_pmap</code>, seeded from <code>start_seed</code>), and the replicate with the highest final log-likelihood is kept – the EM can land in different local optima depending on floating-point accumulation order, and multiple starts are the principled response. Every replicate's outcome is recorded in <code>object\$starts</code>; bootstrap replicates warm-start from the winning solution, same as they would from a single-start fit.
<code>start_jitter</code>	Standard deviation of the $N(0, \text{start_jitter}^2)$ perturbation applied to each component's starting coefficients for the <code>n_starts - 1</code> perturbed starts; their starting C_{EV} is drawn uniformly from the candidate grid instead. Unused when <code>n_starts <= 1</code>.
<code>chunk_size</code>	Number of bootstrap replicates (or starts) handed to a worker at a time in <code>evinf_pmap</code> (round10 J.2, audit §5.11). NULL (the default) picks <code>max(1, ceiling(B / (4 * future::nbrOfWorkers())))</code> at dispatch time (B the number of replicates being dispatched) – four chunks per worker, balancing per-task scheduling overhead (worse for B far larger than the worker count and <code>chunk_size = 1</code>) against a straggler chunk leaving workers idle near the end (worse for a <code>chunk_size</code> close to <code>B / nbrOfWorkers()</code>). Chunk size only changes how replicates are grouped for dispatch, never which <code>boot_ids</code> they draw (each replicate's L'Ecuyer stream comes from its position in the full sequence, not from the chunk it lands in), so bootstrap output is identical across chunk sizes for the same seed. Set explicitly to override, e.g. 1L to restore the pre-J.2 default.

Details

When `c.lim = NULL`, `evzinb()` / `evinb()` choose a range from the data: the lower bound is the 90th percentile of the positive response values (rounded down to an observed value), the upper bound is the third-largest unique response value; if fewer than 10 unique positive values lie in that range the lower bound drops to the 75th percentile. The chosen range is printed with a message and stored in `object$control$c.lim`.

Value

A list of class "evinf_control".

Examples

```
evinf_control(max.no.em.steps = 300, c.lim = c(20, 500))
```

evinf_family	<i>Model family for evzinb() / evinb() (round9 E.0)</i>
--------------	---

Description

Specifies the distributional family of the count state (count) and of the zero state (zero). One object, so later additions (a truncated-count state, a geometric count) do not add more arguments to [evzinb\(\)](#) / [evinb\(\)](#).

Usage

```
evinf_family(count = c("nbinom", "poisson"), zero = c("mixture", "hurdle"))
```

Arguments

count	"nbinom" (the default; a negative-binomial count state) or "poisson" (a Poisson count state – Alpha.NB is dropped entirely: not in par.all, not in coef()/vcov()/confint()/tidy(), shown as absent (not NA) in summary()/glance()).
zero	"mixture" (the default; the existing zero-inflation mixture, where a zero can come from either the zero state or the count state) or "hurdle" (the zero state owns every zero and the count state is zero-truncated; evzinb() only – evinb() has no zero state to hurdle over and errors if asked for one).

Details

Sign convention against [pscl](#) (round10 0.9, review §7): for zero = "hurdle", [hurdle](#)'s zero-hurdle component models $P(Y > 0)$, while [evinf](#)'s zero-state coefficients model $P(\text{zero state})$ directly – so a fitted [evzinb\(family = evinf_family\(zero = "hurdle"\)\)](#) model's zero coefficients (Beta.multinom.ZC) are the exact negatives of the corresponding [pscl::hurdle\(..., dist = "negbin"\)](#) zero coefficients, everything else about the fit being equal (verified in [test-hurdle.R](#)). This is a clean reparameterisation because a hurdle's zero state owns every zero outright. zero = "mixture" (the default) has no such simple relationship to [zeroinfl](#)'s zero-inflation coefficients: a mixture's zeros can also arise from the count state, so the two models' zero components are not a reparameterisation of each other.

Value

An object of class "evinf_family": a list with elements count and zero.

Examples

```
evinf_family()
evinf_family(count = "poisson")
```

evzinb	<i>Running an extreme value and zero inflated negative binomial model with bootstrapping</i>
--------	--

Description

Running an extreme value and zero inflated negative binomial model with bootstrapping

Usage

```
evzinb(
  formula_nb,
  formula_zi = NULL,
  formula_evi = NULL,
  formula_pareto = NULL,
  data,
  bootstrap = TRUE,
  n_bootstraps = 100,
  multicore = NULL,
  ncores = NULL,
  block = NULL,
  weights = NULL,
  time = NULL,
  bootstrap_scheme = NULL,
  block_length = NULL,
  family = evinf_family(),
  boot_seed = NULL,
  start_seed = NULL,
  control = evinf_control(),
  max.diff.par,
  max.no.em.steps,
  max.no.em.steps.warmup,
  c.lim,
  prune.c.range,
  max.upd.par.zc.multinomial,
  max.upd.par.pl.multinomial,
  max.upd.par.nb,
  max.upd.par.pl,
  no.m.bfgs.steps.multinomial,
  no.m.bfgs.steps.nb,
  no.m.bfgs.steps.pl,
  pdf.pl.type,
  eta.int,
```

```

init.Beta.multinom.ZC,
init.Beta.multinom.PL,
init.Beta.NB,
init.Beta.PL,
init.Alpha.NB,
init.C,
verbose = FALSE
)

```

Arguments

formula_nb	Formula for the negative binomial (count) component of the model. May include an <code>offset()</code> term ($\mu_{NB} = \exp(x'\beta + offset)$).
formula_zi	Formula for the zero-inflation component of the model. If <code>NULL</code> taken as the same formula as <code>nb</code> , with any <code>offset()</code> term stripped (an offset applies only where it is written explicitly, never by inheritance). May include its own <code>offset()</code> term: since the zero-inflation logit is the log-odds of the zero state <i>against</i> the count state, an offset there shifts that log-odds, e.g. <code>offset(log(exposure))</code> makes a larger exposure relatively less likely to land in the structural-zero state.
formula_evi	Formula for the extreme-value inflation component of the model. If <code>NULL</code> taken as the same formula as <code>nb</code> , offset stripped as above. May include its own <code>offset()</code> term (e.g. <code>offset(log(population))</code> for a probability of an extreme event that scales with exposure), read the same way: it shifts the EVI log-odds against the count state.
formula_pareto	Formula for the pareto (extreme value) component of the model. If <code>NULL</code> taken as the same formula as <code>nb</code> , offset stripped as above. <code>offset()</code> is not supported here (errors if present): an offset on a shape parameter has no clear reading.
data	data to run the model on
bootstrap	Should bootstrapping be performed. Needed to obtain standard errors and p-values
n_bootstraps	Number of bootstraps to run. For use of bootstrapped p-values, at least 1,000 bootstraps are recommended. For approximate p-values, a lower number can be sufficient
multicore	Bootstrap parallelisation shortcut. The default (<code>NULL</code>) respects whatever future plan is currently set (see the Parallel processing section). <code>TRUE</code> sets a temporary <code>multisession</code> plan for the duration of the call; <code>FALSE</code> forces sequential execution. The previous plan is always restored on exit.
ncores	Number of workers when <code>multicore = TRUE</code> . Default (<code>NULL</code>) is one less than the number of available cores. Ignored when <code>multicore</code> is <code>NULL</code> or <code>FALSE</code> .
block	Optional case-identifier column for block bootstrapping, given either as a bare column name (<code>block = id</code>) or a string (<code>block = "id"</code>). Note that the bundled <code>hks</code> data contain no conflict identifier, so the conflict-level cluster bootstrap in Randahl and Vegelius (2024) cannot be reproduced from them directly (see <code>?hks</code>).
weights	Optional observation weights (round9 D.2), given as a bare column name (<code>weights = wt</code>), a string naming a column (<code>weights = "wt"</code>), or a numeric vector. Must

be positive and finite; need not be integers (analytic weights are allowed, not just frequency counts). **Frequency-weight semantics:** every observation's contribution to the log-likelihood and to the EM/M-step accumulations is multiplied by its weight, and `nobs()` – and therefore AIC, BIC and the approximate t-based p-values – use `sum(weights)`, not the row count (see `sum_weights` in [glance.evzinb](#)). This interpretation of AIC/BIC assumes the weights really are frequency weights (repeat-count equivalents); for analytic weights the information-criterion values are still computed this way but their usual interpretation is weaker. `weights` **does not give design-based standard errors for survey data** – a sampling weight changes the point estimate, not the variance under the sampling design; for that, resample primary sampling units with `block =` instead. The bootstrap resamples rows exactly as without weights and carries each drawn row's weight along (the resampling probabilities themselves are not reweighted).

time	Optional time index for panel/time-series bootstrap resampling (round9 F), given as a bare column name (<code>time = t</code>) or a string (<code>time = "t"</code>); required for <code>bootstrap_scheme</code> <code>%in% c("moving_block", "stationary")</code> . Must be strictly increasing within every block unit's rows as they already appear in the data – this is never sorted for you; sort data by (<code>block</code> , <code>time</code>) first if it isn't already.
bootstrap_scheme	One of <code>"iid"</code> (plain row resampling, the default when <code>block</code> is not given), <code>"cluster"</code> (resample whole block units, the default when <code>block</code> is given – what <code>block =</code> has always done), <code>"moving_block"</code> or <code>"stationary"</code> (block-resample each unit's own time series; see Kunsch 1989 / Politis and Romano 1994). The two block schemes need <code>time</code> ; <code>block</code> is optional for them (the whole data is treated as one unit when omitted). With overlapping blocks the out-of-bag set is smaller and more temporally correlated than under iid resampling, so out-of-bag error (oob_evaluation) is optimistic relative to genuine forecasting performance; each bootstrap replicate's realised out-of-bag fraction is stored as <code>\$oob_fraction</code> next to <code>\$boot_id</code> .
block_length	Block length for <code>bootstrap_scheme</code> <code>%in% c("moving_block", "stationary")</code> ; NULL (the default) uses <code>ceiling(T^(1/3))</code> for each unit's own length <code>T</code> (a message names the value(s) used, once, at the original fit – not on every bootstrap replicate).
family	An <code>evinf_family()</code> object, or a string as shorthand for its count argument (round9 E.0/E.1/E.2), e.g. <code>family = "poisson"</code> . The default reproduces today's negative-binomial, mixture-zero model exactly. <code>count = "poisson"</code> drops Alpha.NB entirely (not merely fixes it): it is absent from <code>par.all</code> , <code>coef()</code> , <code>vcov()</code> , <code>confint()</code> and <code>tidy()</code> , and shown as absent (not NA) in <code>summary()</code> and <code>glance()</code> . <code>zero = "hurdle"</code> makes the zero state own every zero (rather than competing with the count state for them) and zero-truncates the count state; verified to match <code>pscl::hurdle()</code> 's coefficients and log-likelihood to numerical precision when the extreme-value state is unreachable.
boot_seed	Optional bootstrap seed for reproducibility. When supplied it is used as-is; when NULL a seed is drawn and recorded, so <code>object\$boot_seeds</code> is always populated for a bootstrapped model (see add_bootstraps).
start_seed	Optional seed for the perturbed starts when <code>control\$n_starts > 1</code> (round10

	G.1); recorded as <code>object\$start_seed</code> (NULL for the default <code>n_starts = 1</code> , same as <code>boot_seed</code> for an unbootstrapped fit). Unused otherwise.
<code>control</code>	An <code>evinf_control()</code> object holding the EM tuning settings (tolerances, candidate range for C_{EV} , BFGS steps, starting values, ...).
<code>max.diff.par,</code> <code>c.lim,</code> <code>max.upd.par.pl.multinomial,</code> <code>no.m.bfgs.steps.multinomial,</code> <code>pdf.pl.type,</code> <code>init.Beta.NB,</code>	<code>max.no.em.steps,</code> <code>prune.c.range,</code> <code>max.upd.par.nb,</code> <code>no.m.bfgs.steps.nb,</code> <code>eta.int,</code> <code>init.Beta.PL,</code>
<code>max.no.em.steps.warmup,</code> <code>max.upd.par.zc.multinomial,</code> <code>max.upd.par.pl,</code> <code>no.m.bfgs.steps.pl,</code> <code>init.Beta.multinom.ZC,</code> <code>init.Beta.multinom.PL,</code> <code>init.Alpha.NB,</code> <code>init.C</code>	
	Deprecated. These EM tuning arguments still work but should be passed through <code>control = evinf_control(...)</code> ; supplying one directly overrides the corresponding control element and emits a warning. See <code>evinf_control</code> for their meaning. Will be removed in a future release.
<code>verbose</code>	Logical: should progress of the full run of the model be tracked?

