

Package: beezdemand (via r-universe)

September 22, 2026

Version 0.3.0

Date 2026-08-16

Title Behavioral Economic Easy Demand

Author Brent Kaplan [aut, cre, cph]

(<https://orcid.org/0000-0002-3758-6776>), Shawn Gilroy [ctb]

Maintainer Brent Kaplan <bkaplan.ku@gmail.com>

Description Facilitates many of the analyses performed in studies of behavioral economic demand. The package supports commonly-used options for modeling operant demand including (1) data screening proposed by Stein, Koffarnus, Snider, Quisenberry, & Bickel (2015; <[doi:10.1037/pha0000020](https://doi.org/10.1037/pha0000020)>), (2) fitting models of demand such as linear (Hursh, Raslear, Bauman, & Black, 1989, <[doi:10.1007/978-94-009-2470-3_22](https://doi.org/10.1007/978-94-009-2470-3_22)>), exponential (Hursh & Silberberg, 2008, <[doi:10.1037/0033-295X.115.1.186](https://doi.org/10.1037/0033-295X.115.1.186)>) and modified exponential (Koffarnus, Franck, Stein, & Bickel, 2015, <[doi:10.1037/pha0000045](https://doi.org/10.1037/pha0000045)>), (3) calculating numerous measures relevant to applied behavioral economists (Intensity, Pmax, Omax), (4) nonlinear mixed-effects demand models with subject-level random effects, fit either with 'nlme' or with a 'TMB' backend that also provides a two-part (hurdle) model for zero-inflated consumption, and (5) Monte Carlo power analysis for condition differences in demand parameters. Also supports plotting and comparing data.

Depends R (>= 4.1.0)

Imports nlsr, nlstools, ggplot2, stats, optimx, broom, lme4, emmeans, minpack.lm, nls.multstart, performance, scales, tibble, lifecycle, dplyr, tidyr, nlme, rlang, cli, utils, TMB (>= 1.9.0)

Suggests broom.mixed, car, GGally, knitr, tidyverse, rmarkdown, purrr, pwr, conflicted, devtools, here, readr, patchwork, testthat (>= 3.0.0), withr

LinkingTo TMB, RcppEigen

License GPL (>= 2)

URL <https://brentkaplan.github.io/beeze-demand/>,
<https://github.com/brentkaplan/beeze-demand>

BugReports <https://github.com/brentkaplan/beeze-demand/issues>

Encoding UTF-8

Language en-US

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

VignetteBuilder knitr

SystemRequirements GNU make

Config/testthat/edition 3

Config/testthat/parallel true

Config/testthat/start-first fit_demand_tmb, cross-price,
fit_demand_fixed, diagnostics, model-comparison

Config/pak/sysreqs cmake make libicu-dev

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

Date/Publication 2026-09-21 19:20:21 UTC

RemoteUrl <https://github.com/brentkaplan/beeze-demand>

RemoteRef develop

RemoteSha 5ac2438cd75659a099d7d1e2855fe49be9b2aab4

Contents

AIC.beeze-demand_hurdle	7
annotation_logticks2	7
anova.beeze-demand_hurdle	8
anova.beeze-demand_nlme	9
anova.beeze-demand_tmb	10
apt	11
apt_full	12
augment.beeze-demand_fixed	12
augment.beeze-demand_hurdle	13
augment.beeze-demand_nlme	15
augment.beeze-demand_tmb	16
augment.cp_model_lm	17
augment.cp_model_lmer	17
augment.cp_model_nls	18
beeze-demand_descriptive_methods	18
beeze-demand_empirical_methods	20
BIC.beeze-demand_hurdle	22
boot_demand	22
calc_group_metrics	24

calc_group_metrics.beezdemand_tmb	25
calc_observed_pmax_omax	26
calc_omax_pmax	27
calculate_amplitude_persistence	28
cannabisCigarettes	30
ChangeData	31
check_demand_model	32
check_systematic_cp	34
check_systematic_demand	35
check_unsystematic_cp	37
CheckCols	39
CheckUnsystematic	40
coef-methods	41
coef.beezdemand_fixed	42
coef.beezdemand_fixed_grouped	42
coef.beezdemand_hurdle	43
coef.beezdemand_nlme	43
coef.beezdemand_tmb	44
compare_hurdle_models	45
compare_models	46
confint.beezdemand_fixed	48
confint.beezdemand_hurdle	49
confint.beezdemand_nlme	50
confint.beezdemand_tmb	51
confint.cp_model_lm	53
confint.cp_model_lmer	54
confint.cp_model_nls	54
cp	56
cp_posthoc_intercepts	56
cp_posthoc_slopes	57
etm	58
extract_coefficients	58
ExtraF	59
find_n_demand	60
fit_cp_linear	63
fit_cp_nls	65
fit_demand_fixed	68
fit_demand_hurdle	71
fit_demand_mixed	74
fit_demand_tmb	77
FitCurves	83
FitMeanCurves	85
fitted.beezdemand_hurdle	86
fitted.beezdemand_tmb	87
fixed-demand	88
fixef.beezdemand_nlme	88
fixef.beezdemand_tmb	89
fixef.cp_model_lmer	89

formula.beezdemand_hurdle	90
formula.beezdemand_tmb	90
get_demand_comparisons	91
get_demand_comparisons.beezdemand_tmb	93
get_demand_param_emms	95
get_demand_param_emms.beezdemand_tmb	97
get_demand_param_trends	98
get_descriptive_summary	99
get_empirical_measures	101
get_hurdle_param_summary	103
get_individual_coefficients	104
get_k	106
get_observed_demand_param_emms	107
get_subject_pars	109
get_subject_pars.beezdemand_nlme	110
get_subject_pars.beezdemand_tmb	111
GetAnalyticPmax	112
GetAnalyticPmaxFallback	113
GetDescriptives	114
GetEmpirical	115
GetK	116
GetSharedK	117
GetValsForSim	118
glance.beezdemand_fixed	118
glance.beezdemand_fixed_grouped	119
glance.beezdemand_hurdle	120
glance.beezdemand_nlme	120
glance.beezdemand_systematicity	121
glance.beezdemand_tmb	122
glance.cp_model_lm	123
glance.cp_model_lmer	123
glance.cp_model_nls	124
ko	125
lambertW	125
ll4	126
ll4_inv	127
logLik.beezdemand_hurdle	128
lowNicClean	129
model.matrix.beezdemand_hurdle	129
model.matrix.beezdemand_tmb	130
nobs.beezdemand_hurdle	131
nobs.cp_model_lm	131
nobs.cp_model_lmer	132
nobs.cp_model_nls	132
ongoingETM	133
palette_beezdemand	133
pivot_demand_data	134
plot-theme	136

plot.beezdemand_fixed	136
plot.beezdemand_fixed_grouped	138
plot.beezdemand_hurdle	138
plot.beezdemand_nlme	141
plot.beezdemand_tmb	143
plot.cp_model_lm	145
plot.cp_model_lmer	146
plot.cp_model_nls	147
plot_alpha_distribution	148
plot_demand_overlay	149
plot_elasticity	151
plot_expenditure	152
plot_loss_profile	154
plot_loss_surface	155
plot_model_comparison	157
plot_qq	158
plot_re_diagnostics	159
plot_residuals	160
plot_subject	161
PlotCurve	162
PlotCurves	163
power_demand	164
predict.beezdemand_fixed	168
predict.beezdemand_hurdle	169
predict.beezdemand_nlme	172
predict.beezdemand_tmb	174
predict.cp_model_lm	176
predict.cp_model_lmer	177
predict.cp_model_nls	178
print.anova.beezdemand_hurdle	178
print.beezdemand_comparison	179
print.beezdemand_diagnostic_plots	179
print.beezdemand_diagnostics	180
print.beezdemand_fixed	180
print.beezdemand_fixed_grouped	181
print.beezdemand_hurdle	181
print.beezdemand_model_comparison	182
print.beezdemand_nlme	182
print.beezdemand_summary	183
print.beezdemand_systematicity	184
print.beezdemand_tmb	184
print.cp_model_lm	185
print.cp_model_lmer	185
print.cp_model_nls	186
print.cp_posthoc	186
print.summary.beezdemand_fixed	187
print.summary.beezdemand_hurdle	187
print.summary.beezdemand_nlme	188

print.summary.beezdemand_systematicity	188
print.summary.beezdemand_tmb	189
print.summary.cp_model_lm	189
print.summary.cp_model_lmer	190
print.summary.cp_model_nls	190
print.summary.cp_unsystematic	191
print_mc_summary	191
pseudo_ll4_trans	192
ranef.beezdemand_nlme	193
ranef.beezdemand_tmb	194
ranef.cp_model_lmer	195
RecodeOutliers	195
ReplaceZeros	196
residuals.beezdemand_hurdle	197
residuals.beezdemand_tmb	198
run_hurdle_monte_carlo	199
scale_color_beezdemand	200
scale_fill_beezdemand	201
scale_ll4	202
simulate_hurdle_data	203
SimulateDemand	206
summary.beezdemand_fixed	207
summary.beezdemand_fixed_grouped	208
summary.beezdemand_hurdle	208
summary.beezdemand_nlme	209
summary.beezdemand_systematicity	211
summary.beezdemand_tmb	211
summary.cp_model_lm	212
summary.cp_model_lmer	213
summary.cp_model_nls	213
summary.cp_unsystematic	214
systematic-wrappers	215
theme_apr	215
theme_beezdemand	216
tidy.beezdemand_comparison	216
tidy.beezdemand_fixed	217
tidy.beezdemand_fixed_grouped	218
tidy.beezdemand_hurdle	218
tidy.beezdemand_nlme	219
tidy.beezdemand_systematicity	220
tidy.beezdemand_tmb	221
tidy.cp_model_lm	222
tidy.cp_model_lmer	223
tidy.cp_model_nls	223
update.beezdemand_tmb	224
VarCorr.beezdemand_tmb	225
vcov.beezdemand_hurdle	226
vcov.beezdemand_tmb	227

AIC.beezdemand_hurdle	<i>AIC for Hurdle Demand Model</i>
-----------------------	------------------------------------

Description

AIC for Hurdle Demand Model

Usage

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

Arguments

- object A beezdemand_hurdle object.
- ... Additional arguments (unused).
- k Penalty per parameter. Default is 2 (standard AIC).

Value

A numeric AIC value.

annotation_logticks2	<i>annotation_logticks2</i>
----------------------	-----------------------------

Description

Creates annotation layer

Usage

```
annotation_logticks2(
  base = 10,
  sides = "bl",
  scaled = TRUE,
  short = unit(0.1, "cm"),
  mid = unit(0.2, "cm"),
  long = unit(0.3, "cm"),
  colour = "black",
  size = 0.5,
  linetype = 1,
  alpha = 1,
  data = data.frame(x = NA),
  color = NULL,
  ...
)
```

Arguments

base	base for drawing in scale
sides	sides to draw, by default bottom and left
scaled	true by default
short	short tick settings
mid	mid tick settings
long	long tick settings
colour	default to black colour
size	size for labels
linetype	default linetype
alpha	default alpha level
data	data to include
color	colors to include
...	additional arguments

Details

Inherit and extend layer for use in ggplot draw

Value

ggplot2 layer

Author(s)

Shawn Gilroy shawn.gilroy@temple.edu

anova.beezdemand_hurdle

ANOVA Method for Hurdle Demand Models

Description

Compare nested hurdle demand models using likelihood ratio tests.

Usage

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

Arguments

object	A beezdemand_hurdle model.
...	Additional beezdemand_hurdle models to compare.

Details

All models must be fit to the same data. Models are ordered by degrees of freedom, and sequential likelihood ratio tests are performed.

Value

An object of class `anova.beezdemand` containing:

table Data frame with model comparison statistics

lrt Likelihood ratio test results

Examples

```
data(apt)
fit2 <- fit_demand_hurdle(apt, y_var = "y", x_var = "x", id_var = "id",
  random_effects = c("zeros", "q0"))
fit3 <- fit_demand_hurdle(apt, y_var = "y", x_var = "x", id_var = "id",
  random_effects = c("zeros", "q0", "alpha"))
anova(fit2, fit3)
```

anova.beezdemand_nlme *ANOVA Method for NLME Demand Models*

Description

Compare nested NLME demand models using likelihood ratio tests.

Usage

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

Arguments

object A beezdemand_nlme model.
... Additional beezdemand_nlme models to compare.