Value

An object of class 'evzinb'

Parallel processing

Bootstrap fits (and the per-bootstrap work in `add_bootstraps`, `lr_test`, `compare_models`, `predict.evzinb` and `marginal_effects`) are dispatched with **furrr** on top of a **future** plan. Set the plan once for your session and leave multicore at its default:

```
future::plan(future::multisession, workers = 8)
progressr::handlers(global = TRUE) # opt in to progress bars
model <- evzinb(y ~ x1 + x2 + x3, data = genevzinb2, n_bootstraps = 1000)
```

Any future backend works (multisession, cluster, callr, a HPC batchtools plan, ...). As a convenience `multicore = TRUE` sets a temporary multisession plan for the single call and restores the previous plan on exit. Large models may need `options(future.globals.maxSize = <bytes>)` to raise the default limit on the data shipped to each worker.

Reproducibility

The bootstrap **resample indices** (`object$bootstraps[[i]]$boot_id`) are fully determined by `boot_seed` and are independent of the **future** backend, the number of workers, and chunking — a fixed seed always draws the same resamples.

The **fitted coefficients** are reproducible only to within floating-point noise. BLAS operations are not bit-reproducible across processes, so a sequential run and a multisession run of the same seed can differ in the last few digits; and because the EM log-likelihood can have close local optima, an occasional replicate converges to a different one under a different backend. For replication material, record the future plan and `utils::sessionInfo()` alongside `boot_seed`, and produce the canonical set of estimates under a single fixed plan.

Examples

```

data(genevzinb2)
model <- evzinb(y~x1+x2+x3,data=genevzinb2, n_bootstraps = 5)

# Peacekeeping and one-sided violence, with an in-formula log1p() transform
# for the Pareto (extreme-value) component (see `?hks`).

data(hks)
hks_mod <- evzinb(
  osvAll ~ troopLag + policeLag + militaryobserversLag + epduration +
  lntpop + brv_AllLag_log + osvAllLagDum + incomp,
  formula_pareto = ~ log1p(troopLag),
  data = hks, n_bootstraps = 2, multicore = FALSE
)
summary(hks_mod)
glance(hks_mod)
predict(hks_mod, type = "harmonic")

```

evzinbcomp-tidiers	<i>Stacked tidy / glance for a model comparison</i>
--------------------	---

Description

Stacked tidy / glance for a model comparison

Usage

```

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

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

```

Arguments

x	An evzinbcomp object.
...	Passed to the per-model tidy() / glance().

Value

A tibble with a leading model column.

failed_bootstraps	<i>Inspect the bootstrap replicates that failed or came out degenerate</i>
-------------------	--

Description

Bootstrap fits that error are kept as try-error objects rather than discarded; fits that converge but whose extreme-value tail is so heavy that summaries from them are effectively unbounded are flagged *degenerate* (see `alpha_floor` in [evinf_control](#)). This returns both, which are useful diagnostics (e.g. a resample with a singular design, or one whose Pareto shape collapsed).

Usage

```
failed_bootstraps(object)
```

Arguments

`object` A fitted evzinb / evinb model.

Value

A tibble with columns `id`, `type` ("error", "degenerate" or "not_converged" – round10 G.3: a replicate that ran without erroring and isn't degenerate, but whose converge is FALSE, was previously invisible here), `message` (the error text, the degeneracy reason, or NA for "not_converged" – see `n_em_steps/c_converged/ c_warmup_capped` there instead) and, for "not_converged" rows, `n_em_steps`, `c_converged`, `c_warmup_capped` (NA for "error"/"degenerate" rows); zero rows when every replicate is usable and converged.

Examples

```
data(genevzinb2)
model <- evzinb(y ~ x1 + x2 + x3, data = genevzinb2, n_bootstraps = 5)
failed_bootstraps(model)
```

genevzinb	<i>Simulated data from the EVZINB distribution</i>
-----------	--

Description

A simulated dataset of 1,000 observations with one dependent and three independent variables generated using the EVZINB distribution

Usage

```
genevzinb
```

Format

```
## 'genevzinb' A tibble with 1,000 rows and 4 columns:

y Dependent variable following the EVZINB distribution
x1, x2, x3 Continuous independent variables following the standard normal distribution
```

genevzinb2	<i>Simulated data from the EVZINB distribution</i>
------------	--

Description

A simulated dataset of 100 observations with one dependent and three independent variables generated using the EVZINB distribution

Usage

```
genevzinb2
```

Format

```
## 'genevzinb2' A tibble with 100 rows and 4 columns:

y Dependent variable following the EVZINB distribution
x1, x2, x3 Continuous independent variables following the standard normal distribution
```

glance.evinb	<i>EVZINB and EVINB glance functions</i>
--------------	--

Description

EVZINB and EVINB glance functions

Usage

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

Arguments

```
x          An EVZINB or EVINB object
...        Further arguments to be passed to glance()
```

Value

A one-row tibble of goodness-of-fit statistics: number of observations and parameters, family (round9 E.0, e.g. "nbinom/mixture" or "poisson/mixture"), alpha_NB (NA when the family has none, round9 E.1), C_EV, AIC, BIC, log-likelihood, whether the EM algorithm converged (converged) and whether the C_EV profile settled within max.c.iter (c_converged; NA for a model fitted before this field existed), the number of EM steps, the number of observations at or above C_EV, the smallest fitted Pareto shape (min_alpha_pl; see alpha_pl_floor in [evinf_control](#)), sum_weights (round9 D.2: the sum of weights =, equal to nobs for an unweighted fit – this, not nobs, is what aic/bic and the model’s degrees of freedom are computed from), and, for a bootstrapped model, the bootstrap replicate counts (NA otherwise): n_bootstraps (usable), n_failed_bootstraps, n_degenerate_bootstraps — these three partition the number of replicates requested (see [evinf_control](#) for "degenerate") — and n_c_on_boundary, the number of replicates whose C_EV landed on a candidate-grid endpoint (informational, not counted as degenerate); oob_fraction_mean/oob_fraction_min/oob_fraction_max (round10 0.9), the mean/min/max of each usable replicate’s out-of-bag row fraction (NA without bootstraps) – most informative for the block schemes, where overlapping blocks change how much of the data a replicate leaves out; n_starts/n_starts_at_best (round10 G.1, NA for n_starts <= 1), how many perturbed starts were tried and how many of them reached the winning log-likelihood (within 1e-4); median_boot_em_steps/ n_boot_c_capped (round10 G.3, NA without bootstraps), the median EM step count across usable replicates and how many had their C_EV profile hit max.c.iter without settling in either phase.

See Also

[glance](#)

Examples

```
data(genevzinb2)
model <- evinb(y~x1+x2+x3,data=genevzinb2, n_bootstraps = 5)
glance(model)
```

glance.evzinb	<i>EVZINB and EVINB glance functions</i>
---------------	--

Description

EVZINB and EVINB glance functions

Usage

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

Arguments

- x An EVZINB or EVINB object
- ... Further arguments to be passed to glance()

Value

A one-row tibble of goodness-of-fit statistics: number of observations and parameters, family (round9 E.0, e.g. "nbinom/mixture" or "poisson/mixture"), alpha_NB (NA when the family has none, round9 E.1), C_EV, AIC, BIC, log-likelihood, whether the EM algorithm converged (converged) and whether the C_EV profile settled within max.c.iter (c_converged; NA for a model fitted before this field existed), the number of EM steps, the number of observations at or above C_EV, the smallest fitted Pareto shape (min_alpha_pl; see alpha_pl_floor in [evinf_control](#)), sum_weights (round9 D.2: the sum of weights =, equal to nobs for an unweighted fit – this, not nobs, is what aic/bic and the model's degrees of freedom are computed from), and, for a bootstrapped model, the bootstrap replicate counts (NA otherwise): n_bootstraps (usable), n_failed_bootstraps, n_degenerate_bootstraps — these three partition the number of replicates requested (see [evinf_control](#) for "degenerate") — and n_c_on_boundary, the number of replicates whose C_EV landed on a candidate-grid endpoint (informational, not counted as degenerate); oob_fraction_mean/oob_fraction_min/oob_fraction_max (round10 0.9), the mean/min/max of each usable replicate's out-of-bag row fraction (NA without bootstraps) – most informative for the block schemes, where overlapping blocks change how much of the data a replicate leaves out; n_starts/n_starts_at_best (round10 G.1, NA for n_starts <= 1), how many perturbed starts were tried and how many of them reached the winning log-likelihood (within 1e-4); median_boot_em_steps/ n_boot_c_capped (round10 G.3, NA without bootstraps), the median EM step count across usable replicates and how many had their C_EV profile hit max.c.iter without settling in either phase.

See Also

[glance](#)

Examples

```
data(genevzinb2)
model <- evzinb(y~x1+x2+x3,data=genevzinb2, n_bootstraps = 5)
glance(model)
```

glance.zinbboot

zinbboot and nbboot glance functions

Description

zinbboot and nbboot glance functions

Usage

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

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

Arguments

`x` An nbboot or zinbboot object
`...` Further arguments to be passed to glance()

Value

A one-row tibble of goodness-of-fit statistics, including whether the full-sample model converged and the number of (failed) bootstraps.

See Also

[glance](#)

Examples

```
data(genevzinb2)
model <- evzinb(y~x1+x2+x3,data=genevzinb2, n_bootstraps = 5)
zinb_comp <- compare_models(model)
glance(zinb_comp$zinb)
```

```
data(genevzinb2)
model <- evzinb(y~x1+x2+x3,data=genevzinb2, n_bootstraps = 5)
zinb_comp <- compare_models(model)
glance(zinb_comp$nb)
```

glance.zipboot	<i>zipboot and poissonboot glance functions</i>
----------------	---

Description

zipboot and poissonboot glance functions

Usage

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

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

Arguments

`x` A poissonboot or zipboot object
`...` Further arguments to be passed to glance()

Value

A one-row tibble of goodness-of-fit statistics, including whether the full-sample model converged and the number of (failed) bootstraps. `alpha` is NA: neither baseline has a dispersion parameter (round9 E.3).

See Also

[glance](#)

Examples

```
data(genevzinb2)
model <- evzinb(y~x1+x2+x3,data=genevzinb2, family = "poisson", n_bootstraps = 5)
zip_comp <- compare_models(model)
glance(zip_comp$zip)
```

```
data(genevzinb2)
model <- evzinb(y~x1+x2+x3,data=genevzinb2, family = "poisson", n_bootstraps = 5)
zip_comp <- compare_models(model)
glance(zip_comp$poisson)
```

gm_evzinb	<i>A goodness-of-fit gof tibble for GOF metrics when using modelsummary</i>
-----------	---

Description

A goodness-of-fit gof tibble for GOF metrics when using modelsummary. The GM tibble can be used to obtain correct table output when making regression tables with modelsummary

Usage

```
gm_evzinb
```

Format

‘gm_evzinb’ A tibble with 11 rows and 3 columns:

raw The modelsummary/broom internal name for the statistic

clean The table output for the statistic

fmt The number of decimals reported for each statistic by default (can be adapted)

See Also

[gof_map_evinf\(\)](#), which builds this object (and lets you append rows); the regeneration script lives in ‘data-raw/gm_evzinb.R’.

gof_map_evinf

*Goodness-of-fit map for modelsummary***Description**

Returns the `gof_map` that `modelsummary::modelsummary()` needs to label and format the goodness-of-fit rows produced by `glance.evzinb()` / `glance.evinb()`. The bundled data object [gm_evzinb](#) is simply `gof_map_evinf()`.

Usage

```
gof_map_evinf(extra = NULL)
```

Arguments

`extra` Optional data frame / tibble of additional rows (columns `raw`, `clean`, `fmt`) to append, e.g. for a statistic added by a custom `glance()` method.

Details

The coefficient table produced by `tidy()` with `component = "all"` has one row per coefficient *per component*. `modelsummary::modelsummary()` cannot render that with its default arguments (the same limitation it has for `nnet::multinom()`); pass `shape = term + y.level ~ model` together with `gof_map = gof_map_evinf()`.

Value

A tibble with columns `raw` (the `glance()` column name, unchanged), `clean` (the title-case row label in the table) and `fmt` (default number of decimals). Rows: observations, parameters, `alpha_NB`, `C_EV`, observations at or above `C_EV`, log-likelihood, AIC, BIC, the usable / failed / degenerate bootstrap counts, the number of bootstrap replicates with `C_EV` on the candidate-grid boundary, and convergence (both of the EM loop and of the `C_EV` profile).

Examples

```
gof_map_evinf()
gof_map_evinf(extra = data.frame(raw = "n_em_steps", clean = "EM steps",
                                fmt = 0))

data(genevzinb2)
model <- evzinb(y ~ x1 + x2 + x3, data = genevzinb2, n_bootstraps = 5)
if (requireNamespace("modelsummary", quietly = TRUE) &&
    requireNamespace("broom", quietly = TRUE)) {
  modelsummary::modelsummary(
    model,
    shape = term + y.level ~ model,
    gof_map = gof_map_evinf()
  )
}
```

}

hks	<i>Replication data for Hultman, Kathman, and Shannon (2013) United Nations Peacekeeping and Civilian Protection in Civil War</i>
-----	---

Description

A reduced replication data set from Hultman et al. (2013) United Nations Peacekeeping and Civilian Protection in Civil War. Used to reproduce the results of Randahl and Vegelius (2024). To reproduce any other results from Hultman et al. (2013) please download the original replication dataset using the link under source.

Usage

hks

Format

A tibble with 3746 rows and 13 columns:

- osvAll** The number of observed fatalities from one-sided violence against civilians in the specified conflict-month
- troopLag** The number of UN military troops, in thousands (lagged)
- policeLag** The number of UN police, in thousands (lagged)
- militaryobserversLag** The number of UN military observers, in thousands (lagged)
- brv_AllLag_log** The natural logarithm of the total number of battle-related deaths in the conflict in the previous month, log-transformed as in Hultman et al. (2013)
- osvAllLagDum** A dummy variable taking the value 1 if any one-sided violence against civilians took place in the previous conflict month
- incomp** UCDP/PRIO incompatibility: 1 = territory, 2 = government
- epduration** The number of months the current conflict-episode has been ongoing
- Intpop** The natural logarithm of the population of the country in which the conflict takes place
- troopLag_log** ‘log1p(troopLag)’
- epdur_log** ‘log(epduration)’
- policeLag_log** ‘log1p(policeLag)’
- militaryobserversLag_log** ‘log1p(militaryobserversLag)’

Details

The UN personnel counts (‘troopLag’, ‘policeLag’, ‘militaryobserversLag’) are expressed in **thousands** of personnel. This is a rescaling of the original Hultman, Kathman, and Shannon replication data introduced for Randahl and Vegelius (2024).

The columns ending in ‘_log’ (‘troopLag_log’, ‘policeLag_log’, ‘militaryobserversLag_log’, ‘epdur_log’, ‘brv_AllLag_log’) are transforms that were pre-computed for the Pareto component in Randahl and Vegelius (2024). The transform follows the standard rule in the conflict literature: the personnel counts contain structural zeros and use ‘log1p()’, while an episode duration is always at least one month (at least two in these data) and uses ‘log()’, the conventional transform for durations. Since evinf 0.9.4 the component formulas accept in-formula transformations, so ‘log1p(troopLag)’, ‘log(epduration)’ etc. can be written directly in the formula and these pre-computed columns are no longer needed.

The bundled data carry **no conflict identifier**. The conflict-level cluster bootstrap reported in Randahl and Vegelius (2024) therefore cannot be reproduced from the bundled data alone; download the original replication data (link under ‘source’) and merge on your own conflict key to use block in `evzinb()` / `evinb()`.

Source

<https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/6EBCGA>

References

Hultman L, Kathman J, Shannon M (2013). “United Nations peacekeeping and civilian protection in civil war.” *American Journal of Political Science*, **57**(4), 875–891.

Randahl D, Vegelius J (2024). “Inference with Extremes: Accounting for Extreme Values in Count Regression Models.” *International Studies Quarterly*, **68**(4), sqae137. doi:10.1093/isq/sqae137.

lr_test	<i>Likelihood ratio test for individual variables of evzinb</i>
---------	---

Description

Likelihood ratio test for individual variables of evzinb

Usage

```
lr_test(  
  object,  
  vars,  
  single = TRUE,  
  bootstrap = FALSE,  
  multicore = NULL,  
  ncores = NULL,  
  exclude_degenerate = TRUE,  
  verbose = FALSE  
)
```

Arguments

object	EVZINB or EVINB object to perform likelihood ratio test on
vars	Either a list of character vectors with variable names which to be restricted in the LR test or a character vector of variable names. If a list, each character vector of the list will be run separately, allowing for multiple variables to be restricted as once. If a character vector, parameter 'single' can be used to determine whether all variables in the vector should be restricted at once (single = FALSE) or if the variables should be restricted one by one (single = TRUE)
single	Logical. Determining whether variables in 'vars' should be restricted individually (single = TRUE) or all at once (single = FALSE)
bootstrap	Should LR tests be conducted on each bootstrapped sample or only on the original sample.
multicore	Bootstrap parallelisation shortcut. The default (NULL) respects whatever future plan is currently set (see the Parallel processing section). TRUE sets a temporary <code>multisession</code> plan for the duration of the call; FALSE forces sequential execution. The previous plan is always restored on exit.
ncores	Number of workers when multicore = TRUE. Default (NULL) is one less than the number of available cores. Ignored when multicore is NULL or FALSE.
exclude_degenerate	Logical. When bootstrap = TRUE, also drop bootstrap replicates flagged degenerate (see <code>failed_bootstraps</code>), not just those that errored. Default TRUE.
verbose	Logical. Should the function be verbose?

Details

The likelihood ratio statistic is $2(\ell_{full} - \ell_{restricted})$ and is compared to a chi-square distribution with degrees of freedom equal to the number of design-matrix columns removed across all model components (so a restricted four-level factor contributes three degrees of freedom).

Value

A tibble with one row per performed LR test, or, when bootstrap = TRUE, a list with the summary tibble (results) and the per-bootstrap statistics (boot_results).

Parallel processing

Bootstrap fits (and the per-bootstrap work in `add_bootstraps`, `lr_test`, `compare_models`, `predict.evzinb` and `marginal_effects`) are dispatched with **furrr** on top of a **future** plan. Set the plan once for your session and leave multicore at its default:

```
future::plan(future::multisession, workers = 8)
progressr::handlers(global = TRUE) # opt in to progress bars
model <- evzinb(y ~ x1 + x2 + x3, data = genevzinb2, n_bootstraps = 1000)
```

Any future backend works (multisession, cluster, callr, a HPC batchtools plan, ...). As a convenience multicore = TRUE sets a temporary multisession plan for the single call and restores the previous plan on exit. Large models may need `options(future.globals.maxSize = <bytes>)` to raise the default limit on the data shipped to each worker.

Reproducibility

The bootstrap **resample indices** (`object$bootstraps[[i]]$boot_id`) are fully determined by `boot_seed` and are independent of the **future** backend, the number of workers, and chunking — a fixed seed always draws the same resamples.

The **fitted coefficients** are reproducible only to within floating-point noise. BLAS operations are not bit-reproducible across processes, so a sequential run and a `multisession` run of the same seed can differ in the last few digits; and because the EM log-likelihood can have close local optima, an occasional replicate converges to a different one under a different backend. For replication material, record the future plan and `utils::sessionInfo()` alongside `boot_seed`, and produce the canonical set of estimates under a single fixed plan.

Examples

```
data(genevzinb2)
model <- evzinb(y~x1+x2+x3,data=genevzinb2, n_bootstraps = 5)
lr_test(model,'x1')
```

marginal_effects

Average marginal effects for an evzinb / evinb model

Description

Average, over the data, of the numerical effect of each covariate on the chosen predicted quantity, with bootstrap confidence intervals obtained by recomputing the effect on every bootstrap fit.

Usage

```
marginal_effects(
  object,
  variables = NULL,
  type = c("harmonic", "explog", "states", "quantile"),
  quantile = NULL,
  at = list(),
  method = c("derivative", "difference"),
  eps = 1e-04,
  delta = 1,
  conf_level = 0.95,
  newdata = NULL,
  n_max = 500,
  exclude_degenerate = TRUE,
  multicore = NULL,
  ncores = NULL,
  clamp_alpha_pl = FALSE
)
```

Arguments

object	A fitted evzinb / evinb model with bootstraps.
variables	Covariates to compute effects for (default: all raw covariates). An unrecognised name errors, listing the available covariates. A variable that only ever enters the model's formulas wrapped in <code>factor()/as.factor()/cut()</code> is treated as categorical (level-vs-reference contrasts over its observed unique values) even though it is stored as numeric data; one wrapped only in <code>poly()/ns()/bs()</code> keeps the numeric derivative/ difference path but has its perturbation clamped to the covariate's observed range, so the perturbed value is never fed to the spline basis outside the range it was built from. A variable used both ways (e.g. bare in one formula component and <code>factor()</code> -wrapped in another) errors, since a single perturbed value cannot represent both.
type	"harmonic", "explog", "states" or "quantile".
quantile	Quantile for type = "quantile".
at	Named list of values at which to hold covariates before differencing.
method	For numeric covariates, "derivative" (central finite difference of step eps) or "difference" (average change from a delta-unit increase in the covariate). Defaults to "derivative", except for type = "quantile" where it defaults to "difference" (a one-unit change is what a quantile effect means in practice). Factor covariates always use level-vs-reference differences.
eps	Step size for the central finite difference (method = "derivative").
delta	Covariate increase for method = "difference" (default 1 unit; pass e.g. <code>sd(x)</code> for a one-SD change).
conf_level	Confidence level for the bootstrap intervals.
newdata	Data to average over (default: the estimation data).
n_max	For type = "quantile" only: cap the number of rows the effect is averaged over (the per-observation quantile machinery is slow). The cap is applied automatically when <code>nrow(newdata) * n_bootstraps > 2e5</code> , or whenever <code>n_max</code> is passed explicitly; <code>n_max = Inf</code> disables it. The subsample is reproducible from the model's bootstrap seed.
exclude_degenerate	Drop bootstrap replicates flagged degenerate (default TRUE); see the <code>alpha_floor</code> argument of evinf_control .
multicore	Bootstrap parallelisation shortcut. The default (NULL) respects whatever future plan is currently set (see the Parallel processing section). TRUE sets a temporary multisession plan for the duration of the call; FALSE forces sequential execution. The previous plan is always restored on exit.
ncores	Number of workers when multicore = TRUE. Default (NULL) is one less than the number of available cores. Ignored when multicore is NULL or FALSE.
clamp_alpha_pl	(round11 A4) type = "explog" only; forwarded to <code>predict()</code> for both the point estimate and every bootstrap replicate. See <code>?predict.evzinb</code> .

Details

The confidence interval is a percentile interval: `conf.low` / `conf.high` are the empirical quantiles of the effect recomputed on each bootstrap fit. `std.error` is the standard deviation of those recomputed effects, except for `type = "quantile"` where the bootstrap distribution is heavy-tailed and `std.error` is instead a robust scale read off the interval, $(\text{conf.high} - \text{conf.low}) / (2 * \text{qnorm}(1 - (1 - \text{conf_level})/2))$. It is reported for reference and is not used to build the interval.

Harmonic-mean effects and bootstrap intervals. The harmonic-mean prediction is $C(1 + \alpha)/\alpha$, which blows up on bootstrap fits whose Pareto shape α is very small. The point estimate for `type = "harmonic"` is well behaved, but the percentile interval can be extremely wide because a handful of bootstrap draws are enormous. For inference about how a covariate shifts the outcome, prefer `type = "quantile"` (effect on a predicted quantile) or `type = "states"` (effect on the state probabilities), whose bootstrap distributions are bounded.