Details

For NLME models, this method delegates to `nlme::anova.lme()` on the underlying model objects when possible.

Value

An object of class `anova.beezdemand` containing model comparison statistics.

Examples

```
data(ko)
fit1 <- fit_demand_mixed(ko, y_var = "y_ll4", x_var = "x",
                        id_var = "monkey", equation_form = "zben",
                        random_effects = Q0 ~ 1)
fit2 <- fit_demand_mixed(ko, y_var = "y_ll4", x_var = "x",
                        id_var = "monkey", equation_form = "zben",
                        random_effects = Q0 + alpha ~ 1)

anova(fit1, fit2)
```

anova.beezdemand_tmb *Joint Wald and likelihood-ratio tests for a TMB demand fit*

Description

For a single fit, computes joint Wald-chi-square tests on grouped fixed-effect coefficients. For multiple fits passed via `...`, performs sequential likelihood-ratio tests on nested models.

Usage

```
## S3 method for class 'beezdemand_tmb'
anova(
  object,
  ...,
  test = c("Wald", "LRT", "AIC"),
  terms = NULL,
  group_by = c("auto", "parameter", "term")
)
```

Arguments

<code>object</code>	A beezdemand_tmb fit.
<code>...</code>	Additional beezdemand_tmb fits for nested comparison.
<code>test</code>	One of "Wald", "LRT", "AIC". Default: "Wald" for a single fit, "LRT" when extra fits are supplied.
<code>terms</code>	NULL (all fixed effects), a character vector of term names, or a named list mapping group labels to term-name vectors. Term names match display names (Q0:genderMale) or raw names.
<code>group_by</code>	One of "auto" (group non-intercept terms by parameter x factor/covariate), "parameter" (one group per Q0 / alpha), or "term" (one row per coefficient).

Details

The Wald statistic for a coefficient block is $W = \beta_g' \Sigma_{gg}^{-1} \beta_g$, asymptotically χ^2 on `length(beta_g)` df. An exactly rank-deficient (perfectly collinear) block has a singular Σ_{gg} and triggers an explicit error. A near-collinear block is not detected: Σ_{gg} stays invertible and W becomes large and unstable, so such a value should be interpreted with caution. For multiple fits, the likelihood-ratio test screens for detectable non-nesting (equal or decreasing degrees of freedom, or a larger model with lower log-likelihood) but cannot prove nesting from log-likelihood and df alone. Pass genuinely nested models. Rows of the multiple-fit table are ordered by ascending degrees of freedom, and the `Model` column labels them `Model1`, `Model2`, ... in that order.

Value

For a single fit, a tibble with `Group`, `Chisq`, `df`, `p.value`. For multiple fits, a tibble with `Model`, `df`, `AIC`, `Chisq`, ``Pr(>Chisq)``.

See Also

[anova.beezdemand_nlme\(\)](#), [confint.beezdemand_tmb\(\)](#).

Examples

```
data(apt_full)
# 40 subjects per gender keep the example fast; use the full data in practice
ids <- unique(apt_full[c("id", "gender")])
ids <- ids[ids$gender %in% c("Male", "Female"), ]
keep <- unlist(lapply(split(ids$id, ids$gender), head, 40))
dat <- apt_full[apt_full$id %in% keep, ]
fit <- fit_demand_tmb(dat, equation = "exponential",
                      factors = "gender", verbose = 0)

anova(fit)
anova(fit, group_by = "parameter")
```

apt

Example alcohol purchase task data

Description

A dataset containing alcohol purchase task data for a small number of participants

Usage

```
apt
```

Format

Long-form data.frame with columns: `id`, `x`, `y`. Participants were asked how many standard sized alcoholic beverages they would buy at various prices.

apt_full	<i>Full alcohol purchase task dataset</i>
----------	---

Description

A larger dataset containing alcohol purchase task data with demographic covariates. Suitable for testing hurdle models and mixed-effects models with covariates.

Usage

```
apt_full
```

Format

A data frame with 18,700 rows and 8 columns:

id Unique participant identifier (1-1100)
gender Participant gender (Male/Female)
age Participant age in years
binges Number of binge drinking episodes
totdrinks Total number of drinks consumed
tothours Total hours spent drinking
x Price point for the purchase task
y Number of drinks participant would purchase at price x

Examples

```
data(apt_full)
# Use a subset for quick demonstration
apt_sub <- apt_full[apt_full$id %in% unique(apt_full$id)[1:20], ]
fit <- fit_demand_hurdle(apt_sub, y_var = "y", x_var = "x", id_var = "id")
summary(fit)
```

augment.beezdemand_fixed	<i>Augment a beezdemand_fixed Model with Fitted Values and Residuals</i>
--------------------------	--

Description

Returns the original data with fitted values and residuals from individual demand curve fits. This enables easy model diagnostics and visualization with the tidyverse.

Usage

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

Arguments

x	An object of class beeزدemand_fixed.
newdata	Optional data frame of new data for prediction. If NULL, uses the original data from the model.
...	Additional arguments (currently unused).

Details

For "hs" equation models where fitting is done on the log10 scale, fitted values are back-transformed to the natural scale.

Value

A tibble containing the original data plus:

.fitted Fitted demand values on the response scale

.resid Residuals (observed - fitted)

Examples

```
data(apt)
fit <- fit_demand_fixed(apt, y_var = "y", x_var = "x", id_var = "id")
augmented <- augment(fit)

# Plot residuals by subject
library(ggplot2)
ggplot(augmented, aes(x = .fitted, y = .resid)) +
  geom_point(alpha = 0.5) +
  facet_wrap(~id) +
  geom_hline(yintercept = 0, linetype = "dashed")
```

```
augment.beeزدemand_hurdle
```

Augment a beeزدemand_hurdle Model with Fitted Values and Residuals

Description

Returns the original data with fitted values, residuals, and predictions from a hurdle demand model, in the format expected by tidyverse plotting and diagnostic tools.

Usage

```
## S3 method for class 'beezdemand_hurdle'
augment(x, newdata = NULL, component = c("combined", "continuous"), ...)
```

Arguments

x	An object of class beezdemand_hurdle.
newdata	Optional data frame of new data for prediction. If NULL, uses the original data from the model.
component	Character. Which residuals to compute: "combined" (Default) Randomized quantile residuals (Dunn & Smyth, 1996) that assess both the binary and continuous components simultaneously. If the model is correctly specified, these are exactly $N(0,1)$. "continuous" Log-scale residuals $\log(y) - \mu$ for positive observations only (zeros are NA). Assesses Part II specification. "binary" Not returned as residuals; use the .fitted_prob column and observed binary indicators for calibration diagnostics.
...	Additional arguments (currently unused).

Details**Residual types for hurdle models:**

The hurdle model has two components, each requiring different diagnostic approaches:

- **Continuous residuals** (component = "continuous"): Standard log-scale residuals $\log(y) - \mu$ for observations where $y > 0$. Zeros are excluded (NA). Assesses whether the lognormal conditional distribution is well- specified.
- **Randomized quantile residuals** (component = "combined", default): Following Dunn & Smyth (1996), maps each observation through the fitted hurdle CDF and then the standard normal quantile function. If the model is correctly specified, these residuals are exactly $N(0,1)$ regardless of which component generated the observation. For zeros, a uniform random variate within $[0, P(\text{zero})]$ breaks ties that would otherwise create a spike in the QQ plot.

Value

A tibble containing the original data plus:

.fitted Fitted demand values (natural scale)

.fitted_link Fitted values on log scale (Part II mean)

.fitted_prob Predicted probability of consumption ($1 - P(\text{zero})$)

.resid Residuals (type depends on component; see above)

.resid_response Residuals on response scale ($y - \text{.fitted}$)

Examples

```
data(apt)
fit <- fit_demand_hurdle(apt, y_var = "y", x_var = "x", id_var = "id")
augmented <- augment(fit)

# Plot residuals
library(ggplot2)
ggplot(augmented, aes(x = .fitted, y = .resid)) +
  geom_point(alpha = 0.5) +
  geom_hline(yintercept = 0, linetype = "dashed")
```

```
augment.beezdemand_nlme
```

Augment a beezdemand_nlme Model with Fitted Values and Residuals

Description

Returns the original data with fitted values and residuals from a nonlinear mixed-effects demand model. This enables easy model diagnostics and visualization with the tidyverse.

Usage

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

Arguments

<code>x</code>	An object of class <code>beezdemand_nlme</code> .
<code>newdata</code>	Optional data frame of new data for prediction. If <code>NULL</code> , uses the original data from the model.
<code>...</code>	Additional arguments (currently unused).

Details

The fitted values and residuals are on the same scale as the response variable used in the model. For `equation_form = "zben"`, this is the LL4-transformed scale. For `equation_form = "simplified"` or `"exponentiated"`, this is the natural consumption scale.

To back-transform predictions to the natural scale for "zben" models, use: `ll4_inv(augmented$.fitted)`

Value

A tibble containing the original data plus:

.fitted Fitted values on the model scale (may be transformed, e.g., LL4)

.resid Residuals on the model scale

.fixed Fitted values from fixed effects only (population-level)

Examples

```
data(ko)
fit <- fit_demand_mixed(ko, y_var = "y_ll4", x_var = "x",
                        id_var = "monkey", factors = "dose", equation_form = "zben")
augmented <- augment(fit)

# Plot residuals
library(ggplot2)
ggplot(augmented, aes(x = .fitted, y = .resid)) +
  geom_point(alpha = 0.5) +
  geom_hline(yintercept = 0, linetype = "dashed")
```

```
augment.beezdemand_tmb
```

Augment a beezdemand_tmb Model

Description

Augment a beezdemand_tmb Model

Usage

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

Arguments

x	A beezdemand_tmb object.
newdata	Optional data frame.
...	Additional arguments.

Value

A tibble with original data plus .fitted, .resid, and .std_resid columns. Residuals are computed on the model's native scale (log scale for "exponential", natural/LL4 scale for others) to match the C++ likelihood.

Examples

```
data(apt)
fit <- fit_demand_tmb(apt, equation = "exponential", verbose = 0)
head(augment(fit))
```

`augment.cp_model_lm` *Augment a Cross-Price Demand Model (Linear)*

Description

Augment a Cross-Price Demand Model (Linear)

Usage

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

Arguments

`x` A `cp_model_lm` object.
`...` Additional arguments (unused).

Value

A tibble with the original modelling data and added `.fitted` and `.resid` columns.

`augment.cp_model_lmer` *Augment a Cross-Price Demand Model (Mixed-Effects)*

Description

Augment a Cross-Price Demand Model (Mixed-Effects)

Usage

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

Arguments

`x` A `cp_model_lmer` object.
`...` Additional arguments (unused).

Value

A tibble with the original modelling data and added `.fitted`, `.resid`, and `.fixed` (population-level prediction with random effects set to zero) columns.

`augment.cp_model_nls` *Augment a Cross-Price Demand Model (Nonlinear)*

Description

Augment a Cross-Price Demand Model (Nonlinear)

Usage

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

Arguments

`x` A `cp_model_nls` object.
`...` Additional arguments (unused).

Value

A tibble with the original modelling data and added `.fitted` and `.resid` columns. Both are on the model's response scale: raw `y` for `equation = "exponentiated" / "additive"`, and `log10(y)` for `equation = "exponential"` (which is fit to `log10(y)`).

`beezdemand_descriptive_methods`
S3 Methods for beezdemand_descriptive Objects

Description

Methods for printing, summarizing, and visualizing objects of class `beezdemand_descriptive` created by `get_descriptive_summary()`.