Quantile effects. The mixture quantile is a step function of a count, so its derivative is either 0 or huge; `type = "quantile"` therefore defaults to `method = "difference"` (the average change in the predicted quantile from a delta-unit increase in the covariate). The derivative is still available with `method = "derivative"` but its bootstrap distribution can be very heavy-tailed. The quantile effect is evaluated per observation by bisection on the mixture CDF, which is still the slowest prediction type, so it is averaged over at most `n_max` rows (see that argument).

Value

A tibble with columns `variable`, `contrast`, `type`, `estimate`, `std.error`, `conf.low`, `conf.high`.

Parallel processing

Bootstrap fits (and the per-bootstrap work in [add_bootstraps](#), [lr_test](#), [compare_models](#), [predict.evzinb](#) and [marginal_effects](#)) are dispatched with **furrr** on top of a **future** plan. Set the plan once for your session and leave multicore at its default:

```
future::plan(future::multisession, workers = 8)
progressr::handlers(global = TRUE) # opt in to progress bars
model <- evzinb(y ~ x1 + x2 + x3, data = genevzinb2, n_bootstraps = 1000)
```

Any future backend works (multisession, cluster, callr, a HPC batchtools plan, ...). As a convenience `multicore = TRUE` sets a temporary multisession plan for the single call and restores the previous plan on exit. Large models may need `options(future.globals.maxSize = <bytes>)` to raise the default limit on the data shipped to each worker.

Reproducibility

The bootstrap **resample indices** (`object$bootstraps[[i]]$boot_id`) are fully determined by `boot_seed` and are independent of the **future** backend, the number of workers, and chunking — a fixed seed always draws the same resamples.

The **fitted coefficients** are reproducible only to within floating-point noise. BLAS operations are not bit-reproducible across processes, so a sequential run and a multisession run of the same seed can differ in the last few digits; and because the EM log-likelihood can have close local optima, an occasional replicate converges to a different one under a different backend. For replication material, record the future plan and `utils::sessionInfo()` alongside `boot_seed`, and produce the canonical set of estimates under a single fixed plan.

Examples

```
data(genevzinb2)
model <- evzinb(y ~ x1 + x2 + x3, data = genevzinb2, n_bootstraps = 10)
marginal_effects(model, variables = "x1")
```

oob_evaluation

Out of bag predictive performance of EVZINB and EVINB models

Description

Out of bag predictive performance of EVZINB and EVINB models

Usage

```
oob_evaluation(
  object,
  predict_type = c("harmonic", "explog"),
  metric = c("rmsle", "rmse", "mse", "mae"),
  exclude_degenerate = TRUE
)
```

Arguments

object	A fitted evzinb / evinb model with bootstraps, or an evzinbcomp object from compare_models() .
predict_type	What type of prediction should be made? Harmonic mean, or exp(log(prediction))?
metric	What metric should be used for the out of bag evaluation? Default options include rmsle, rmse, mse, and mae. Can also take a user supplied function of the form function(y_pred, y_true) which returns a single value
exclude_degenerate	For a single evinf model (or its \$model slot inside an evzinbcomp), return NA for bootstrap replicates flagged degenerate (default TRUE) instead of their out-of-bag error, so positions still line up with the compared models' replicates; see the alpha_floor argument of evinf_control .

Value

For a single model, a vector of length n_bootstraps. For an evzinbcomp object, a tibble with one column per compared model (evinf, nb, zinb, ...) and one row per bootstrap. A replicate that is NA in any column (a degenerate evinf replicate, or a compared model's fit that errored) is NA in *every* column, so a column-wise na.rm = TRUE summary (e.g. summarize(oob, across(everything(), median, na.rm = TRUE))) compares the same set of replicates across models rather than silently different ones; the row mask itself is available as attr(out, "excluded") (round8 0.4, review §5).

Examples

```
data(genevzinb2)
model <- evzinb(y~x1+x2+x3,data=genevzinb2, n_bootstraps = 5)
oob_evaluation(model)
```

plot.evzinb

Plots for evzinb / evinb models

Description

Plots for evzinb / evinb models

Usage

```
## S3 method for class 'evzinb'
plot(
  x,
  type = c("states", "prediction", "coefficients", "ppc", "ppc_quantiles", "trace"),
  variable = NULL,
  quantiles = c(0.5, 0.95),
  ...
)

## S3 method for class 'evinb'
plot(
  x,
  type = c("states", "prediction", "coefficients", "ppc", "ppc_quantiles", "trace"),
  variable = NULL,
  quantiles = c(0.5, 0.95),
  ...
)
```

Arguments

x	A fitted evzinb / evinb model.
type	"states" - prior state probabilities over variable, faceted by state; "prediction" - harmonic-mean prediction with a bootstrap ribbon plus the requested quantiles over variable (log1p y axis); "coefficients" - bootstrap densities of each coefficient, faceted by component and term, with the point estimate marked; "ppc" - posterior predictive check: a binned observed-vs-expected frequency panel; "ppc_quantiles" - posterior predictive check on the tails: the observed sample quantiles for probabilities c(0.5, 0.75, 0.9, 0.95, 0.99, 0.999) against the median simulated quantile (with a 5-95% band across simulations) on log1p axes, with a 45-degree reference line; "trace" (round10 G.2, audit §5.10) - the log-likelihood (\$loglik_trace) and C_{EV} (\$c_trace) traces, in two stacked panels with the warm-up/convergence boundary marked (dashed line). With

evinf_control(n_starts > 1), every perturbed start's trace is overlaid in grey behind the winning start's, in black (traces are only stored per-start when n_starts > 1, to keep a single-start object the same size as before this type existed).

variable Covariate to vary (required for "states" and "prediction").

quantiles Quantiles to draw for type = "prediction".

... Passed to [predict_grid](#).

Value

A ggplot object.

Examples

```
data(genevzinb2)
model <- evzinb(y ~ x1 + x2 + x3, data = genevzinb2, n_bootstraps = 5)
plot(model, type = "coefficients")
plot(model, type = "prediction", variable = "x1")
```

```
data(hks)
hks_mod <- evzinb(osvAll ~ troopLag + lnpop + brv_AllLag_log,
                  data = hks, n_bootstraps = 2, multicore = FALSE)
plot(hks_mod, type = "prediction", variable = "troopLag")
```

plot_c_profile	<i>Plot the log-likelihood profile over the candidate values of C_EV</i>
----------------	--

Description

Plot the log-likelihood profile over the candidate values of C_EV

Usage

```
plot_c_profile(object)
```

Arguments

object A fitted evzinb / evinb object.

Value

A ggplot object.

Examples

```
data(genevzinb2)
model <- evzinb(y ~ x1 + x2 + x3, data = genevzinb2, n_bootstraps = 5)
plot_c_profile(model)
```

predict.evinb	<i>Predictions from evinb object</i>
---------------	--------------------------------------

Description

Predictions from evinb object

Usage

```
## S3 method for class 'evinb'
predict(
  object,
  newdata = NULL,
  type = c("harmonic", "explog", "counts", "pareto_alpha", "evinf", "count_state",
    "states", "all", "quantile", "distribution", "exceedance", "draws"),
  pred = c("original", "bootstrap_median", "bootstrap_mean"),
  quantile = NULL,
  confint = FALSE,
  conf_level = 0.95,
  multicore = NULL,
  ncores = NULL,
  return_bootstraps = FALSE,
  exclude_degenerate = TRUE,
  support = NULL,
  max_support = 1e+05,
  format = c("long", "matrix"),
  threshold = NULL,
  n_draws = 1000,
  parameter_uncertainty = FALSE,
  seed = NULL,
  keep = NULL,
  clamp_alpha_pl = FALSE,
  ...
)
```

Arguments

object	An evinb object for which to produce predicted values
newdata	Optional new data (tibble) to produce predicted values from

type	Character string, 'harmonic' for the harmonic mean and 'explog' for exponentiated expected log, 'counts' for predicted count of the negative binomial component, 'pareto_alpha' for the predicted pareto alpha value, 'states' for the predicted component states (prior), 'count_state' for predicted probability of the count state, 'evinf' for predicted probability of the pareto state, 'all' for all predicted values, 'quantile' for quantile prediction (scalar or vector, see 'quantile'), 'distribution' for the full predictive distribution, 'exceedance' for exceedance probabilities (see 'threshold'), and 'draws' for predictive draws (see 'n_draws').
pred	Type of prediction to be used, defaults to the original prediction from the fitted model, with alternatives being the bootstrapped median or mean. Note that bootstrap mean may yield infinite values, especially when doing quantile prediction
quantile	Quantile(s) for which to produce quantile prediction. A single value keeps the existing scalar behavior exactly (a vector, e.g. 'c(.5, .9, .99)' (round10 H.3), returns a tibble with one 'qXX' column per probability ('qXX_lo'/'qXX_hi' too, with 'confint = TRUE').
confint	Should confidence intervals be made for the predictions? Note: only available for vector type predictions and not 'states', 'all' or 'distribution'.
conf_level	What confidence level should be used for confidence intervals
multicore	Bootstrap parallelisation shortcut. The default (NULL) respects whatever future plan is currently set (see the Parallel processing section). TRUE sets a temporary multisession plan for the duration of the call; FALSE forces sequential execution. The previous plan is always restored on exit.
ncores	Number of workers when multicore = TRUE. Default (NULL) is one less than the number of available cores. Ignored when multicore is NULL or FALSE.
return_bootstraps	Should the bootstrapped predictions be returned as well? Useful for further custom analyses of the bootstrapped predictions.
exclude_degenerate	Drop bootstrap replicates flagged degenerate (default TRUE); see the alpha_floor argument of evinf_control().
support	(round10 H.2) 'type = 'distribution'' only: shared support (vector of 'y' values) for every row. 'NULL' (default) uses '0:K', 'K' the ceiling of the 0.999 mixture quantile of the heaviest-tailed row, capped at 'max_support'.
max_support	(round10 H.2) Upper bound on the default 'support's 'K' (default 1e5); ignored when 'support' is given explicitly.
format	(round10 H.2) 'type = 'distribution'' only: 'long' (default; a tibble with 'row', 'y', 'prob') or 'matrix' (the raw n x K matrix; 'keep' is ignored).
threshold	(round10 H.4) 'type = 'exceedance'' only: a vector of thresholds; result columns are 'p_ge_<threshold>', 'P(Y >= threshold)'.
n_draws	(round10 H.5) 'type = 'draws'' only: number of predictive draws per row.
parameter_uncertainty	(round10 H.5) 'type = 'draws'' only: if 'TRUE', each draw uses a randomly chosen usable bootstrap replicate's parameters instead of the full-sample estimate.

seed	(round10 H.5) 'type = 'draws'' only: optional seed; the caller's '.Random.seed' is left untouched either way (same convention as 'simulate()').
keep	(round10 H.6) Optional character vector of 'newdata' column names to carry into the result (a join key for panel data); supported for 'type' in 'quantile' (vector), 'distribution' ('format = 'long'), 'exceedance' and 'draws'.
clamp_alpha_pl	(round11 A4) 'type = 'explog'' (and 'all') only: 'FALSE' (default) uses the fitted 'alpha_pl' as-is and warns below 0.1; 'TRUE' clamps at 0.1; a positive number clamps there instead. Exempt from the global 'alpha_pl_floor' (which stays in force for 'harmonic'/'quantile'). The clamp used is recorded as a "clamp_alpha_pl" attribute on the result.
...	Not used; any name here is an unknown argument and errors (round11 A5), naming the valid arguments for the requested 'type'. A known argument supplied for a 'type' it does nothing for (e.g. 'threshold' with 'type = "harmonic"') warns instead, since it has a named formal below and never reaches here.

Details

The likelihood, CDF, quantile prediction, residuals and `simulate()` all use the discretised Pareto distribution (the integer-valued distribution the model is actually fit on). The 'harmonic' and 'explog' point predictions instead use the harmonic and geometric means of the *continuous* Pareto distribution as a closed-form approximation to the corresponding discretised-Pareto moments; this keeps existing point predictions unchanged but means they are not computed from exactly the same distribution as the rest of the model.