Usage

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

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

## S3 method for class 'beezdemand_descriptive'
plot(x, x_trans = "identity", y_trans = "identity", show_zeros = FALSE, ...)
```

Arguments

x, object	A beezdemand_descriptive object
...	Additional arguments (currently unused)
x_trans	Character string specifying x-axis transformation. Options: "identity" (default), "log10", "log", "sqrt". See scales::transform_log10() etc.
y_trans	Character string specifying y-axis transformation. Options: "identity" (default), "log10", "log", "sqrt", "pseudo_log" (signed log).
show_zeros	Logical indicating whether to show proportion of zeros as labels on the boxplot (default: FALSE)

Details**Print Method:**

Displays a compact summary showing the number of subjects and prices analyzed, plus a preview of the statistics table.

Summary Method:

Provides extended information including:

- Data summary (subjects, prices analyzed)
- Distribution of means across prices (min, median, max)
- Proportion of zeros by price (range)
- Missing data summary

Plot Method:

Creates a boxplot showing the distribution of consumption at each price point. Features:

- Red cross markers indicate means
- Boxes show median and quartiles
- Whiskers extend to 1.5 * IQR
- Supports axis transformations (log, sqrt, etc.)
- Uses modern beezdemand styling via [theme_apas\(\)](#)

Value

- `print()` - Returns the object invisibly (called for side effects)
- `summary()` - Returns a list with extended summary information
- `plot()` - Returns a ggplot2 object

See Also

[get_descriptive_summary\(\)](#)

Examples

```

data(apt, package = "beezdemand")
desc <- get_descriptive_summary(apt)

# Print compact summary
print(desc)

# Extended summary
summary(desc)

# Default boxplot
plot(desc)

# With log-transformed y-axis
plot(desc, y_trans = "log10")

# With pseudo-log y-axis (defined at zero)
plot(desc, y_trans = "pseudo_log")

```

beezdemand_empirical_methods

S3 Methods for beezdemand_empirical Objects

Description

Methods for printing, summarizing, and visualizing objects of class `beezdemand_empirical` created by `get_empirical_measures()`.

Usage

```

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

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

## S3 method for class 'beezdemand_empirical'
plot(x, type = "histogram", ...)

```

Arguments

<code>x, object</code>	A <code>beezdemand_empirical</code> object
<code>...</code>	Additional arguments passed to plotting functions
<code>type</code>	Character string specifying plot type. Options: "histogram" (default) - Faceted histograms showing distribution of each measure

- "matrix" - Scatterplot matrix showing pairwise relationships between measures

Details

Print Method:

Displays a compact summary showing the number of subjects analyzed and a preview of the empirical measures table.

Summary Method:

Provides extended information including:

- Data summary (subjects, zero consumption patterns, completeness)
- Descriptive statistics for each empirical measure (min, median, mean, max, SD)
- Missing data patterns

Plot Method:

Creates visualizations of empirical measures across subjects.

Histogram type (default):

- Six-panel faceted plot showing distribution of each measure
- Helps identify central tendencies and outliers

Matrix type:

- Scatterplot matrix (pairs plot) showing relationships between measures
- Useful for identifying correlated demand metrics
- Lower triangle: scatterplots with smoothed trend lines
- Diagonal: density plots
- Upper triangle: correlation coefficients

Value

- `print()` - Returns the object invisibly (called for side effects)
- `summary()` - Returns a list with extended summary information
- `plot()` - Returns a ggplot2 object

See Also

`get_empirical_measures()`

Examples

```
data(apt, package = "beezdemand")
emp <- get_empirical_measures(apt)

# Print compact summary
print(emp)

# Extended summary
summary(emp)
```

```
# Histogram of measure distributions
plot(emp)

# Scatterplot matrix
plot(emp, type = "matrix")
```

BIC.beezdemand_hurdle	<i>BIC for Hurdle Demand Model</i>
-----------------------	------------------------------------

Description

BIC for Hurdle Demand Model

Usage

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

Arguments

object A beezdemand_hurdle object.
... Additional arguments (unused).

Value

A numeric BIC value.

boot_demand	<i>Bootstrap Confidence Intervals for Derived Demand Metrics</i>
-------------	--

Description

Computes confidence intervals on derived demand metrics (Pmax, Omax, Qmax, EV, elasticity-at-Pmax) for a TMB mixed-effects demand fit, via a parametric bootstrap. Draws of the fixed-effect parameter vector are taken from the asymptotic Gaussian sampling approximation $N(\hat{\beta}, \hat{\Sigma})$ (the distribution Wald intervals assume; no prior is involved), mapped to per-condition (Q_0, α, k) through the model's fixed-effect design, passed through the canonical Pmax/Omax engine, and summarized by empirical quantiles.

For a factor-expanded fit, one CI is returned per factor cell (e.g. one row per gender level); for an intercept-only fit, one row per statistic with condition = NA. The per-cell point estimate reproduces `calc_group_metrics`(fit, at = cell); the bootstrap supplies only the interval.

Usage

```
boot_demand(
  fit,
  statistics = c("Pmax", "Omax", "EV"),
  method = c("parametric"),
  R = 1000L,
  ci_level = 0.95,
  at = NULL,
  seed = NULL,
  ...
)
```

Arguments

<code>fit</code>	A <code>beezdemand_tmb</code> object. NLME (<code>beezdemand_nlme</code>) and hurdle fits are not supported in this version and raise an informative error.
<code>statistics</code>	Character vector; any of <code>c("Pmax", "Omax", "Qmax", "EV", "elasticity_at_pmax")</code> . Default <code>c("Pmax", "Omax", "EV")</code> .
<code>method</code>	Resampling scheme. Only "parametric" is available in this version (nonparametric subject resampling is planned).
<code>R</code>	Integer number of bootstrap draws; minimum 100, default 1000.
<code>ci_level</code>	Confidence level for the empirical-quantile interval (default 0.95).
<code>at</code>	Optional named list of factor-level filters / continuous-covariate value overrides, with the same shape as the <code>at</code> argument of <code>calc_group_metrics</code> . When <code>NULL</code> (default) all factor cells are enumerated; supplying <code>at</code> conditions to (or filters) the requested cell(s).
<code>seed</code>	Optional integer seed for reproducible draws. The caller's RNG state is left unperturbed.
<code>...</code>	Reserved for future extension; must be empty. Unknown arguments (e.g. a misspelled statistics) raise an error rather than being silently ignored.

Details

The parametric bootstrap is asymptotically equivalent to the delta method but avoids its linearization, so it is preferable for the strongly nonlinear derived metrics (Pmax/Omax via Lambert-W). Draws are fixed-effect-only (population / per-condition metrics); per-subject metric CIs would require random-effect-aware draws and are out of scope for now.

When `k` is estimated, its uncertainty is propagated (the `log_k` column is in the draw matrix); when `k` is fixed, the fixed value is used. The point estimate always uses the point `k`, matching `calc_group_metrics()`.

Note that percentile intervals of a nonlinear transform are not guaranteed to bracket the point estimate; `conf.low <= conf.high` always holds, but `conf.low <= estimate <= conf.high` may not at boundary cases.

Some draws can leave a metric's domain (e.g. Pmax is undefined when a drawn `k` falls below the Lambert-W threshold). Such non-finite draws are excluded from the quantiles; the per-row count

of excluded draws is recorded in `attr(x, "n_nonfinite")` (so the realized draw count is `R` minus that). If *every* draw of a requested metric/condition is non-finite, an error is raised because the interval is undefined.

Value

A tibble with one row per (statistic, condition):

statistic Metric name.

condition Factor-cell label (e.g. "gender=Male"); NA when the fit has no factors.

estimate Point estimate (from the coefficient vector and point `k`).

conf.low, conf.high Empirical-quantile interval bounds.

level The confidence level used.

The bootstrap settings are attached as attributes "method", "R", and "seed", plus "n_nonfinite" (a per-row count of draws excluded as non-finite).

See Also

[calc_group_metrics](#), [fit_demand_tmb](#), [confint.beezdemand_tmb](#)

Examples

```
data(apt, package = "beezdemand")
fit <- fit_demand_tmb(apt, equation = "exponential", verbose = 0)
boot_demand(fit, statistics = c("Pmax", "Omax", "EV"), R = 500, seed = 1)
```

calc_group_metrics	<i>Calculate Group-Level Demand Metrics</i>
--------------------	---

Description

Calculates group-level (population) Omax and Pmax from a fitted hurdle demand model.

Usage

```
calc_group_metrics(object, ...)
```

Arguments

object	A fitted <code>beezdemand_hurdle</code> or <code>beezdemand_tmb</code> object.
...	Method-specific arguments. The <code>beezdemand_tmb</code> method accepts an <code>at</code> argument for explicit conditioning of continuous covariates and factor levels; see calc_group_metrics.beezdemand_tmb .

Value

A named list with group-level Pmax, Omax, and Qmax.

See Also

[calc_omax_pmax](#), [fit_demand_hurdle](#)

Examples

```
data(apt)
fit <- fit_demand_hurdle(apt, y_var = "y", x_var = "x", id_var = "id")
calc_group_metrics(fit)
```

calc_group_metrics.beezdemand_tmb

Calculate Population-Level Demand Metrics for TMB Model

Description

Calculate Population-Level Demand Metrics for TMB Model

Usage

```
## S3 method for class 'beezdemand_tmb'
calc_group_metrics(object, at = NULL, ...)
```

Arguments

object	A beezdemand_tmb object.
at	Named list of factor-level filters or continuous-covariate value overrides (e.g. list(condition = "C1", FTND_z = 0.5)). When NULL (default), continuous covariates are evaluated at their training mean and factors are marginalized across observed levels (equal weights). When supplied, conditions the parameter EMMs to the specified factor levels and/or covariate values before deriving Pmax/Omax. Same shape as the at argument of get_demand_param_emms.beezdemand_tmb and get_demand_comparisons.beezdemand_tmb .
...	Additional arguments (currently unused).

Value

A list with Pmax, Omax, Qmax, elasticity_at_pmax, method, and conditioned_on describing the reference point used. The conditioned_on field reports the actual conditioning applied (covariate values used, factor treatment per factor) so programmatic consumers do not have to re-derive it.

Marginalization order

For derived metrics (Pmax/Omax/Qmax) that depend nonlinearly on Q_0 and α jointly, this function marginalizes parameters first then derives metrics:

1. Compute log- Q_0 and log- α EMMs at each cell of the reference grid produced by `.tmb_build_emm_ref_grid()`.
2. Marginalize each parameter across factor cells with equal weights (matches the `emmeans` default).
3. Derive Pmax/Omax/Qmax from the marginalized log-parameters at the user-supplied (or training-mean default) covariate point.

The result is "metrics evaluated at the average parameter values" rather than "average metrics across cells". The two answers differ for nonlinear transforms. The convention matches the parameter-level marginalization used by `get_demand_param_emms()`.

Marginalisation policy (TMB vs NLME)

This method averages the log-scale linear predictors with equal weight over the factor cells observed in the fitting data and then exponentiates. The NLME method (`calc_group_metrics.beezdemand_nlme()`) follows the same policy: per-cell EMMs from `emmeans`, rows for unobserved cells dropped, equal-weight geometric mean over the remaining cells. Cells of the full factorial crossing with no subjects are never averaged over by either backend. The two agree for any design fit in log space and can differ only when the NLME fit uses `param_space = "natural"`. In both backends continuous covariates are held at the training mean unless it supplies a single value; a multi-value continuous at entry warns and uses its first value.

See Also

[fit_demand_tmb](#), [get_demand_param_emms.beezdemand_tmb](#)

Examples

```
data(apt)
fit <- fit_demand_tmb(apt, equation = "exponential", verbose = 0)
calc_group_metrics(fit)
# Conditioned at a specific covariate value:
# calc_group_metrics(fit_with_cov, at = list(FTND_z = 1))
```

calc_observed_pmax_omax

Calculate Observed Pmax/Omax Grouped by ID

Description

Calculate Observed Pmax/Omax Grouped by ID

Usage

```
calc_observed_pmax_omax(  
  data,  
  id_var = "id",  
  price_var = "x",  
  consumption_var = "y"  
)
```

Arguments

data	Data frame with id, price, and consumption columns
id_var	Name of ID column
price_var	Name of price column
consumption_var	Name of consumption column

Value

Data frame with observed pmax/omax for each subject

Examples

```
data(apt, package = "beezdemand")  
calc_observed_pmax_omax(apt, id_var = "id", price_var = "x", consumption_var = "y")
```

calc_omax_pmax	<i>Calculate Omax and Pmax for Demand Curves</i>
----------------	--

Description

Calculates the maximum expenditure (Omax) and the price at maximum expenditure (Pmax) for the exponential demand model used in the two-part hurdle model.

Usage

```
calc_omax_pmax(Q0, k, alpha, price_range = NULL)
```

Arguments

Q0	Intensity parameter (consumption at price 0).
k	Scaling parameter for the exponential decay.
alpha	Elasticity parameter (rate of decay).
price_range	Numeric vector of length 2 specifying the price range to search for Pmax. Default is NULL, which uses an adaptive range based on alpha (approximately 0 to 10/alpha).

Details

For the demand function:

$$Q(p) = Q_0 \cdot \exp(k \cdot (\exp(-\alpha \cdot p) - 1))$$

Expenditure is $E(p) = p \cdot Q(p)$. Omax is the maximum of $E(p)$ and Pmax is the price at which this maximum occurs. These are found numerically.

The search range is automatically adjusted based on alpha to ensure the maximum is found. For small alpha values, Pmax can be quite large.

Value

A named list with:

Pmax Price at maximum expenditure

Omax Maximum expenditure (price * quantity)

Qmax Quantity at Pmax

See Also

[calc_group_metrics](#), [fit_demand_hurdle](#)

Examples

```
# Calculate for group-level parameters
calc_omax_pmax(Q0 = 10, k = 2, alpha = 0.5)

# With k > e (~2.718), a strict interior maximum exists
calc_omax_pmax(Q0 = 10, k = 3, alpha = 0.5)
```

calculate_amplitude_persistence

Calculate Amplitude and Persistence

Description

Calculates Amplitude and Persistence latent factors from demand metrics for various beezdemand model objects.

Usage

```
calculate_amplitude_persistence(
  fit,
  amplitude = c("Intensity", "Q0d", "Q0"),
  persistence = c("BP0", "Pmaxe", "Omaxe", "Alpha"),
  use_inv_alpha = TRUE,
  strict = TRUE,
  min_persistence_components = 2L,
  empirical_y_var = NULL,
  basis_means = NULL,
  basis_sds = NULL,
  ...
)
```

Arguments

<code>fit</code>	An object of class <code>beezdemand_fixed</code> , <code>beezdemand_hurdle</code> , <code>beezdemand_nlme</code> , or a <code>data.frame</code> .
<code>amplitude</code>	Character vector of column names to consider for the Amplitude factor. The function will use the first column found in the data. Default is <code>c("Intensity", "Q0d", "Q0")</code> .
<code>persistence</code>	Character vector of column names to include in the Persistence factor. Default is <code>c("BP0", "Pmaxe", "Omaxe", "Alpha")</code> . All found columns will be used.
<code>use_inv_alpha</code>	Logical. If "Alpha" (or a variation) is present in <code>persistence</code> , should it be inverted ($1/\text{Alpha}$) before standardization? Default is <code>TRUE</code> .
<code>strict</code>	Logical. If <code>TRUE</code> (default), missing metrics, duplicated ids, and other data integrity problems produce errors instead of warnings.
<code>min_persistence_components</code>	Integer. Minimum number of non-missing standardized persistence components required to compute Persistence for a given id. If fewer are available, Persistence is set to NA. Default is 2.
<code>empirical_y_var</code>	For <code>beezdemand_nlme</code> objects, optional column name in <code>fit\$data</code> to use when computing empirical indices (e.g., <code>BP0</code>). This is important when the fitted <code>y_var</code> is transformed (e.g., <code>log10</code>). If <code>NULL</code> , the method will attempt to choose a sensible default and may error in <code>strict</code> mode if ambiguous.
<code>basis_means</code>	Optional named numeric vector of means to use for Z-score standardization. Names must match the columns used (e.g., <code>c(Intensity = 10, BP0 = 5)</code>). If <code>NULL</code> (default), the sample means are used.
<code>basis_sds</code>	Optional named numeric vector of standard deviations to use for Z-score standardization. Names must match the columns used. If <code>NULL</code> (default), the sample SDs are used.
<code>...</code>	Additional arguments passed to methods.

Details

This function calculates Amplitude and Persistence by:

1. Extracting the relevant demand metrics from the `fit` object.
2. Resolving requested metric columns (case-insensitive, with limited synonym support for common beezdemand outputs).
3. Inverting Alpha if requested ($1/\text{Alpha}$).
4. Standardizing (Z-scoring) the metrics. If `basis_means` and `basis_sds` are provided, they are used; otherwise, the current sample's statistics are used.
5. Aggregating the Z-scores into the two latent factors.

Amplitude is defined by the variable specified in `amplitude` (typically Intensity/Q0). **Persistence** is defined as the mean of the standardized values of the variables specified in `persistence` (typically Breakpoint, Pmax, Omax, and $1/\text{Alpha}$).

Value

A data frame with the original ID and calculated Amplitude and Persistence scores, along with the standardized (Z-scored) constituent metrics.

Models

- **beezdemand_fixed**: Extracts metrics from `fit$results`.
- **beezdemand_hurdle**: Extracts metrics from `fit$subject_pars`.
- **beezdemand_nlme**: Calculates subject-specific parameters from fixed and random effects. Parameters Q0 and Alpha are assumed to be on log10 scale for zben and simplified equations and are converted to linear scale. Omax and Pmax are calculated empirically from predictions. Breakpoint is calculated empirically from the raw data.

Examples

```
data(apt, package = "beezdemand")
fit <- FitCurves(apt, "hs", k = "share")
calculate_amplitude_persistence(fit)
```

cannabisCigarettes	<i>Cannabis/cigarette cross-price responses</i>
--------------------	---

Description

Cross-price style data with cannabis and cigarette context.

Usage

```
cannabisCigarettes
```

Format

A data frame with columns including: id, x, y, commodity, and auxiliary fields (Q1035, CigPrice, CanPrice, variable, value).

Examples

```
# data(cannabisCigarettes)
```

ChangeData	<i>ChangeData</i>
------------	-------------------

Description

Changes demand data

Usage

```
ChangeData(  
  dat,  
  nrepl = 1,  
  replnum = 0.01,  
  rem0 = FALSE,  
  remq0e = FALSE,  
  replfree = NULL,  
  xcol = "x",  
  ycol = "y",  
  idcol = "id"  
)
```

Arguments

dat	A long form dataframe
nrepl	Number of zeros to replace with replacement value (replnum). Can accept either a number or "all" if all zeros should be replaced. Default is to replace the first zero only
replnum	Value to replace zeros. Default is .01
rem0	If TRUE, removes all 0s in consumption data prior to analysis. Default value is FALSE
remq0e	If TRUE, removes consumption and price where price == 0. Default value is FALSE
replfree	Optionally replaces price == 0 with specified value.
xcol	Column name in dataframe that signifies x values (usually price or the IV)
ycol	Column name in dataframe that signifies y values (usually consumption or the DV)
idcol	Column name in dataframe that signifies identifying id grouping

Details

Change demand data in various ways. Ways include replacing any number of 0 values with a replacement number (or remove them completely), removing price and consumption at free, replacing free with some number. This will soon replace ReplaceZeros and certain arguments in FitCurves.

Value

Long form dataframe resembling the originally provided dataframe

Author(s)

Brent Kaplan bkaplan.ku@gmail.com

Examples

```
## Change just the first instance of 0 within each unique value of id with .1
ChangeData(apt, nrepl = 1, replnum = .1)
```

check_demand_model	<i>Check Demand Model Diagnostics</i>
--------------------	---------------------------------------

Description

Performs diagnostic checks on fitted demand models, returning information about convergence, boundary conditions, and residual patterns.

Usage

```
check_demand_model(object, ...)

## S3 method for class 'beezdemand_hurdle'
check_demand_model(object, ...)

## S3 method for class 'beezdemand_nlme'
check_demand_model(object, ...)

## S3 method for class 'beezdemand_fixed'
check_demand_model(object, ...)

## S3 method for class 'beezdemand_tmb'
check_demand_model(object, ...)
```

Arguments

object	A fitted model object of class beezdemand_hurdle, beezdemand_nlme, or beezdemand_fixed.
...	Additional arguments passed to methods.

Details

The function checks for:

- Convergence status and optimization messages
- Parameters at or near boundaries
- Residual patterns (heteroscedasticity, outliers)
- Random effect variance estimates near zero
- For `fit_demand_tmb()` fits with `estimate_k = TRUE`, whether the fit shows the signature of an unidentified k : an implausible k , a decay exponent $\alpha * Q_0 * \text{price}$ that never leaves its linear regime over the observed prices, a degenerate α , or $\log_k$ resting on a user-supplied bound. This is a screen, not a formal test: no flag means nothing was detected, not that k is identified.
- Correlation matrices near singularity

Value

An object of class `beezdemand_diagnostics` containing:

convergence List with convergence status and messages

boundary List with boundary condition warnings. For `beezdemand_tmb` fits it also carries `k_identification`: NULL unless k was estimated as a free parameter, otherwise the screen described in Details.

residuals Summary statistics for residuals

random_effects Summary of random effects (if applicable). `variances` holds random-effect variances: on the $\log_{10}$ scale for `beezdemand_tmb` fits, and as reported by `nlme::VarCorr()` for `beezdemand_nlme` fits. `beezdemand_tmb` fits also carry `sd_log10` (the corresponding standard deviations, matching `summary(fit)$variance_components`) and `sd_internal_log` (the raw natural-log-scale standard deviations used by the near-zero degeneracy check).

issues Character vector of identified issues

recommendations Character vector of recommendations

Note

This function is named `check_demand_model()` to avoid potential conflicts with `performance::check_model()` from the `performance` package.

See Also

`plot_residuals()`, `plot_qq()`

Examples

```
data(apt)
fit <- fit_demand_hurdle(apt, y_var = "y", x_var = "x", id_var = "id")
diagnostics <- check_demand_model(fit)
print(diagnostics)
```

check_systematic_cp	<i>Check Cross-Price Data for Unsystematic Responding</i>
---------------------	---

Description

Modern interface for screening cross-price data with standardized output vocabulary aligned with `check_systematic_demand()`.

Usage

```
check_systematic_cp(
  data,
  trend_threshold = 0.025,
  bounce_threshold_down = 0.1,
  bounce_threshold_up = 0.1,
  bounce_threshold_none = 0.1,
  consecutive_zeros = 2,
  consecutive_nonzeros = 2,
  expected_down = FALSE,
  x_var = "x",
  y_var = "y",
  id_var = "id",
  by = NULL
)
```

Arguments

<code>data</code>	Data frame with columns: <code>id</code> (optional), <code>x</code> (price), <code>y</code> (consumption).
<code>trend_threshold</code>	Numeric. Threshold for trend detection. Default 0.025.
<code>bounce_threshold_down</code>	Numeric. Bounce threshold for upward trends. Default 0.1.
<code>bounce_threshold_up</code>	Numeric. Bounce threshold for downward trends. Default 0.1.
<code>bounce_threshold_none</code>	Numeric. Bounce threshold when no trend. Default 0.1.
<code>consecutive_zeros</code>	Integer. Zeros for reversal detection. Default 2.
<code>consecutive_nonzeros</code>	Integer. Non-zeros for return detection. Default 2.
<code>expected_down</code>	Logical. Suppress reversal detection if TRUE. Default FALSE.
<code>x_var</code>	Character. Name of the price column. Default "x".
<code>y_var</code>	Character. Name of the consumption column. Default "y".
<code>id_var</code>	Character. Name of the subject identifier column. Default "id".
<code>by</code>	Optional character vector of column names to group by. When supplied, the check is run separately within each unique combination of the <code>by</code> columns. Group columns are prepended to <code>\$results</code> . Default NULL (no grouping).

Details

If the data contains an `id` column (or column specified by `id_var`), each unique ID is checked separately. Otherwise, the entire dataset is treated as a single pattern.

For cross-price data, the wrapper preserves the legacy meaning of `check_unsystematic_cp()`:

- `trend_direction` and `bounce_direction` are taken directly from the legacy function outputs.
- `trend_pass` is set to NA because cross-price systematicity does not use a separate trend “pass/fail” criterion in the same way as purchase-task screening; instead, trend classification determines which bounce rule applies.
- `bounce_stat` is reported as the proportion relevant to the legacy bounce rule for the detected `trend_direction` (or expected_down case), computed from the legacy bounce counts and the number of price steps.

Value

An object of class `beezdemand_systematicity` with the same structure as `check_systematic_demand()`, with `type = "cp"`.

Examples

```
data(etm)
check <- check_systematic_cp(etm)
```

check_systematic_demand

Check Demand Data for Unsystematic Responding

Description

Modern interface for screening purchase task data using Stein et al. (2015) criteria. Returns a structured object with standardized output vocabulary that is consistent with `check_systematic_cp()`.

Usage