`type = 'explog'` is $C \cdot \exp(1/\alpha_{PL})$, which grows explosively as the fitted Pareto shape α_{PL} approaches 0. Unlike 'harmonic' and 'quantile', this path is exempt from the global `evinf_control(alpha_pl_floor = )`; `clamp_alpha_pl` is its own, separate knob (round11 A4). With `clamp_alpha_pl = FALSE` (the default), the fitted α_{PL} is used as-is and a warning fires whenever any value is below 0.1, since the prediction is effectively undefined there ($e^{10} \approx 2.2 \times 10^4$, and it may be Inf); check `glance()$min_alpha_pl`, and either pass `clamp_alpha_pl = TRUE` (clamps at 0.1) or a positive number (clamps there instead), or use `type = 'harmonic'`. The clamp actually applied (FALSE, or the numeric floor) is recorded as a "clamp_alpha_pl" attribute on the result.

Value

A vector of predicted values for type 'harmonic', 'explog', 'counts', 'pareto_alpha', 'evinf', 'count_state', and 'quantile' (scalar), or a tibble for type 'states', 'all', 'quantile' (vector), 'distribution' ('format = 'long'), 'exceedance', 'draws', or if `confint=T`; a matrix for 'distribution' with 'format = 'matrix'.

Parallel processing

Bootstrap fits (and the per-bootstrap work in `add_bootstraps`, `lr_test`, `compare_models`, `predict.evzinb` and `marginal_effects`) are dispatched with **furrr** on top of a **future** plan. Set the plan once for your session and leave multicore at its default:

```
future::plan(future::multisession, workers = 8)
progressr::handlers(global = TRUE) # opt in to progress bars
model <- evzinb(y ~ x1 + x2 + x3, data = genevzinb2, n_bootstraps = 1000)
```

Any future backend works (multisession, cluster, callr, a HPC batchtools plan, ...). As a convenience `multicore = TRUE` sets a temporary multisession plan for the single call and restores the previous plan on exit. Large models may need `options(future.globals.maxSize = <bytes>)` to raise the default limit on the data shipped to each worker.

Reproducibility

The bootstrap **resample indices** (`object$bootstraps[[i]]$boot_id`) are fully determined by `boot_seed` and are independent of the **future** backend, the number of workers, and chunking — a fixed seed always draws the same resamples.

The **fitted coefficients** are reproducible only to within floating-point noise. BLAS operations are not bit-reproducible across processes, so a sequential run and a multisession run of the same seed can differ in the last few digits; and because the EM log-likelihood can have close local optima, an occasional replicate converges to a different one under a different backend. For replication material, record the future plan and `utils::sessionInfo()` alongside `boot_seed`, and produce the canonical set of estimates under a single fixed plan.

Examples

```
data(genevzinb2)
model <- evinb(y~x1+x2+x3,data=genevzinb2, n_bootstraps = 5)
predict(model)
predict(model, type='all', quantile = 0.9) # all available predicted values
predict(model, type='quantile', quantile = c(.5, .9, .99)) # several quantiles (H.3)
predict(model, type='distribution') # the full predictive distribution (H.2)
predict(model, type='exceedance', threshold = c(10, 100)) # exceedance probs (H.4)
predict(model, type='draws', n_draws = 100) # predictive draws (H.5)
```

predict.evzinb	<i>Predictions from evzinb object</i>
----------------	---------------------------------------

Description

Predictions from evzinb object

Usage

```
## S3 method for class 'evzinb'
predict(
  object,
  newdata = NULL,
  type = c("harmonic", "explog", "counts", "pareto_alpha", "zi", "evinf", "count_state",
    "states", "all", "quantile", "distribution", "exceedance", "draws"),
  pred = c("original", "bootstrap_median", "bootstrap_mean"),
  quantile = NULL,
  confint = FALSE,
  conf_level = 0.95,
```

```

multicore = NULL,
ncores = NULL,
return_bootstraps = FALSE,
exclude_degenerate = TRUE,
support = NULL,
max_support = 1e+05,
format = c("long", "matrix"),
threshold = NULL,
n_draws = 1000,
parameter_uncertainty = FALSE,
seed = NULL,
keep = NULL,
clamp_alpha_pl = FALSE,
...
)

```

Arguments

object	An evzinb object for which to produce predicted values
newdata	Optional new data (tibble) to produce predicted values from
type	Character string, 'harmonic' for the harmonic mean and 'explog' for exponentiated expected log, 'counts' for predicted count of the negative binomial component, 'pareto_alpha' for the predicted pareto alpha value, 'states' for the predicted component states (prior), 'count_state' for predicted probability of the count state, 'evinf' for predicted probability of the pareto state, 'zi' for the predicted probability of the zero state, 'all' for all predicted values, 'quantile' for quantile prediction (scalar or vector, see 'quantile'), 'distribution' for the full predictive distribution, 'exceedance' for exceedance probabilities (see 'threshold'), and 'draws' for predictive draws (see 'n_draws').
pred	Type of prediction to be used, defaults to the original prediction from the fitted model, with alternatives being the bootstrapped median or mean. Note that bootstrap mean may yield infinite values, especially when doing quantile prediction
quantile	Quantile(s) for which to produce quantile prediction. A single value keeps the existing scalar behavior exactly (a vector, e.g. 'c(.5, .9, .99)' (round10 H.3), returns a tibble with one 'qXX' column per probability ('qXX_lo'/'qXX_hi' too, with 'confint = TRUE').
confint	Should confidence intervals be made for the predictions? Note: only available for vector type predictions and not 'states', 'all' or 'distribution'.
conf_level	What confidence level should be used for confidence intervals
multicore	Bootstrap parallelisation shortcut. The default (NULL) respects whatever future plan is currently set (see the Parallel processing section). TRUE sets a temporary multisession plan for the duration of the call; FALSE forces sequential execution. The previous plan is always restored on exit.
ncores	Number of workers when multicore = TRUE. Default (NULL) is one less than the number of available cores. Ignored when multicore is NULL or FALSE.

return_bootstraps	Should the bootstrapped predictions be returned as well? Useful for further custom analyses of the bootstrapped predictions.
exclude_degenerate	Drop bootstrap replicates flagged degenerate (default TRUE); see the <code>alpha_floor</code> argument of <code>evinf_control()</code> .
support	(round10 H.2) <code>'type = 'distribution''</code> only: shared support (vector of <code>'y'</code> values) for every row. <code>'NULL'</code> (default) uses <code>'0:K'</code> , <code>'K'</code> the ceiling of the 0.999 mixture quantile of the heaviest-tailed row, capped at <code>'max_support'</code> .
max_support	(round10 H.2) Upper bound on the default <code>'support'</code> 's <code>'K'</code> (default 1e5); ignored when <code>'support'</code> is given explicitly.
format	(round10 H.2) <code>'type = 'distribution''</code> only: <code>'long'</code> (default; a tibble with <code>'row'</code> , <code>'y'</code> , <code>'prob'</code>) or <code>'matrix'</code> (the raw $n \times K$ matrix; <code>'keep'</code> is ignored).
threshold	(round10 H.4) <code>'type = 'exceedance''</code> only: a vector of thresholds; result columns are <code>'p_ge_<threshold>'</code> , <code>'P(Y >= threshold)'</code> .
n_draws	(round10 H.5) <code>'type = 'draws''</code> only: number of predictive draws per row.
parameter_uncertainty	(round10 H.5) <code>'type = 'draws''</code> only: if <code>'TRUE'</code> , each draw uses a randomly chosen usable bootstrap replicate's parameters instead of the full-sample estimate.
seed	(round10 H.5) <code>'type = 'draws''</code> only: optional seed; the caller's <code>'.Random.seed'</code> is left untouched either way (same convention as <code>'simulate()'</code>).
keep	(round10 H.6) Optional character vector of <code>'newdata'</code> column names to carry into the result (a join key for panel data); supported for <code>'type'</code> in <code>'quantile'</code> (vector), <code>'distribution'</code> (<code>'format = 'long''</code>), <code>'exceedance'</code> and <code>'draws'</code> .
clamp_alpha_pl	(round11 A4) <code>'type = 'explog''</code> (and <code>'all''</code>) only: <code>'FALSE'</code> (default) uses the fitted <code>'alpha_pl'</code> as-is and warns below 0.1; <code>'TRUE'</code> clamps at 0.1; a positive number clamps there instead. Exempt from the global <code>'alpha_pl_floor'</code> (which stays in force for <code>'harmonic'/'quantile'</code>). The clamp used is recorded as a <code>"clamp_alpha_pl"</code> attribute on the result.
...	Not used; any name here is an unknown argument and errors (round11 A5), naming the valid arguments for the requested <code>'type'</code> . A known argument supplied for a <code>'type'</code> it does nothing for (e.g. <code>'threshold'</code> with <code>'type = "harmonic"'</code>) warns instead, since it has a named formal below and never reaches here.

Details

The likelihood, CDF, quantile prediction, residuals and `simulate()` all use the discretised Pareto distribution (the integer-valued distribution the model is actually fit on). The `'harmonic'` and `'explog'` point predictions instead use the harmonic and geometric means of the **continuous** Pareto distribution as a closed-form approximation to the corresponding discretised-Pareto moments; this keeps existing point predictions unchanged but means they are not computed from exactly the same distribution as the rest of the model.

`type = 'explog'` is $C \cdot \exp(1/\alpha_{PL})$, which grows explosively as the fitted Pareto shape α_{PL} approaches 0. Unlike `'harmonic'` and `'quantile'`, this path is exempt from the global `evinf_control(alpha_pl_floor`

=); `clamp_alpha_pl` is its own, separate knob (round11 A4). With `clamp_alpha_pl = FALSE` (the default), the fitted α_{PL} is used as-is and a warning fires whenever any value is below 0.1, since the prediction is effectively undefined there ($e^{10} \approx 2.2 \times 10^4$, and it may be Inf); check `glance()$min_alpha_pl`, and either pass `clamp_alpha_pl = TRUE` (clamps at 0.1) or a positive number (clamps there instead), or use `type = 'harmonic'`. The clamp actually applied (FALSE, or the numeric floor) is recorded as a "`clamp_alpha_pl`" attribute on the result.

Value

A vector of predicted values for type `'harmonic'`, `'explog'`, `'counts'`, `'pareto_alpha'`, `'zi'`, `'evinf'`, `'count_state'`, and `'quantile'` (scalar), or a tibble for type `'states'`, `'all'`, `'quantile'` (vector), `'distribution'` (`'format = 'long'`), `'exceedance'`, `'draws'`, or if `confint=T`; a matrix for `'distribution'` with `'format = 'matrix'`.

Parallel processing

Bootstrap fits (and the per-bootstrap work in [add_bootstraps](#), [lr_test](#), [compare_models](#), [predict.evzinb](#) and [marginal_effects](#)) are dispatched with **furrr** on top of a **future** plan. Set the plan once for your session and leave multicore at its default:

```
future::plan(future::multisession, workers = 8)
progressr::handlers(global = TRUE) # opt in to progress bars
model <- evzinb(y ~ x1 + x2 + x3, data = genevzinb2, n_bootstraps = 1000)
```

Any future backend works (multisession, cluster, callr, a HPC batchtools plan, ...). As a convenience `multicore = TRUE` sets a temporary multisession plan for the single call and restores the previous plan on exit. Large models may need `options(future.globals.maxSize = <bytes>)` to raise the default limit on the data shipped to each worker.

Reproducibility

The bootstrap **resample indices** (`object$bootstraps[[i]]$boot_id`) are fully determined by `boot_seed` and are independent of the **future** backend, the number of workers, and chunking — a fixed seed always draws the same resamples.

The **fitted coefficients** are reproducible only to within floating-point noise. BLAS operations are not bit-reproducible across processes, so a sequential run and a multisession run of the same seed can differ in the last few digits; and because the EM log-likelihood can have close local optima, an occasional replicate converges to a different one under a different backend. For replication material, record the future plan and `utils::sessionInfo()` alongside `boot_seed`, and produce the canonical set of estimates under a single fixed plan.