```
check_systematic_demand(
  data,
  trend_threshold = 0.025,
  bounce_threshold = 0.1,
  max_reversals = 0,
  consecutive_zeros = 2,
  x_var = "x",
  y_var = "y",
  id_var = "id",
  by = NULL
)
```

Arguments

<code>data</code>	Data frame in long format with columns: <code>id</code> , <code>x</code> (price), <code>y</code> (consumption).
<code>trend_threshold</code>	Numeric. Threshold for trend detection (log-log slope). Default 0.025.
<code>bounce_threshold</code>	Numeric. Threshold for bounce proportion. Default 0.10.
<code>max_reversals</code>	Integer. Maximum allowed reversals from zero. Default 0.
<code>consecutive_zeros</code>	Integer. Consecutive zeros required for reversal detection. Default 2 (per Stein et al. 2015).
<code>x_var</code>	Character. Name of the price column. Default "x".
<code>y_var</code>	Character. Name of the consumption column. Default "y".
<code>id_var</code>	Character. Name of the subject identifier column. Default "id".
<code>by</code>	Optional character vector of column names to group by. When supplied, the check is run separately within each unique combination of the <code>by</code> columns. Group columns are prepended to <code>\$results</code> . Default NULL (no grouping).

Details

The `results` tibble contains standardized columns for both demand and cross-price systematicity checks:

id	Subject identifier
type	"demand" for this function
trend_stat	DeltaQ statistic (log-log slope)
trend_threshold	Threshold used
trend_direction	"down", "up", or "none"
trend_pass	Logical: passed trend criterion
bounce_stat	Bounce proportion
bounce_threshold	Threshold used
bounce_direction	"significant" or "none"
bounce_pass	Logical: passed bounce criterion
reversals	Count of reversals from zero
reversals_pass	Logical: passed reversals criterion
returns	NA for demand (CP-specific)
n_positive	Count of positive values
systematic	Logical: passed all criteria

Value

An object of class `beezdemand_systematicity` with components:

results Tibble with one row per subject containing systematicity metrics

type "demand"

call The original function call

n_total Total number of subjects

n_systematic Number of subjects passing all criteria

n_unsystematic Number of subjects failing at least one criterion

Examples

```
data(apt)
check <- check_systematic_demand(apt)
print(check)
summary(check)
tidy(check)

# Grouped check: results include group column
data(apt_full)
check_g <- check_systematic_demand(apt_full, by = "gender")
check_g$results
```

`check_unsystematic_cp` *Check for Unsystematic Patterns in Cross-Price Data*

Description

Analyzes whether consumption data shows systematic trends or unsystematic patterns ("bounces") with respect to price. Includes detection of zero-value reversal/return sequences and allows flexible output based on the level of detail requested. See Rzeszutek et al. (in press) for more details.

Usage

```
check_unsystematic_cp(
  data,
  delta_threshold = 0.025,
  bounce_down_threshold = 0.1,
  bounce_up_threshold = 0.1,
  bounce_none_threshold = 0.1,
  rev_zeroes = 2,
  ret_nums = 2,
  expected_down = FALSE,
  verbose = FALSE,
  detailed = FALSE
)
```

Arguments

<code>data</code>	A data frame with columns 'x' and 'y', where 'x' is price and 'y' is consumption.
<code>delta_threshold</code>	Numeric. Threshold for detecting log-scale trends (default 0.025).
<code>bounce_down_threshold</code>	Numeric. Minimum downward bounce proportion to count as significant in upward trends.
<code>bounce_up_threshold</code>	Numeric. Minimum upward bounce proportion to count as significant in downward trends.
<code>bounce_none_threshold</code>	Numeric. Minimum bounce proportion to count as significant in no-trend cases.
<code>rev_zeroes</code>	Integer. Length of zero sequences to detect reversals (default 2).
<code>ret_nums</code>	Integer. Length of non-zero sequences to detect returns (default 2).
<code>expected_down</code>	Logical. If TRUE, suppress reversal detection.
<code>verbose</code>	Logical. If TRUE, print intermediate values (default FALSE).
<code>detailed</code>	Logical. If TRUE, return additional columns including all trend/bounce flags.

Value

A data frame of class `cp_unsystematic` with core results:

delta_direction Character: 'down', 'up', or 'none'.

bounce_direction Character: 'up', 'down', 'significant', or 'none'.

bounce_any Logical. TRUE if any bounce pattern detected.

bounce_above Integer. Number of upward changes meeting threshold.

bounce_below Integer. Number of downward changes meeting threshold.

reversals Integer. Detected reversals from 0 to non-0.

returns Integer. Detected returns from non-0 to 0.

If `detailed = TRUE`, returns additional columns:

delta_down Logical. Significant downward trend.

delta_up Logical. Significant upward trend.

delta_none Logical. No significant trend.

bounce_up Logical. Significant bounce up in a downward trend.

bounce_down Logical. Significant bounce down in an upward trend.

bounce_none Logical. Significant bounces in no-trend data.

Examples

```
x_seq <- 10^(seq(-2, 2, length.out = 10))
pattern <- data.frame(x = x_seq, y = c(10, 5, 10, 9, 10, 13, 10, 10, 7, 9))
check_unsystematic_cp(pattern)
```

CheckCols	<i>Check Column Names</i>
-----------	---------------------------

Description

Checks to ensure column names are specified

Usage

```
CheckCols(dat, xcol, ycol, idcol, groupcol = NULL)
```

Arguments

dat	Dataframe
xcol	Name of x column
ycol	Name of y column
idcol	Name of id column
groupcol	Name of group column

Details

Check column names

Value

Dataframe

Author(s)

Brent Kaplan bkaplan.ku@gmail.com

Examples

```
dat <- data.frame(price = 1:5, quantity = c(10, 8, 5, 2, 0), subj = rep(1, 5))
CheckCols(dat, xcol = "price", ycol = "quantity", idcol = "subj")
```

CheckUnsystematic

Systematic Purchase Task Data Checker

Description

Applies Stein, Koffarnus, Snider, Quisenberry, & Bickel's (2015) criteria for identification of non-systematic purchase task data.

Usage

```
CheckUnsystematic(dat, deltaq = 0.025, bounce = 0.1, reversals = 0, ncons0 = 2)
```

Arguments

dat	Dataframe in long form. Columns are id, x, y.
deltaq	Numeric vector of length equal to one. The criterion by which the relative change in quantity purchased will be compared. Relative changes in quantity purchased below this criterion will be flagged. Default value is 0.025.
bounce	Numeric vector of length equal to one. The criterion by which the number of price-to-price increases in consumption that exceed 25% of initial consumption at the lowest price, expressed relative to the total number of price increments, will be compared. The relative number of price-to-price increases above this criterion will be flagged. Default value is 0.10.
reversals	Numeric vector of length equal to one. The criterion by which the number of reversals from number of consecutive (see ncons0) 0s will be compared. Number of reversals above this criterion will be flagged. Default value is 0.
ncons0	Numer of consecutive 0s prior to a positive value is used to flag for a reversal. Value can be either 1 (relatively more conservative) or 2 (default; as recommended by Stein et al., (2015)).

Details

This function applies the 3 criteria proposed by Stein et al., (2015) for identification of nonsystematic purchase task data. The three criteria include trend (deltaq), bounce, and reversals from 0. Also reports number of positive consumption values.

Value

Dataframe

Author(s)

Brent Kaplan bkaplan.ku@gmail.com

Examples

```
## Using all default values
CheckUnsystematic(apt, deltaq = 0.025, bounce = 0.10, reversals = 0, ncons0 = 2)
## Specifying just 1 zero to flag as reversal
CheckUnsystematic(apt, deltaq = 0.025, bounce = 0.10, reversals = 0, ncons0 = 1)
```

coef-methods

*Extract Coefficients from Cross-Price Demand Models***Description**

Methods to extract coefficients from various cross-price demand model objects.

Usage

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

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

## S3 method for class 'cp_model_lmer'
coef(object, fixed_only = FALSE, combine = TRUE, ...)
```

Arguments

object	A cp_model_lm object
...	Additional arguments (not used).
fixed_only	Logical; if TRUE, returns only fixed effects. Default is FALSE.
combine	Logical; if TRUE and fixed_only=FALSE, returns fixed + random effects combined. Default is TRUE.

Value

Named vector of coefficients

A named numeric vector of model coefficients.

Functions

- `coef(cp_model_nls)`: Extract coefficients from a nonlinear cross-price model
- `coef(cp_model_lm)`: Extract coefficients from a linear cross-price model
- `coef(cp_model_lmer)`: Extract coefficients from a mixed-effects cross-price model

coef.beezdemand_fixed *Extract Coefficients from Fixed-Effect Demand Fit*

Description

Extract Coefficients from Fixed-Effect Demand Fit

Usage

```
## S3 method for class 'beezdemand_fixed'
coef(object, report_space = c("internal", "natural", "log10"), ...)
```

Arguments

object	A beezdemand_fixed object.
report_space	One of "internal", "natural", or "log10". Default "internal".
...	Unused.

Value

A tibble with columns id, term, estimate, estimate_scale, term_display.

coef.beezdemand_fixed_grouped
Extract Coefficients from Grouped Fixed-Effect Demand Fit

Description

Extract Coefficients from Grouped Fixed-Effect Demand Fit

Usage

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

Arguments

object	A beezdemand_fixed_grouped object
...	Additional arguments passed to coef.beezdemand_fixed()

Value

A tibble with group columns prepended

coef.beezdemand_hurdle

Extract Coefficients from Hurdle Demand Model

Description

Extract Coefficients from Hurdle Demand Model

Usage

```
## S3 method for class 'beezeemand_hurdle'
coef(object, report_space = c("natural", "log10", "internal"), ...)
```

Arguments

object	An object of class beezdemand_hurdle.
report_space	Character. One of "natural" (default), "log10", or "internal". Default is "natural" for consistency with tidy().
...	Additional arguments (currently unused).

Value

Named numeric vector of fixed effect coefficients.

coef.beezeemand_nlme

Extract Coefficients from a beezdemand_nlme Model

Description

Provides methods to extract fixed effects, random effects, or subject-specific (combined fixed + random) coefficients from a beezdemand_nlme object. This is an S3 method for the generic coef function.

Usage

```
## S3 method for class 'beezeemand_nlme'
coef(
  object,
  type = "combined",
  report_space = c("internal", "natural", "log10"),
  ...
)
```

Arguments

object	A beezdemand_nlme object.
type	Character, type of coefficients to extract. One of: • "fixed": Returns only fixed effects (equivalent to <code>fixef(object)</code>). • "random": Returns only random effects (equivalent to <code>ranef(object)</code>). • "combined" (default): Returns subject-specific coefficients, where each subject's coefficient is the sum of the corresponding fixed effect and that subject's random effect deviation. This is equivalent to what <code>stats::coef()</code> on an nlme object returns.
report_space	Character. One of "internal" (default), "natural", or "log10".
...	Additional arguments passed to the underlying nlme coefficient extraction functions (<code>nlme::fixef()</code> , <code>nlme::ranef()</code> , or <code>stats::coef.nlme()</code>).

Value

Depending on type:

- type="fixed": A named numeric vector of fixed-effect coefficients.
- type="random": A data frame (or list of data frames if multiple levels of grouping) of random effects, as returned by `ranef.nlme()`.
- type="combined": A data frame where rows are subjects (from `id_var`) and columns are the Q0 and alpha parameters, representing subject-specific estimates (on the log10 scale).

See Also

[fixef.beezdemand_nlme](#), [ranef.beezdemand_nlme](#)

Examples

```
data(ko)
fit <- fit_demand_mixed(ko, y_var = "y_ll4", x_var = "x",
                        id_var = "monkey", equation_form = "zben")
coef(fit, type = "fixed")
coef(fit, type = "random")
coef(fit, type = "combined")
```

coef.beezdemand_tmb	<i>Extract Coefficients from TMB Model</i>
---------------------	--

Description

Extract coefficients from a fitted beezdemand_tmb model. The type argument selects the return shape. The default, "internal", is unchanged: a named numeric vector of the optimizer's flat parameterization (entries include `beta_q0`, `beta_alpha`, `logsigma*`, and any covariance hyperparameters; intercepts are on the log scale because the optimizer works in unconstrained space). This is the numeric-vector escape hatch consumed by tooling such as `car::deltaMethod` and `multcomp::glht`.

Usage

```
## S3 method for class 'beezdemand_tmb'
coef(object, type = c("internal", "subject", "combined", "fixed"), ...)
```

Arguments

object	A beezdemand_tmb object.
type	One of "internal" (default; raw optimizer vector), "subject" or its alias "combined" (per-subject parameter tibble), or "fixed" (one-row tibble of fixed-effect coefficients).
...	Additional arguments (currently unused; supplying report_space is an error).

Details

type = "subject" (alias "combined") returns the per-subject parameter tibble from [get_subject_pars](#) (with expanded = NULL, so within-id factor expansion is auto-detected). This is concept-parity with `coef.beezdemand_nlme(type = "combined")` but not column-identical: it returns resolved per-subject parameters (Q_0 , α , ...), not a per-design-term coefficient matrix. type = "fixed" returns a one-row tibble of the fixed-effect coefficients only (the β_{Q_0} / β_{α} block on the internal parameterization), excluding $\log_k$, $\log\sigma$, and ρ .

Scale conversion is not performed here: supplying report_space through ... is an error. Use [get_subject_pars](#) or [predict.beezdemand_tmb](#) for natural-scale parameters.

Value

For type = "internal", a named numeric vector. For type = "subject"/"combined", a tibble with one row per subject (or one row per subject-by-within-id-factor-level cell when the fit has within-id factor variation). For type = "fixed", a one-row tibble of fixed-effect coefficients.

Examples

```
data(apt)
fit <- fit_demand_tmb(apt, equation = "exponential", verbose = 0)
coef(fit)                # raw optimizer vector (default, "internal")
coef(fit, type = "subject") # per-subject parameter tibble
coef(fit, type = "fixed")  # fixed-effect coefficients
```

Description

Performs a likelihood ratio test comparing two nested hurdle demand models. Typically used to test whether adding the random effect on alpha (c_i) significantly improves model fit (3-RE vs 2-RE models).

Usage

```
compare_hurdle_models(model_full, model_reduced)
```

Arguments

model_full A beezdemand_hurdle object with 3 random effects.
model_reduced A beezdemand_hurdle object with 2 random effects.

Value

Invisibly returns a list with:

lr_stat Likelihood ratio test statistic
df Degrees of freedom
p_value P-value from chi-squared distribution
model_comparison Data frame with model comparison statistics

See Also

[fit_demand_hurdle](#)

Examples

```
data(apt)
fit3 <- fit_demand_hurdle(apt, y_var = "y", x_var = "x", id_var = "id",
  random_effects = c("zeros", "q0", "alpha"))
fit2 <- fit_demand_hurdle(apt, y_var = "y", x_var = "x", id_var = "id",
  random_effects = c("zeros", "q0"))
compare_hurdle_models(fit3, fit2)
```

compare_models

Compare Demand Models

Description

Compare multiple demand models using information criteria and likelihood ratio tests (when applicable). Works with all beezdemand model classes.

Usage

```
compare_models(..., test = c("auto", "lrt", "none"))
```

Arguments

- ... Two or more model objects of class `beezdemand_hurdle`, `beezdemand_nlme`, or `beezdemand_fixed`.
- test** Character; type of statistical test. One of:
- "auto" (default): Use LRT if models are nested and comparable, otherwise IC-only.
 - "lrt": Force likelihood ratio test (requires nested models from same backend).
 - "none": Only report information criteria, no p-value.

Details

Models are compared using AIC and BIC. For models from the same statistical backend (e.g., two hurdle models or two NLME models), likelihood ratio tests can be performed if the models are nested.

When comparing models from different backends (e.g., hurdle vs NLME), only information criteria comparisons are possible since the likelihoods are not directly comparable for LRT purposes.

Backend Compatibility:

Backend 1	Backend 2	LRT Possible?
hurdle	hurdle	Yes (if nested)
nlme	nlme	Yes (if nested)
fixed	fixed	No (no likelihood)
hurdle	nlme	No
hurdle	fixed	No
nlme	fixed	No

Value

An object of class `beezdemand_model_comparison` containing:

comparison Data frame with model fit statistics

test_type Type of test performed

lrt_results LRT results if performed (NULL otherwise)

best_model Index of best model by BIC

notes Character vector of notes/warnings

nesting_verified Logical; always FALSE since nesting is not automatically verified. Users must ensure models are properly nested for valid LRT interpretation.

Statistical Notes

The likelihood ratio test (LRT) assumes that:

1. The models are **nested** (the reduced model is a special case of the full model obtained by constraining parameters).

2. Both models are fit to **identical data**.
3. Under the null hypothesis, the LR statistic follows a chi-square distribution with degrees of freedom equal to the difference in the number of parameters.

Important caveat for mixed-effects models: When variance components are tested at the boundary (e.g., testing whether a random effect variance is zero), the standard chi-square distribution is not appropriate. The correct null distribution is a mixture of chi-squares (Stram & Lee, 1994). The p-values reported here use the standard chi-square approximation, which is conservative (p-values are too large) for boundary tests.

This function does **not** automatically verify that models are nested. Users should ensure models are properly nested before interpreting LRT p-values.

References

Stram, D. O., & Lee, J. W. (1994). Variance components testing in the longitudinal mixed effects model. *Biometrics*, 50(4), 1171-1177.

See Also

[compare_hurdle_models\(\)](#) for the legacy hurdle-specific comparison

Examples

```
data(apt)
fit2 <- fit_demand_hurdle(apt, y_var = "y", x_var = "x", id_var = "id",
  random_effects = c("zeros", "q0"))
fit3 <- fit_demand_hurdle(apt, y_var = "y", x_var = "x", id_var = "id",
  random_effects = c("zeros", "q0", "alpha"))
compare_models(fit2, fit3)
```

confint.beezdemand_fixed

Confidence Intervals for Fixed-Effect Demand Model Parameters

Description

Computes confidence intervals for Q0, alpha, and k parameters from individual demand curve fits. Uses asymptotic normal approximation based on standard errors when available.

Usage

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

Arguments

object	A beezdemand_fixed object from <code>fit_demand_fixed()</code> .
parm	Character vector of parameter names to compute CIs for. Default includes all available parameters.
level	Confidence level (default 0.95).
...	Additional arguments (ignored).

Details

For beezdemand_fixed objects, confidence intervals are computed using the asymptotic normal approximation: estimate $\pm$ z * SE. If standard errors are not available for a parameter, the confidence bounds will be NA.

When the underlying NLS fit objects are available (from detailed = TRUE), this method attempts to use nlstools::confint2() for more accurate profile-based intervals.

Value

A tibble with columns: id, term, estimate, conf.low, conf.high, level.

Examples

```
fit <- fit_demand_fixed(apt, equation = "hs", k = 2)
confint(fit)
confint(fit, level = 0.90)
confint(fit, parm = "Q0")
```

confint.beezdemand_hurdle

Confidence Intervals for Hurdle Demand Model Parameters

Description

Computes confidence intervals for fixed effect parameters from a TMB-based hurdle demand model using the asymptotic normal approximation.

Usage

```
## S3 method for class 'beezdemand_hurdle'
confint(
  object,
  parm = NULL,
  level = 0.95,
  report_space = c("internal", "natural"),
  ...
)
```

Arguments

object	A beezdemand_hurdle object from <code>fit_demand_hurdle()</code> .
parm	Character vector of parameter names to compute CIs for. Default includes all fixed effect parameters.
level	Confidence level (default 0.95).
report_space	Character. Reporting space for parameters: • "internal": parameters on internal/fitting scale (log for Q0, alpha) • "natural": back-transformed to natural scale
...	Additional arguments (ignored).

Details

Confidence intervals are computed using the asymptotic normal approximation based on standard errors from `TMB::sdreport()`. For parameters estimated on the log scale (Q0, alpha, k), intervals can be back-transformed to the natural scale using `report_space = "natural"`.

The transformation uses:

- For log-scale parameters: $\exp(\text{estimate} \pm z * \text{SE})$

Value

A tibble with columns: term, estimate, conf.low, conf.high, level, component, estimate_scale.

Examples

```
data(apt)
fit <- fit_demand_hurdle(apt, y_var = "y", x_var = "x", id_var = "id")
confint(fit)
```

confint.beezdemand_nlme

Confidence Intervals for Mixed-Effects Demand Model Parameters

Description

Computes confidence intervals for fixed effect parameters from an NLME-based mixed-effects demand model.

Usage

```
## S3 method for class 'beezdemand_nlme'
confint(object, parm = NULL, level = 0.95, method = c("wald", "profile"), ...)
```

Arguments

object	A beezdemand_nlme object from <code>fit_demand_mixed()</code> .
parm	Character vector of parameter names to compute CIs for. Default includes all fixed effect parameters.
level	Confidence level (default 0.95).
method	Character. Method for computing intervals: • "wald": Wald-type intervals using asymptotic normality (default, fast) • "profile": Profile likelihood intervals via <code>nlme::intervals()</code> (slower but more accurate for small samples)
...	Additional arguments passed to <code>nlme::intervals()</code> when <code>method = "profile"</code> .

Details

For Wald intervals, confidence bounds are computed as $\text{estimate} \pm z * \text{SE}$ using standard errors from the model summary.

For profile intervals, `nlme::intervals()` is called on the underlying nlme model object. This method provides more accurate intervals but can be computationally intensive for complex models.

Value

A tibble with columns: term, estimate, conf.low, conf.high, level, component.

Examples

```
data(ko)
fit <- fit_demand_mixed(ko, y_var = "y_ll4", x_var = "x",
                        id_var = "monkey", equation_form = "zben")
confint(fit)
```

confint.beezdemand_tmb

Confidence Intervals for TMB Model Parameters

Description

Confidence Intervals for TMB Model Parameters

Usage

```
## S3 method for class 'beezdemand_tmb'
confint(
  object,
  parm = NULL,
  level = 0.95,
  report_space = c("internal", "natural"),
  method = c("wald", "simulate"),
  R = 1000L,
  seed = NULL,
  ...
)
```

Arguments

object	A beezdemand_tmb object.
parm	Character vector of parameter names.
level	Confidence level (default 0.95).
report_space	Character. "internal" or "natural". When "natural", beta_q0, beta_alpha, and log_k are exponentiated to the natural scale. For the intercept, this gives Q0 or alpha at the reference level. For non-intercept terms, the exponentiated value represents a multiplicative fold-change (ratio) relative to the reference level, not the absolute parameter value for that group. Variance parameters (logsigma_*, rho_bc_raw) remain on their internal scales; use summary() or .tmb_format_variance_components() for transformed variance components.
method	Character. "wald" (default) returns Hessian-based Wald intervals (coef +/- z * se). "simulate" draws R parametric Monte Carlo samples from the asymptotic Gaussian sampling approximation $N(\hat{\beta}, \hat{\Sigma})$ (with $\hat{\Sigma} = \text{vcov}(\text{object})$) and reports per-coefficient empirical quantiles.
R	Integer. Number of Monte Carlo draws for method = "simulate". Must be ≥ 100 ; ≥ 1000 is recommended for stable quantiles. Ignored for method = "wald".
seed	Optional integer seed for method = "simulate" reproducibility. When supplied, the caller's RNG state is restored on exit so the global stream is left unperturbed.
...	Additional arguments.

Details

method = "simulate" is Monte Carlo simulation from the asymptotic Gaussian sampling approximation to the MLE (neither a Bayesian posterior, nor a data-resampling bootstrap, nor a profile-likelihood interval). Because the sampled distribution is the same Gaussian that Wald assumes, the simulated per-coefficient quantiles converge to the Wald intervals as $R \rightarrow \infty$; the method does **not** improve on Wald at boundary cases and offers no positivity guarantee on the internal scale (logsigma_* intervals can be negative). Its value is (a) a diagnostic side-by-side check on the Gaussian approximation, and (b) a shared draw primitive ([.tmb_parametric_draws\(\)](#)) for derived-metric confidence intervals.

Value

A tibble with term, estimate, conf.low, conf.high, level.

See Also

`confint.beezdemand_nlme()`, `vcov.beezdemand_tmb()`.

Examples

```
data(apt)
fit <- fit_demand_tmb(apt, equation = "exponential", verbose = 0)
confint(fit)
confint(fit, report_space = "natural")
# Diagnostic Monte Carlo intervals (asymptotically Wald-equivalent):
confint(fit, method = "simulate", R = 1000, seed = 42)
```

confint.cp_model_lm	<i>Confidence Intervals for a Cross-Price Demand Model (Linear)</i>
---------------------	---

Description

Wald confidence intervals via `stats::confint.default()`.

Usage

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

Arguments

object	A cp_model_lm object.
parm	Optional character vector of parameter names. NULL returns intervals for all parameters.
level	Confidence level (default 0.95).
...	Additional arguments (unused).