Examples

```
data(genevzinb2)
model <- evzinb(y~x1+x2+x3,data=genevzinb2, n_bootstraps = 5)
predict(model)
predict(model, type='all', quantile = 0.9) # all available predicted values
predict(model, type='quantile', quantile = c(.5, .9, .99)) # several quantiles (H.3)
predict(model, type='distribution') # the full predictive distribution (H.2)
predict(model, type='exceedance', threshold = c(10, 100)) # exceedance probs (H.4)
```

```
predict(model, type='draws', n_draws = 100) # predictive draws (H.5)
```

predict.nbboot	<i>Prediction for nbboot</i>
----------------	------------------------------

Description

Prediction for nbboot

Usage

```
## S3 method for class 'nbboot'
predict(
  object,
  newdata = NULL,
  type = c("predicted", "all", "quantile"),
  pred = c("original", "bootstrap_median", "bootstrap_mean"),
  quantile = NULL,
  confint = FALSE,
  conf_level = 0.9,
  ...
)
```

Arguments

object	a fitted nbboot object
newdata	Data to make predictions on
type	What prediction should be computed?
pred	Prediction type, 'original', 'bootstrap_median', or 'bootstrap_mean'
quantile	Quantile for quantile prediction
confint	Should confidence intervals be created?
conf_level	Confidence level when predicting with CIs
...	Not used

Value

Predictions from nbboot

predict.poissonboot	<i>Prediction for poissonboot</i>
---------------------	-----------------------------------

Description

Prediction for poissonboot

Usage

```
## S3 method for class 'poissonboot'
predict(
  object,
  newdata = NULL,
  type = c("predicted", "all", "quantile"),
  pred = c("original", "bootstrap_median", "bootstrap_mean"),
  quantile = NULL,
  confint = FALSE,
  conf_level = 0.9,
  ...
)
```

Arguments

object	a fitted poissonboot object
newdata	Data to make predictions on
type	What prediction should be computed?
pred	Prediction type, 'original', 'bootstrap_median', or 'bootstrap_mean'
quantile	Quantile for quantile prediction
confint	Should confidence intervals be created?
conf_level	Confidence level when predicting with CIs
...	Not used

Value

Predictions from poissonboot

predict.zinbboot	<i>Prediction for zinbboot</i>
------------------	--------------------------------

Description

Prediction for zinbboot

Usage

```
## S3 method for class 'zinbboot'
predict(
  object,
  newdata = NULL,
  type = c("predicted", "counts", "zi", "count_state", "states", "all", "quantile"),
  pred = c("original", "bootstrap_median", "bootstrap_mean"),
  quantile = NULL,
  confint = FALSE,
  conf_level = 0.9,
  ...
)
```

Arguments

object	a fitted zinbboot object
newdata	Data to make predictions on
type	What prediction should be computed? One of "predicted", "counts", "zi", "count_state", "states", "all" or "quantile". (A zero-inflated negative binomial has no extreme-value state, so "evinf" is not accepted.)
pred	Prediction type, 'original', 'bootstrap_median', or 'bootstrap_mean'
quantile	Quantile for quantile prediction
confint	Should confidence intervals be created?
conf_level	Confidence level when predicting with CIs
...	Not used

Value

Predictions from zinbboot

predict.zipboot	<i>Prediction for zipboot</i>
-----------------	-------------------------------

Description

Prediction for zipboot

Usage

```
## S3 method for class 'zipboot'
predict(
  object,
  newdata = NULL,
  type = c("predicted", "counts", "zi", "count_state", "states", "all", "quantile"),
  pred = c("original", "bootstrap_median", "bootstrap_mean"),
  quantile = NULL,
  confint = FALSE,
  conf_level = 0.9,
  ...
)
```

Arguments

object	a fitted zipboot object
newdata	Data to make predictions on
type	What prediction should be computed? One of "predicted", "counts", "zi", "count_state", "states", "all" or "quantile". (A zero-inflated Poisson has no extreme-value state, so "evinf" is not accepted.)
pred	Prediction type, 'original', 'bootstrap_median', or 'bootstrap_mean'
quantile	Quantile for quantile prediction
confint	Should confidence intervals be created?
conf_level	Confidence level when predicting with CIs
...	Not used

Value

Predictions from zipboot

predict_grid

*Build a prediction grid and evaluate a model over it***Description**

Varies one covariate over a range (or set of values) while holding the others at representative values, and returns the model predictions in long form - the raw material for an effect plot.

Usage

```
predict_grid(
  object,
  variable,
  values = NULL,
  n = 50,
  at = list(),
  fixed = c("mean", "median"),
  type = c("states", "harmonic", "explog", "quantile", "counts", "pareto_alpha"),
  quantile = NULL,
  confint = FALSE,
  conf_level = 0.9,
  clamp_alpha_pl = FALSE
)
```

Arguments

object	A fitted evzinb / evinb model.
variable	Name of the covariate to vary.
values	Optional explicit values for variable; otherwise an evenly spaced sequence (numeric) or all levels (factor).
n	Number of grid points for a numeric variable.
at	Named list of values at which to hold specific other covariates.
fixed	How to hold the remaining numeric covariates: "mean" or "median".
type	Prediction type: "states", "harmonic", "explog", "quantile", "counts" or "pareto_alpha".
quantile	Quantile for type = "quantile".
confint	Add bootstrap confidence intervals (not for "states").
conf_level	Confidence level.
clamp_alpha_pl	(round11 A4) type = "explog" only; forwarded to predict(). See ?predict.evzinb.

Value

A tibble with columns variable, value, type, estimate, conf.low, conf.high (the last two NA when confint = FALSE).

Examples

```
data(genevzinb2)
model <- evzinb(y ~ x1 + x2 + x3, data = genevzinb2, n_bootstraps = 5)
predict_grid(model, "x1", type = "harmonic")
```

```
data(hks)
hks_mod <- evzinb(osvAll ~ troopLag + lnpop + brv_AllLag_log,
                  data = hks, n_bootstraps = 2, multicore = FALSE)
predict_grid(hks_mod, "troopLag", type = "states")
```

print.evinb

EVINB print function

Description

EVINB print function

Usage

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

Arguments

x	A fitted evinb model
...	Not used

Value

x, invisibly.

Examples

```
data(genevzinb2)
model <- evinb(y~x1+x2+x3,data=genevzinb2, n_bootstraps = 5)
print(model)
```

print.evzinb	<i>EVZINB print function</i>
--------------	------------------------------

Description

EVZINB print function

Usage

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

Arguments

x	A fitted evzinb model
...	Not used

Value

x, invisibly.

Examples

```
data(genevzinb2)
model <- evzinb(y~x1+x2+x3,data=genevzinb2, n_bootstraps = 5)
print(model)
```

print.evzinbcomp	<i>Print method for compare_models() output</i>
------------------	---

Description

Shows the compared models, a compact [glance\(\)](#) table (one row per slot, key fit statistics only), and the paired bootstrap comparison from [compare_fit\(\)](#) (round10 I.4, audit §5.9) – previously only the compared-model names and bootstrap count.

Usage

```
## S3 method for class 'evzinbcomp'
print(x, metrics = c("aic", "bic", "rmse", "rmsle"), ...)
```

Arguments

x	An evzinbcomp object returned by compare_models .
metrics	Metrics for the compare_fit() table; see there.
...	Not used

Value

x, invisibly.

Examples

```
data(genevzinb2)
model <- evzinb(y~x1+x2+x3,data=genevzinb2, n_bootstraps = 5)
print(compare_models(model))
```

```
print.summary.evzinb  Print methods for evzinb / evinb summaries
```

Description

Print methods for evzinb / evinb summaries

Usage

```
## S3 method for class 'summary.evzinb'
print(
  x,
  digits = max(3L, getOption("digits") - 3L),
  signif.stars = getOption("show.signif.stars"),
  ...
)

## S3 method for class 'summary.evinb'
print(
  x,
  digits = max(3L, getOption("digits") - 3L),
  signif.stars = getOption("show.signif.stars"),
  ...
)
```

Arguments

x	A summary.evzinb or summary.evinb object.
digits	Number of significant digits for the coefficient tables.
signif.stars	Logical; show significance stars.
...	Not used.

Value

x, invisibly.

revinb_fit	<i>Random draws from a fitted evinb model</i>
------------	---

Description

Superseded: kept for backwards compatibility, with no plans for removal, but new code should use [simulate\(\)](#), which returns a tidy data frame (and a reproducible "seed" attribute) instead of a bare vector/list.

Usage

```
revinb_fit(object, newdata = NULL, n_draws = 1)
```

Arguments

object	A fitted EVINB object
newdata	Optional newdata
n_draws	Number of random draws to make

Value

A vector of randomly drawn values from the fitted evinb if `n_draws == 1`, or a list of length `n_draws` with random drawn values if `n_draws > 1`

See Also

[simulate.evinb](#)

Examples

```
data(genevzinb2)
model <- evinb(y~x1+x2+x3, data=genevzinb2, n_bootstraps = 5)
revinb_fit(model)
```

revzinb_fit	<i>Random draws from a fitted evzinb model</i>
-------------	--

Description

Superseded: kept for backwards compatibility, with no plans for removal, but new code should use [simulate\(\)](#), which returns a tidy data frame (and a reproducible "seed" attribute) instead of a bare vector/list.

Usage

```
revzinb_fit(object, newdata = NULL, n_draws = 1)
```

Arguments

object	A fitted EVZINB object
newdata	Optional newdata
n_draws	Number of random draws to make

Value

A vector of randomly drawn values from the fitted evzinb if `n_draws == 1`, or a list of length `n_draws` with random drawn values if `n_draws > 1`

See Also

```
simulate.evzinb
```

Examples

```
data(genevzinb2)
model <- evzinb(y~x1+x2+x3, data=genevzinb2, n_bootstraps = 5)
revzinb_fit(model)
```

state_table	<i>Cross-tabulate prior vs posterior state classification</i>
-------------	---

Description

Cross-tabulate prior vs posterior state classification

Usage

```
state_table(object)
```

Arguments

object	A fitted evzinb / evinb model.
--------	--------------------------------

Value

An object of class `evinf_state_table`: the `map_prior` x `map_posterior` contingency table with row percentages.

Examples

```
data(genevzinb2)
model <- evzinb(y ~ x1 + x2 + x3, data = genevzinb2, n_bootstraps = 5)
state_table(model)
```

summary.evinb

*EVINB summary function***Description**

EVINB summary function

Usage

```
## S3 method for class 'evinb'
summary(
  object,
  coef = c("original", "bootstrapped_mean", "bootstrapped_median"),
  standard_error = TRUE,
  p_value = c("bootstrapped", "approx", "both", "none"),
  bootstrapped_props = c("none", "mean", "median"),
  approx_t_value = TRUE,
  symmetric_bootstrap_p = TRUE,
  exclude_degenerate = TRUE,
  ...
)
```

Arguments

object	an EVINB object
coef	Type of coefficients. Original are the coefficient estimates from the non-bootstrapped version of the model. 'bootstrapped_mean' are the mean coefficients across bootstraps, and 'bootstrapped_median' are the median coefficients across bootstraps
standard_error	Should standard errors be computed?
p_value	What type of p_values should be computed? 'bootstrapped' are bootstrapped p_values through confidence interval inversion. 'approx' are p-values based on the t-value produced by dividing the coefficient with the standard error. 'both' returns both.
bootstrapped_props	Type of bootstrapped proportions of component proportions to be returned
approx_t_value	Should approximate t-values be returned
symmetric_bootstrap_p	Should bootstrap p-values be computed as symmetric (leaving alpha/2 percent in each tail)? FALSE gives non-symmetric, but narrower, intervals. TRUE corresponds most closely to conventional p-values.

```

exclude_degenerate      Drop bootstrap replicates flagged degenerate (default TRUE); see the alpha_floor
                        argument of evinf_control().
...                     Additional arguments passed to the summary function

```

Details

When object was fitted with `bootstrap = FALSE`, the bootstrap-based quantities (standard errors, p-values, approximate t-values and bootstrapped proportions) are unavailable; the summary then reports the point estimates only, `n_failed_bootstraps` is NA, and a message is emitted. Requesting bootstrapped coefficients for such a model is an error.