Value

A tibble with columns term, estimate, conf.low, conf.high, level, method.

confint.cp_model_lmer *Confidence Intervals for a Cross-Price Demand Model (Mixed-Effects)*

Description

Wraps `lme4::confint.merMod()`. Defaults to Wald intervals; `method = "profile"` or `method = "boot"` use the corresponding lme4 backends and are slower.

Usage

```
## S3 method for class 'cp_model_lmer'
confint(
  object,
  parm = NULL,
  level = 0.95,
  method = c("Wald", "profile", "boot"),
  ...
)
```

Arguments

<code>object</code>	A <code>cp_model_lmer</code> object.
<code>parm</code>	Optional character vector of parameter names. <code>NULL</code> returns intervals for all parameters.
<code>level</code>	Confidence level (default 0.95).
<code>method</code>	One of "Wald" (default), "profile", or "boot".
<code>...</code>	Additional arguments passed to <code>lme4::confint.merMod()</code> .

Value

A tibble with columns `term`, `estimate`, `conf.low`, `conf.high`, `level`, `method`.

confint.cp_model_nls *Confidence Intervals for Cross-Price NLS Model Parameters*

Description

Computes confidence intervals for parameters from a nonlinear cross-price demand model using `nlstools::confint2()`.

Usage

```
## S3 method for class 'cp_model_nls'
confint(
  object,
  parm = NULL,
  level = 0.95,
  method = c("asymptotic", "profile"),
  ...
)
```

Arguments

object	A cp_model_nls object from <code>fit_cp_nls()</code> .
parm	Character vector of parameter names to compute CIs for. Default includes all parameters.
level	Confidence level (default 0.95).
method	Character. Method for computing intervals passed to <code>nlstools::confint2()</code> : • "asymptotic" (default): Wald-type asymptotic intervals • "profile": Profile-t confidence intervals
...	Additional arguments passed to <code>nlstools::confint2()</code> .

Details

This method wraps `nlstools::confint2()` to provide confidence intervals for the log10-parameterized coefficients (`log10_qalone`, `I`, `log10_beta`).

For back-transformed natural-scale confidence intervals, apply the transformation: $10^{\text{conf.low}}$ and $10^{\text{conf.high}}$ for log10-scale parameters.

Value

A tibble with columns: `term`, `estimate`, `conf.low`, `conf.high`, `level`, `method`.

Examples

```
data(etm)
fit <- fit_cp_nls(etm, equation = "exponentiated")
confint(fit)
```

cp	<i>Example cross-price dataset</i>
----	------------------------------------

Description

A small illustrative dataset of price (x) and consumption (y) target (target), and group (group).

Usage

cp

Format

A data frame with N rows and columns:

id unique identifier
x price of the alternative
y consumption/demand
target target (e.g., alone, own, alt)
group e.g., drug or product

cp_posthoc_intercepts *Run pairwise intercept comparisons for cross-price demand model*

Description

Performs pairwise comparisons of intercepts between groups in a cross-price demand model, but only when a significant interaction is present. The emmeans table showing estimated marginal means for intercepts is always returned.

Usage

```
cp_posthoc_intercepts(object, alpha = 0.05, adjust = "tukey", ...)
```

Arguments

object	A cp_model_lmer object from fit_cp_linear
alpha	Significance level for testing (default: 0.05)
adjust	Method for p-value adjustment; see emmeans::contrast (default: "tukey")
...	Additional arguments passed to emmeans

Value

List containing the emmeans table and optionally pairwise comparisons if interaction is significant

Examples

```
data(etm)
fit <- fit_cp_linear(etm, type = "mixed", group_effects = TRUE)
cp_posthoc_intercepts(fit)
```

cp_posthoc_slopes	<i>Run pairwise slope comparisons for cross-price demand model</i>
-------------------	--

Description

Performs pairwise comparisons of slopes between groups in a cross-price demand model, but only when a significant interaction is present. The emmeans table showing estimated marginal means for slopes is always returned.

Usage

```
cp_posthoc_slopes(object, alpha = 0.05, adjust = "tukey", ...)
```

Arguments

object	A cp_model_lmer object from fit_cp_linear
alpha	Significance level for testing (default: 0.05)
adjust	Method for p-value adjustment; see emmeans::contrast (default: "tukey")
...	Additional arguments passed to emmeans

Value

List containing the emmeans table and optionally pairwise comparisons if interaction is significant

Examples

```
data(etm)
fit <- fit_cp_linear(etm, type = "mixed", group_effects = TRUE)
cp_posthoc_slopes(fit)
```

 etm

Example Experimental Tobacco Marketplace data

Description

A dataset containing ETM data for a small number of participants

Usage

```
etm
```

Format

Long-form data.frame with columns: id, x, y, target, group. Participants were asked how many cigarettes, e-cigarettes, combustible, and non-combustible products they would buy at various prices.

 extract_coefficients *Extract All Coefficient Types from Cross-Price Demand Models*

Description

A convenience function to extract coefficients from any type of cross-price demand model in a unified format. For mixed effects models, returns a list with different coefficient types.

Usage

```
extract_coefficients(object, ...)
```

Arguments

object	A cross-price demand model object (cp_model_nls, cp_model_lm, or cp_model_lmer)
...	Additional arguments passed to the appropriate coef method

Value

For cp_model_nls and cp_model_lm, returns the model coefficients. For cp_model_lmer, returns a list with fixed, random, and combined coefficients.

Examples

```
data(etm, package = "beezdemand")
fit <- fit_cp_nls(etm, equation = "exponentiated")
extract_coefficients(fit)
```

<i>ExtraF</i>	<i>ExtraF</i>
---------------	---------------

Description

Extra Sum of Squares F-test

Usage

```
ExtraF(  
  dat,  
  equation = "hs",  
  groups = NULL,  
  verbose = FALSE,  
  k,  
  compare = "alpha",  
  idcol = "id",  
  xcol = "x",  
  ycol = "y",  
  groupcol = NULL,  
  start_alpha = 0.001  
)
```

Arguments

<code>dat</code>	Long form data frame
<code>equation</code>	"hs"
<code>groups</code>	NULL for all. Character vector matching groups in <code>groupcol</code>
<code>verbose</code>	If TRUE, prints all output including models
<code>k</code>	User-defined k value; if missing will attempt to find shared k and then mean empirical range (in log units)
<code>compare</code>	Specify whether to compare alpha or Q0. Default is alpha
<code>idcol</code>	The column name that should be treated as dataset identifier
<code>xcol</code>	The column name that should be treated as "x" data
<code>ycol</code>	The column name that should be treated as "y" data
<code>groupcol</code>	The column name that should be treated as the groups
<code>start_alpha</code>	Optional numeric to inform starting value for alpha

Details

One alpha better than individual alphas?

Value

List of results and models

Author(s)

Brent Kaplan bkaplan.ku@gmail.com

Examples

```
## Compare two groups using equation by Koffarnus et al., 2015 and a fixed k of 2

apt$group <- NA
apt[apt$id %in% sample(unique(apt$id), length(unique(apt$id))/2), "group"] <- "a"
apt$group[is.na(apt$group)] <- "b"
ExtraF(apt, "koff", k = 2, groupcol = "group")
```

find_n_demand

Find the smallest sample size reaching a target power (demand)

Description

Bisection search over `n_subjects` for the smallest `N` whose Monte Carlo power estimate from `power_demand()` reaches `target_power`. The search accounts for Monte Carlo noise: at each evaluated `N`, replicates are added in batches (up to `n_sim_max`) until the Wilson interval for power lies wholly above or below the target; if it still straddles the target at the cap, the decision falls back to the point estimate and the result is flagged uncertain. The selected `N` and its lower neighbor are then re-evaluated with fresh replicates before minimality is claimed.

The returned `n` is an *estimated minimum under Monte Carlo uncertainty* rather than an exact bound. For grant-quality reporting, rerun `power_demand()` at the returned `n` with a large `n_sim` (2000+) and report that estimate with its Monte Carlo confidence interval.

Monotonicity assumption. Bisection presumes that power is non-decreasing in `n_subjects`. Because every evaluated `N` is judged from its own independent replicates (and a convergence-conditioned denominator), a Monte Carlo fluctuation can make a lower `N` read "below" when its true power is above the target, so the search may step past a lower crossing that it never revisits. Evaluated `N` that contradict the assumption (a lower `N` reading "above" the selected `N`) demote the status to "uncertain"; `N` that were never evaluated cannot be checked. Widen `n_sim/n_sim_max` when the reported `n` matters. When the target is already met at `n_range[1]`, that bound is likewise re-evaluated with fresh replicates before "at_lower_bound" is reported; if the second look does not clear the target the bound is treated as below and the bisection proceeds upward.

Usage

```
find_n_demand(
  target_power = 0.8,
  effect = list(delta_q0 = NULL, delta_alpha = NULL),
  design = list(),
  n_range = c(6, 200),
  n_sim = 200,
  n_sim_max = 4 * n_sim,
```

```

    alpha = 0.05,
    df = NULL,
    seed = NULL,
    equation = "simplified",
    random_effects = NULL,
    multi_start = FALSE,
    verbose = TRUE,
    design_type = c("within", "between"),
    ...
)

```

Arguments

target_power	Target power in (0, 1).
effect	Named list supplying exactly one of delta_q0 or delta_alpha: the true condition shift on natural-log Q0 (or natural-log alpha) for condition 2 relative to condition 1. 0 is allowed (useful for Type I error checks). E.g. delta_q0 = log(1.5) means condition 2's Q0 is 1.5 times condition 1's.
design	Named list of data-generating settings, merged over the simulator defaults: prices (vector), log_q0_pop, log_alpha_pop, sigma_b (per-condition subject SD on log Q0), sigma_d (same for log alpha), rho_bd (must be 0 in this version), and sigma_e (residual SD on log consumption). All SDs must be strictly positive.
n_range	Integer bracket c(lower, upper) to search ($2 \leq \text{lower} < \text{upper}$). The search errors (rather than extrapolating) if the target is not reached at upper.
n_sim	Replicates per evaluation batch. Smaller than the <code>power_demand()</code> default because several N values are evaluated; the adaptive rule adds batches where the verdict is close.
n_sim_max	Maximum replicates per evaluated N (default $4 * n_sim$).
alpha	Nominal two-sided test level.
df	Degrees of freedom for the Wald test's t reference. NULL (default) tracks the evaluated sample size as $n - 1$ for design_type = "within" and $n - 2$ for design_type = "between"; a numeric value (or Inf for the asymptotic z-test) is used at every evaluated N. With the default df and design_type = "between", n_range[1] must be ≥ 3 (df = $n - 2$ needs $n \geq 3$).
seed	Optional integer seed; identical seeds give identical results. The caller's RNG state is restored on exit.
equation	Demand equation passed to <code>fit_demand_tmb()</code> . The default "simplified" shares the simulator's mean function, so the simulated delta and the fitted contrast share the same scale. Note the error model is a <i>working model</i> : the simulator draws multiplicative lognormal errors while "simplified" fits additive Gaussian errors on raw consumption; the approximation is closest at small sigma_e, and Type I calibration is verified by the test suite at the default and a 3x-larger sigma_e. Other equations are sensitivity analyses with a different estimand, so their contrasts are not on the scale of the simulated delta.
random_effects	Random-effects specification passed to <code>fit_demand_tmb()</code> . NULL (default) resolves to a specification matching design_type: nlme::pdDiag(Q0 + alpha

	~ condition - 1) for "within" (independent per-condition subject effects on both parameters, matching the simulator's data-generating process when $\rho_{bd} = 0$), and $\text{nlme}::\text{pdDiag}(Q0 + \alpha \sim 1)$ for "between" (per-subject intercepts, which are correctly specified because each subject appears in only one condition). Supply a specification to override.
multi_start	Passed to <code>fit_demand_tmb()</code> . Defaults to FALSE for speed (roughly 3x fewer optimizations); non-convergent replicates are excluded and surfaced rather than biasing the estimate.
verbose	Logical; report each evaluation.
design_type	Either "within" (default) or "between". "within" simulates the two-condition within-subject design (every subject observed at every price in both conditions) and tests the within-subject condition contrast. "between" assigns each subject to exactly one of two arms ($\text{ceiling}(n/2)$ to condition 1, the rest to condition 2) and tests the group difference; the two-arm dataset is composed from the same simulator run once per arm with a single condition, so no new data-generating process is introduced (see Details). The <code>df</code> and <code>random_effects</code> defaults track <code>design_type</code> .
...	Additional arguments passed to <code>fit_demand_tmb()</code> .