A bootstrapped p-value is never reported below $1/B$, where B is the number of usable bootstrap replicates: with no draw crossing the estimate, the true p-value could be anywhere in $[0, 1/B)$, so it is floored at $1/B$ rather than reported as exactly 0. `print.summary.evzinb()` / `print.summary.evinb()` show this as " $< 1/B$ " (e.g. " < 0.01 " for 100 usable bootstraps) rather than the default, misleadingly precise " $< 2e-16$ ".

Value

An EVINB summary object

Examples

```

data(genevzinb2)
model <- evinb(y~x1+x2+x3,data=genevzinb2, n_bootstraps = 5)
summary(model)

```

summary.evzinb	<i>EVZINB summary function</i>
----------------	--------------------------------

Description

EVZINB summary function

Usage

```

## S3 method for class 'evzinb'
summary(
  object,
  coef = c("original", "bootstrapped_mean", "bootstrapped_median"),
  standard_error = TRUE,
  p_value = c("bootstrapped", "approx", "both", "none"),
  bootstrapped_props = c("none", "mean", "median"),
  approx_t_value = TRUE,
  symmetric_bootstrap_p = TRUE,
  exclude_degenerate = TRUE,
  ...
)

```

Arguments

object	an EVZINB object
coef	Type of coefficients. Original are the coefficient estimates from the non-bootstrapped version of the model. 'bootstrapped_mean' are the mean coefficients across bootstraps, and 'bootstrapped_median' are the median coefficients across bootstraps
standard_error	Should standard errors be computed?
p_value	What type of p_values should be computed? 'bootstrapped' are bootstrapped p_values through confidence interval inversion. 'approx' are p-values based on the t-value produced by dividing the coefficient with the standard error. 'both' returns both.
bootstrapped_props	Type of bootstrapped proportions of component proportions to be returned
approx_t_value	Should approximate t-values be returned
symmetric_bootstrap_p	Should bootstrap p-values be computed as symmetric (leaving alpha/2 percent in each tail)? FALSE gives non-symmetric, but narrower, intervals. TRUE corresponds most closely to conventional p-values.
exclude_degenerate	Drop bootstrap replicates flagged degenerate (default TRUE); see the alpha_floor argument of evinf_control().
...	Additional arguments passed to the summary function

Details

When object was fitted with `bootstrap = FALSE`, the bootstrap-based quantities (standard errors, p-values, approximate t-values and bootstrapped proportions) are unavailable; the summary then reports the point estimates only, `n_failed_bootstraps` is NA, and a message is emitted. Requesting bootstrapped coefficients for such a model is an error.

A bootstrapped p-value is never reported below $1/B$, where B is the number of usable bootstrap replicates: with no draw crossing the estimate, the true p-value could be anywhere in $[0, 1/B)$, so it is floored at $1/B$ rather than reported as exactly 0. `print.summary.evzinb()` / `print.summary.evinb()` show this as " $< 1/B$ " (e.g. " < 0.01 " for 100 usable bootstraps) rather than the default, misleadingly precise " $< 2e-16$ ".

Value

An EVZINB summary object

Examples

```
data(genevzinb2)
model <- evzinb(y~x1+x2+x3,data=genevzinb2, n_bootstraps = 5)
summary(model)
```

tidy.evinb

*EVINB tidy function***Description**

EVINB tidy function

Usage

```
## S3 method for class 'evinb'
tidy(
  x,
  component = c("all", "count", "evi", "pareto"),
  coef_type = c("original", "bootstrap_mean", "bootstrap_median"),
  standard_error = TRUE,
  p_value = c("bootstrapped", "approx", "none"),
  confint = c("none", "bootstrapped", "approx"),
  conf_level = 0.95,
  approx_t_value = TRUE,
  symmetric_bootstrap_p = TRUE,
  exclude_degenerate = TRUE,
  ...
)
```

Arguments

x	An evinb object
component	Which component should be shown? One of "count", "evi", "pareto" or "all" (the default).
coef_type	Type of coefficients. Original are the coefficient estimates from the non-bootstrapped version of the model. 'bootstrap_mean' are the mean coefficients across bootstraps, and 'bootstrap_median' are the median coefficients across bootstraps
standard_error	Should bootstrapped standard errors be computed?
p_value	What type of p_values should be computed? 'bootstrapped' are bootstrapped p_values through confidence interval inversion. 'approx' are p-values based on the t-value produced by dividing the coefficient with the standard error.
confint	What type of confidence interval should be computed: "none", "bootstrapped" (percentile interval from the bootstrap distribution) or "approx" (estimate +/- qt() * std.error).
conf_level	Confidence level for the confidence interval
approx_t_value	Should approximate t-values be returned
symmetric_bootstrap_p	Should bootstrap p-values be computed as symmetric (leaving alpha/2 percent in each tail)? FALSE gives non-symmetric, but narrower, intervals. TRUE corresponds most closely to conventional p-values.

```

exclude_degenerate      Drop bootstrap replicates flagged degenerate (default TRUE); see the alpha_floor
                        argument of evinf_control().
...                      Other arguments passed to the tidy function

```

Details

When `x` was fitted with `bootstrap = FALSE`, standard errors, p-values, t-values and confidence intervals are unavailable; the estimate column is returned on its own and a message is emitted. Requesting bootstrapped coefficients for such a model is an error.

A bootstrapped p-value is never reported below $1 / B$, where B is the number of usable bootstrap replicates: with no draw crossing the estimate, the true p-value could be anywhere in $[0, 1/B)$, so it is floored at $1/B$ rather than reported as exactly 0.

With `component = "all"` the output has one row per coefficient *per component* (a `y.level` column names the component). Like `nnet::multinom()`, this multi-component shape cannot be rendered by `modelsummary::modelsummary()` with its default arguments; pass `shape = term + y.level ~ model` so each component becomes its own block of rows.

Value

A tibble with one row per coefficient

Examples

```

data(genevzinb2)
model <- evinb(y~x1+x2+x3,data=genevzinb2, n_bootstraps = 5)
tidy(model)

```

tidy.evzinb	<i>EVZINB tidy function</i>
-------------	-----------------------------

Description

EVZINB tidy function

Usage

```

## S3 method for class 'evzinb'
tidy(
  x,
  component = c("all", "count", "zero", "evi", "pareto"),
  coef_type = c("original", "bootstrap_mean", "bootstrap_median"),
  standard_error = TRUE,
  p_value = c("bootstrapped", "approx", "none"),
  confint = c("none", "bootstrapped", "approx"),
  conf_level = 0.95,

```

```

    approx_t_value = TRUE,
    symmetric_bootstrap_p = TRUE,
    exclude_degenerate = TRUE,
    ...
  )

```

Arguments

<code>x</code>	An evzinb object
<code>component</code>	Which component should be shown? One of "count", "zi", "evi", "pareto" or "all" (the default).
<code>coef_type</code>	Type of coefficients. Original are the coefficient estimates from the non-bootstrapped version of the model. 'bootstrap_mean' are the mean coefficients across bootstraps, and 'bootstrap_median' are the median coefficients across bootstraps
<code>standard_error</code>	Should bootstrapped standard errors be computed?
<code>p_value</code>	What type of p_values should be computed? 'bootstrapped' are bootstrapped p_values through confidence interval inversion. 'approx' are p-values based on the t-value produced by dividing the coefficient with the standard error.
<code>confint</code>	What type of confidence interval should be computed: "none", "bootstrapped" (percentile interval from the bootstrap distribution) or "approx" (estimate +/- qt() * std.error).
<code>conf_level</code>	Confidence level for the confidence interval
<code>approx_t_value</code>	Should approximate t-values be returned
<code>symmetric_bootstrap_p</code>	Should bootstrap p-values be computed as symmetric (leaving alpha/2 percent in each tail)? FALSE gives non-symmetric, but narrower, intervals. TRUE corresponds most closely to conventional p-values.
<code>exclude_degenerate</code>	Drop bootstrap replicates flagged degenerate (default TRUE); see the <code>alpha_floor</code> argument of <code>evinf_control()</code> .
<code>...</code>	Other arguments passed to the tidy function

Details

When `x` was fitted with `bootstrap = FALSE`, standard errors, p-values, t-values and confidence intervals are unavailable; the estimate column is returned on its own and a message is emitted. Requesting bootstrapped coefficients for such a model is an error.

A bootstrapped p.value is never reported below $1 / B$, where B is the number of usable bootstrap replicates: with no draw crossing the estimate, the true p-value could be anywhere in $[0, 1/B)$, so it is floored at $1/B$ rather than reported as exactly 0.

With `component = "all"` the output has one row per coefficient *per component* (a `y.level` column names the component). Like `nnet::multinom()`, this multi-component shape cannot be rendered by `modelsummary::modelsummary()` with its default arguments; pass `shape = term + y.level ~ model` so each component becomes its own block of rows.

Value

A tibble with one row per coefficient

Examples

```
data(genevzinb2)
model <- evzinb(y~x1+x2+x3,data=genevzinb2, n_bootstraps = 5)
tidy(model)

# multi-component table with modelsummary
if (requireNamespace("modelsummary", quietly = TRUE) &&
    requireNamespace("broom", quietly = TRUE)) {
  modelsummary::modelsummary(
    model,
    shape = term + y.level ~ model,
    gof_map = gof_map_evinf()
  )
}

data(hks)
hks_mod <- evzinb(osvAll ~ troopLag + lnpop + brv_AllLag_log,
                  data = hks, n_bootstraps = 2, multicore = FALSE)
tidy(hks_mod)
```

tidy.nbboot

Tidy function for nbboot

Description

Tidy function for nbboot

Usage

```
## S3 method for class 'nbboot'
tidy(
  x,
  coef_type = c("original", "bootstrap_mean", "bootstrap_median"),
  standard_error = TRUE,
  p_value = c("bootstrapped", "approx", "none"),
  confint = c("none", "bootstrapped", "approx"),
  conf_level = 0.95,
  approx_t_value = TRUE,
  symmetric_bootstrap_p = TRUE,
  include_ylev = FALSE,
  ...
)
```

Arguments

<code>x</code>	A fitted bootstrapped zero-inflated model
<code>coef_type</code>	What type of coefficient should be reported, original, bootstrapped mean, or bootstrapped median
<code>standard_error</code>	Should bootstrapped standard errors be reported?
<code>p_value</code>	What type of p-value should be reported? Bootstrapped <code>p_values</code> , approximate p-values, or none?
<code>confint</code>	What type of confidence intervals should be reported? Bootstrapped <code>p_values</code> , approximate p-values, or none?
<code>conf_level</code>	Confidence level for confidence intervals
<code>approx_t_value</code>	Should approximate <code>t_values</code> be reported
<code>symmetric_bootstrap_p</code>	Should bootstrap p-values be computed as symmetric (leaving $\alpha/2$ percent in each tail)? FALSE gives non-symmetric, but narrower, intervals. TRUE corresponds most closely to conventional p-values.
<code>include_ylev</code>	Logical. Should <code>y.lev</code> be included in the tidy output? Makes for nicer tables when using <code>modelsummary</code>
<code>...</code>	Other arguments to be passed to <code>tidy</code>

Value

A tidy function for a bootstrapped nb model

Examples

```
data(genevzinb2)
model <- evzinb(y~x1+x2+x3,data=genevzinb2, n_bootstraps = 5)
zinb_comp <- compare_models(model)
tidy(zinb_comp$nb)
```

tidy.poissonboot

Tidy function for poissonboot

Description

Tidy function for poissonboot

Usage

```
## S3 method for class 'poissonboot'
tidy(
  x,
  coef_type = c("original", "bootstrap_mean", "bootstrap_median"),
  standard_error = TRUE,
  p_value = c("bootstrapped", "approx", "none"),
  confint = c("none", "bootstrapped", "approx"),
  conf_level = 0.95,
  approx_t_value = TRUE,
  symmetric_bootstrap_p = TRUE,
  include_ylev = FALSE,
  ...
)
```

Arguments

<code>x</code>	A fitted bootstrapped Poisson model
<code>coef_type</code>	What type of coefficient should be reported, original, bootstrapped mean, or bootstrapped median
<code>standard_error</code>	Should bootstrapped standard errors be reported?
<code>p_value</code>	What type of p-value should be reported? Bootstrapped p_values, approximate p-values, or none?
<code>confint</code>	What type of confidence intervals should be reported? Bootstrapped p_values, approximate p-values, or none?
<code>conf_level</code>	Confidence level for confidence intervals
<code>approx_t_value</code>	Should approximate t_values be reported
<code>symmetric_bootstrap_p</code>	Should bootstrap p-values be computed as symmetric (leaving alpha/2 percent in each tail)? FALSE gives non-symmetric, but narrower, intervals. TRUE corresponds most closely to conventional p-values.
<code>include_ylev</code>	Logical. Should y.lev be included in the tidy output? Makes for nicer tables when using modelsummary
<code>...</code>	Other arguments to be passed to tidy

Value

A tidy function for a bootstrapped Poisson model

Examples

```
data(genevzinb2)
model <- evzinb(y~x1+x2+x3,data=genevzinb2, family = "poisson", n_bootstraps = 5)
zip_comp <- compare_models(model)
tidy(zip_comp$poisson)
```

tidy.zinbboot	<i>Tidy function for zinbboot</i>
---------------	-----------------------------------

Description

Tidy function for zinbboot

Usage

```
## S3 method for class 'zinbboot'
tidy(
  x,
  component = c("all", "count", "zero"),
  coef_type = c("original", "bootstrap_mean", "bootstrap_median"),
  standard_error = TRUE,
  p_value = c("bootstrapped", "approx", "none"),
  confint = c("none", "bootstrapped", "approx"),
  conf_level = 0.95,
  approx_t_value = TRUE,
  symmetric_bootstrap_p = TRUE,
  ...
)
```

Arguments

x	A fitted bootstrapped zero-inflated model
component	Which component should be shown?
coef_type	What type of coefficient should be reported, original, bootstrapped mean, or bootstrapped median
standard_error	Should bootstrapped standard errors be reported?
p_value	What type of p-value should be reported? Bootstrapped p_values, approximate p-values, or none?
confint	What type of confidence intervals should be reported? Bootstrapped p_values, approximate p-values, or none?
conf_level	Confidence level for confidence intervals
approx_t_value	Should approximate t_values be reported
symmetric_bootstrap_p	Should bootstrap p-values be computed as symmetric (leaving alpha/2 percent in each tail)? FALSE gives non-symmetric, but narrower, intervals. TRUE corresponds most closely to conventional p-values.
...	Other arguments to be passed to tidy

Value

A tidy function for a bootstrapped zinb model

Examples

```
data(genevzinb2)
model <- evzinb(y~x1+x2+x3,data=genevzinb2, n_bootstraps = 5)
zinb_comp <- compare_models(model)
tidy(zinb_comp$zinb)
```

tidy.zipboot	<i>Tidy function for zipboot</i>
--------------	----------------------------------

Description

Tidy function for zipboot

Usage

```
## S3 method for class 'zipboot'
tidy(
  x,
  component = c("all", "count", "zero"),
  coef_type = c("original", "bootstrap_mean", "bootstrap_median"),
  standard_error = TRUE,
  p_value = c("bootstrapped", "approx", "none"),
  confint = c("none", "bootstrapped", "approx"),
  conf_level = 0.95,
  approx_t_value = TRUE,
  symmetric_bootstrap_p = TRUE,
  ...
)
```

Arguments

x	A fitted bootstrapped zero-inflated Poisson model
component	Which component should be shown?
coef_type	What type of coefficient should be reported, original, bootstrapped mean, or bootstrapped median
standard_error	Should bootstrapped standard errors be reported?
p_value	What type of p-value should be reported? Bootstrapped p_values, approximate p-values, or none?
confint	What type of confidence intervals should be reported? Bootstrapped p_values, approximate p-values, or none?
conf_level	Confidence level for confidence intervals
approx_t_value	Should approximate t_values be reported

```

symmetric_bootstrap_p
    Should bootstrap p-values be computed as symmetric (leaving alpha/2 percent
    in each tail)? FALSE gives non-symmetric, but narrower, intervals. TRUE cor-
    responds most closely to conventional p-values.
...
    Other arguments to be passed to tidy

```

Value

A tidy function for a bootstrapped ZIP model

Examples

```

data(genevzinb2)
model <- evzinb(y~x1+x2+x3,data=genevzinb2, family = "poisson", n_bootstraps = 5)
zip_comp <- compare_models(model)
tidy(zip_comp$zip)

```

update.evzinb	<i>Update and re-fit an evzinb / evinb model</i>
---------------	--

Description

Update and re-fit an evzinb / evinb model

Usage

```

## S3 method for class 'evzinb'
update(
  object,
  formula_nb.,
  formula_zi.,
  formula_evi.,
  formula_pareto.,
  ...,
  evaluate = TRUE
)

## S3 method for class 'evinb'
update(
  object,
  formula_nb.,
  formula_evi.,
  formula_pareto.,
  ...,
  evaluate = TRUE
)

```

Arguments

`object` A fitted model (must carry `object$call`).

`formula_nb.`, `formula_zi.`, `formula_evi.`, `formula_pareto.`
 Optional `update.formula`-style changes per component, e.g. `formula_pareto. = . ~ . - x3`. Each component is updated against its *own current* formula (the one the fitted object actually used, whether the user supplied it or it was inherited from `formula_nb.` when NULL). Changing `formula_nb.` does not propagate to a component formula that was originally left NULL and inherited from it – that component keeps whatever formula it was fitted with; pass that component’s own `formula_*` explicitly to change it too.

`...` Other arguments of `evzinb` / `evinb` to change.

`evaluate` If TRUE (default) re-fit; otherwise return the updated call.

Details

If `object`’s candidate range for C_{EV} was itself data-driven (`control$c.lim` was NULL at the original fit; see `object$c.lim.default`), and `data` is one of the arguments being changed, `control$c.lim` and `control$init.C` are reset to NULL so they are re-resolved from the new data (with the usual message) instead of silently reusing the range chosen for the old data. A `c.lim` the user pinned explicitly is always kept as-is.

Value

The updated fit, or the call.

vcov.evzinb

Bootstrap covariance matrix of an evzinb / evinb model

Description

Bootstrap covariance matrix of an evzinb / evinb model

Usage

```
## S3 method for class 'evzinb'
vcov(object, exclude_degenerate = TRUE, ...)

## S3 method for class 'evinb'
vcov(object, exclude_degenerate = TRUE, ...)
```

Arguments

`object` A fitted model with bootstraps.

`exclude_degenerate`
 Drop bootstrap replicates flagged degenerate (default TRUE); see `evinf_control`.

`...` Unused.

Details

c_ev is estimated on the grid of unique observed values of the response, not by a smooth optimiser, so it sits on a discrete scale. Its bootstrap variance (and a percentile confint() on it) is meaningful, but delta-method standard errors that perturb c_ev continuously – as `marginaleffects::avg_slopes()` etc. do – should be read with that in mind. For uncertainty on covariate *effects*, prefer the bootstrap route, `marginal_effects()`.

Value

The covariance matrix of the bootstrap coefficient draws, with the same names and order as `coef(object)` (includes `alpha_nb` and `c_ev`).

Index

* datasets

- genevzinb, [32](#)
- genevzinb2, [33](#)
- gm_evzinb, [37](#)
- hks, [39](#)

add_bootstraps, [3](#), [4](#), [13](#), [18](#), [29](#), [30](#), [41](#), [44](#), [50](#), [54](#)

augment_evinb, [5](#)

augment_evzinb, [5](#), [5](#)

c_profile, [6](#)

check_evinf, [7](#)

classify_states, [6](#), [8](#)

coef_evinb (coef_evzinb), [9](#)

coef_evzinb, [9](#)

coefficient_extractor, [9](#)

compare_fit, [11](#), [61](#)

compare_models, [4](#), [11](#), [12](#), [13](#), [18](#), [30](#), [41](#), [44](#), [45](#), [50](#), [54](#), [61](#)

confint_evinb (confint_evzinb), [14](#)

confint_evzinb, [14](#)

evinb, [15](#), [23](#), [26](#), [40](#), [77](#)

evinf-s3-accessors, [19](#)

evinf-s3-predict, [20](#)

evinf_bench_data, [22](#)

evinf_control, [7](#), [10](#), [11](#), [15](#), [18](#), [23](#), [30](#), [32](#), [34](#), [35](#), [43](#), [45](#), [77](#)

evinf_family, [18](#), [26](#), [29](#)

evinf_pmap, [25](#)

evzinb, [23](#), [24](#), [26](#), [27](#), [40](#), [77](#)

evzinbcomp-tidiers, [31](#)

failed_bootstraps, [32](#), [41](#)

fitted_evinb (evinf-s3-predict), [20](#)

fitted_evzinb (evinf-s3-predict), [20](#)

formula_evinb (evinf-s3-accessors), [19](#)

formula_evzinb (evinf-s3-accessors), [19](#)

genevzinb, [32](#)

genevzinb2, [22](#), [33](#)

glance, [34–37](#), [61](#)

glance_evinb, [33](#)

glance_evzinb, [17](#), [29](#), [34](#)

glance_evzinbcomp (evzinbcomp-tidiers), [31](#)

glance.nbboot (glance.zinbboot), [35](#)

glance.poissonboot (glance.zipboot), [36](#)

glance.zinbboot, [35](#)

glance.zipboot, [36](#)

gm_evzinb, [37](#), [38](#)

gof_map_evinf, [37](#), [38](#)

hks, [17](#), [28](#), [39](#)

hurdle, [26](#)

logLik_evinb (evinf-s3-accessors), [19](#)

logLik_evzinb (evinf-s3-accessors), [19](#)

lr_test, [4](#), [13](#), [18](#), [30](#), [40](#), [41](#), [44](#), [50](#), [54](#)

marginal_effects, [4](#), [13](#), [18](#), [30](#), [41](#), [42](#), [44](#), [50](#), [54](#), [78](#)

model.frame_evinb (evinf-s3-accessors), [19](#)

model.frame_evzinb (evinf-s3-accessors), [19](#)

multisession, [3](#), [13](#), [17](#), [28](#), [41](#), [43](#), [49](#), [52](#)

nobs_evinb (evinf-s3-accessors), [19](#)

nobs_evzinb (evinf-s3-accessors), [19](#)

oob_evaluation, [17](#), [29](#), [45](#)

plot_evinb (plot_evzinb), [46](#)

plot_evzinb, [46](#)

plot_evzinbcomp (compare_fit), [11](#)

plot_c_profile, [47](#)

predict_evinb, [48](#)

predict_evzinb, [4](#), [13](#), [18](#), [30](#), [41](#), [44](#), [50](#), [51](#), [54](#)

predict.nbboot, [55](#)

predict.poissonboot, [56](#)
predict.zinbboot, [57](#)
predict.zipboot, [58](#)
predict_grid, [47](#), [59](#)
print.evinb, [60](#)
print.evzinb, [61](#)
print.evzinbcomp, [61](#)
print.summary.evinb
 (print.summary.evzinb), [62](#)
print.summary.evzinb, [62](#)

residuals.evinb (evinf-s3-predict), [20](#)
residuals.evzinb (evinf-s3-predict), [20](#)
revinb_fit, [63](#)
revzinb_fit, [63](#)

simulate, [21](#)
simulate(), [63](#)
simulate.evinb, [63](#)
simulate.evinb (evinf-s3-predict), [20](#)
simulate.evzinb, [64](#)
simulate.evzinb (evinf-s3-predict), [20](#)
state_table, [64](#)
summary.evinb, [65](#)
summary.evzinb, [66](#)

terms.evinb (evinf-s3-accessors), [19](#)
terms.evzinb (evinf-s3-accessors), [19](#)
tidy.evinb, [68](#)
tidy.evzinb, [69](#)
tidy.evzinbcomp (evzinbcomp-tidiers), [31](#)
tidy.nbboot, [71](#)
tidy.poissonboot, [72](#)
tidy.zinbboot, [74](#)
tidy.zipboot, [75](#)

update.evinb (update.evzinb), [76](#)
update.evzinb, [76](#)
update.formula, [77](#)

vcov.evinb (vcov.evzinb), [77](#)
vcov.evzinb, [77](#)

zeroinfl, [26](#)