Value

An object of class `beezdemand_power_n`: a list with

n Estimated smallest `n_subjects` reaching `target_power`; NA when the confirmation pass contradicted the search (`status = "unresolved"`).

target_power As supplied.

status "confirmed" (selected `N` re-confirmed above target and `N - 1` below), "uncertain" (a decision relied on a point estimate, `N - 1` also cleared the target on reconfirmation, or an evaluated lower `N` read above the target, so the returned `N` may not be minimal), "unresolved" (the selected `N` failed its own reconfirmation; `n` is NA), or "at_lower_bound" (the target was already met at `n_range[1]` on two independent looks; smaller `N` was not explored; widen `n_range` downward if that matters). These labels describe a heuristic Monte Carlo decision rule (repeated looks at ordinary Wilson intervals across several `N`) rather than a formal sequential error guarantee.

uncertain Logical; TRUE when any search decision was made on a point estimate rather than a conclusive Wilson interval, or the status is not "confirmed"/"at_lower_bound".

evaluations Tibble of every evaluation: `n_subjects`, `n_sim_total`, `n_used`, `usable_fraction`, `power`, `ci_lower`, `ci_upper`, `decision`. A warning fires when any evaluation had fewer than 95% usable fits.

alpha, **df**, **effect**, **design**, **n_range**, **n_sim**, **n_sim_max**, **seed**, **settings**, **call** Echoed inputs and effective settings.

See Also

`power_demand()` for the Monte Carlo engine.

Other power-analysis: `power_demand()`

Examples

```
# Tiny search for demonstration only (use n_sim >= 200 and a wide
# n_range for real planning; see vignette("power-analysis"))
res <- find_n_demand(
  target_power = 0.8,
  effect = list(delta_q0 = log(2.5)),
  n_range = c(4, 8), n_sim = 5, n_sim_max = 10, seed = 1, verbose = FALSE
)
print(res)
```

fit_cp_linear

*Fit a Linear Cross-Price Demand Model***Description**

Fit a Linear Cross-Price Demand Model

Usage

```
fit_cp_linear(
  data,
  type = c("fixed", "mixed"),
  x_var = "x",
  y_var = "y",
  id_var = "id",
  group_var = "group",
  target_var = "target",
  filter_target = TRUE,
  target_level = "alt",
  formula = NULL,
  log10x = FALSE,
  group_effects = FALSE,
  random_slope = FALSE,
  return_all = TRUE,
  ...
)

fit_cp_linear.default(
  data,
  formula = NULL,
  log10x = FALSE,
  return_all = FALSE,
  ...
)
```

```

fit_cp_linear.mixed(
  data,
  formula = NULL,
  log10x = FALSE,
  return_all = FALSE,
  ...
)

```

Arguments

<code>data</code>	A data frame containing columns for price, consumption, and optionally target indicator, subject identifier, and group.
<code>type</code>	The type of model: "fixed" for standard linear or "mixed" for mixed effects.
<code>x_var</code>	Character string; name of the price column. Default is "x". Renamed to "x" internally.
<code>y_var</code>	Character string; name of the consumption column. Default is "y". Renamed to "y" internally.
<code>id_var</code>	Character string; name of the subject identifier column. Default is "id". Required for mixed models; renamed to "id" internally.
<code>group_var</code>	Character string; name of the group column. Default is "group". Renamed to "group" internally when present.
<code>target_var</code>	Character string; name of the target indicator column. Default is "target". Renamed to "target" internally when present.
<code>filter_target</code>	Logical; if TRUE (default), filters to rows where the target column equals <code>target_level</code> .
<code>target_level</code>	Character string; value of the target column to retain when <code>filter_target = TRUE</code> . Default is "alt".
<code>formula</code>	Optional formula override. If NULL, a formula will be constructed based on other parameters. If non-NULL and any <code>*_var</code> argument differs from its default, an error is thrown because the formula references canonical column names that no longer exist before renaming; rename columns before calling, or omit the formula argument.
<code>log10x</code>	Logical; if TRUE and formula is NULL, uses $\log_{10}(x)$ instead of x in the formula. Default is FALSE.
<code>group_effects</code>	Logical or character; if TRUE, includes group as a factor with interactions. Can also be "intercept" for group intercepts only or "interaction" for full interactions. Default is FALSE.
<code>random_slope</code>	Logical; for mixed models, if TRUE, includes random slopes for x . Default is FALSE.
<code>return_all</code>	Logical; if TRUE, returns additional model metadata.
<code>...</code>	Additional arguments passed to underlying modeling functions.

Value

Fitted linear model.

Examples

```
data(etm)
## Fixed-effects linear cross-price model
fit_fixed <- fit_cp_linear(etm, type = "fixed", group_effects = TRUE)
summary(fit_fixed)

## Mixed-effects linear cross-price model
fit_mixed <- fit_cp_linear(etm, type = "mixed", group_effects = TRUE)
summary(fit_mixed)
```

fit_cp_nls

Fit cross-price demand with NLS (with fallback optimizers)

Description

Fits a **cross-price demand** curve using log10-parameterization for numerical stability. The optimizer estimates parameters on the log10 scale where applicable, so positivity constraints hold automatically.

Equation forms:

- **Exponentiated** (default):

$$y = Q_{alone} \cdot 10^{I \cdot \exp(-\beta \cdot x)}$$

- **Exponential** (fits on log10 response scale):

$$\log_{10}(y) = \log_{10}(Q_{alone}) + I \cdot \exp(-\beta \cdot x)$$

- **Additive** (level on y):

$$y = Q_{alone} + I \cdot \exp(-\beta \cdot x)$$

where x is the alternative product price (or "cross" price) and y is consumption of the target good.

Optimizer parameters (log10 parameterization):

- $\log_{10_qalone}$: $\log_{10}(Q_{alone})$ - baseline consumption when the alternative is effectively absent.
- I : cross-price **interaction intensity**; sign and magnitude reflect substitution/complementarity. Unconstrained (can be negative for substitutes).
- $\log_{10_beta}$: $\log_{10}(\beta)$ - rate at which cross-price influence decays as x increases.

Natural-scale values are recovered as $Q_{alone} = 10^{\log_{10_qalone}}$ and $\beta = 10^{\log_{10_beta}}$.

The function first attempts a multi-start nonlinear least squares fit (`nls.multstart`). If that fails (or if explicit `start_values` are provided), it falls back to `minpack.lm::nlsLM`. Optionally, it will make a final attempt with `nlsr::wrapnlsr`. Returns either the fitted model or a structured object with metadata for downstream methods.

Usage

```
fit_cp_nls(
  data,
  equation = c("exponentiated", "exponential", "additive"),
  x_var = "x",
  y_var = "y",
  start_values = NULL,
  start_vals = lifecycle::deprecated(),
  iter = 100,
  bounds = NULL,
  fallback_to_nlsr = TRUE,
  return_all = TRUE
)
```

Arguments

<code>data</code>	A data frame with columns for price and consumption. Additional columns are ignored. Input is validated internally.
<code>equation</code>	Character string; model family, one of <code>c("exponentiated", "exponential", "additive")</code> . Default is "exponentiated".
<code>x_var</code>	Character string; name of the price column in data. Default is "x". The column is renamed to "x" internally; see <code>validate_cp_data()</code> .
<code>y_var</code>	Character string; name of the consumption column in data. Default is "y". The column is renamed to "y" internally.
<code>start_values</code>	Optional named list of initial values for parameters <code>log10_qalone</code> , <code>I</code> , and <code>log10_beta</code> . If <code>NULL</code> , the function derives plausible ranges from the data and uses multi-start search.
<code>start_vals</code>	[Deprecated] Use <code>start_values</code> instead.
<code>iter</code>	Integer; number of random starts for <code>nls.multstart</code> (default 100).
<code>bounds</code>	Deprecated. Log10-parameterized parameters are naturally unbounded. This argument is ignored but retained for backwards compatibility.
<code>fallback_to_nlsr</code>	Logical; if <code>TRUE</code> (default), try <code>nlsr::wrapnlsr</code> when both multi-start NLS and <code>nlsLM</code> fail.
<code>return_all</code>	Logical; if <code>TRUE</code> (default), return a list containing the model and useful metadata. If <code>FALSE</code> , return the bare fitted model object.

Details

Start values. When `start_values` is missing, the function: (1) estimates a reasonable range for `log10_qalone` from the observed `y`, (2) estimates `log10_beta` from the price range, and (3) launches a multi-start grid in `nls.multstart`.

Zero handling for exponential equation. Since the exponential equation fits on the $\log_{10}(y)$ scale, observations with $y \leq 0$ are automatically removed with a warning. Use the exponentiated or additive forms if you need to retain zero consumption values.

Fitting pipeline (short-circuiting):

1. `nls.multstart::nls_multstart()` with random starts.
2. If that fails (or if `start_values` provided): `minpack.lm::nlsLM()` using `start_values` (user or internally estimated).
3. If that fails and `fallback_to_nlsr = TRUE`: `nlsr::wrapnlsr()`.

The returned object has class `"cp_model_nls"` (when `return_all = TRUE`) with components: `model`, `method` (the algorithm used), `equation`, `start_vals`, `nlsLM_fit`, `nlsr_fit`, and the data used. This is convenient for custom print/summary/plot methods.

Value

If `return_all = TRUE` (default): a list of class `"cp_model_nls"`:

- `model`: the fitted object from the successful backend.
- `method`: one of `"nls_multstart"`, `"nlsLM"`, or `"wrapnlsr"`.
- `equation`: the model family used.
- `start_vals`: named list of starting values (final used; kept for backward compatibility).
- `nlsLM_fit`, `nlsr_fit`: fits from later stages (if attempted; NULL when that stage was not reached, or was attempted but did not itself produce a fit).
- `data`: the 2-column data frame actually fit.
- `convergence`: list with `isConv`, `finIter`, `stopCode`, `stopMessage` for the winning backend, read from `model$convInfo`; populated for `nls/nlsLM`-class fits, and for `nlsr::wrapnlsr()` fits too when it returns a plain `nls`-class object (its usual successful case); `isConv = NA` only when the winning fit carries no `convInfo` at all.

If `return_all = FALSE`: the fitted model object from the successful backend.

Convergence & warnings

- `summary()` and `confint()` on the returned object emit a classed warning (`beezdemand_cp_nls_nonconverged_warning`) when `convergence$isConv` is explicitly FALSE for the winning fit; unreported convergence (NA, when the winning object carries no `convInfo`) is silent.
- Poor scaling or extreme y dispersion can make parameters weakly identified.
- For `"exponential"`, the model fits on the $\log_{10}(y)$ scale internally.

See Also

[check_unsystematic_cp](#) for pre-fit data screening, [validate_cp_data](#) for input validation.

Examples

```
## --- Example: Real data (E-Cigarettes, id = 1) ---
dat <- structure(list(
  id = c(1, 1, 1, 1, 1, 1),
  x = c(2, 4, 8, 16, 32, 64),
  y = c(3, 5, 5, 16, 17, 13),
  target = c("alt", "alt", "alt", "alt", "alt", "alt"),
  group = c("E-Cigarettes", "E-Cigarettes", "E-Cigarettes",
```

```
      "E-Cigarettes", "E-Cigarettes", "E-Cigarettes")
), row.names = c(NA, -6L), class = c("tbl_df", "tbl", "data.frame"))

## Fit the default (exponentiated) cross-price form
fit_ecig <- fit_cp_nls(dat, equation = "exponentiated", return_all = TRUE)
summary(fit_ecig)           # model summary
fit_ecig$method             # backend actually used (e.g., "nls_multstart")
coef(fit_ecig$model)       # parameter estimates: log10_qalone, I, log10_beta
```

fit_demand_fixed	<i>Fit Fixed-Effect Demand Curves</i>
------------------	---------------------------------------

Description

Modern interface for fitting individual demand curves via nonlinear least squares. Returns a structured S3 object with standard methods including `summary()`, `tidy()`, and `glance()`.

Usage

```
fit_demand_fixed(
  data,
  equation = c("hs", "koff", "simplified", "linear", "exponential", "exponentiated"),
  k = 2,
  agg = NULL,
  x_var = "x",
  y_var = "y",
  id_var = "id",
  param_space = c("natural", "log10"),
  by = NULL,
  multistart = TRUE,
  S = NULL,
  ...
)
```

Arguments

data	Data frame in long format with columns: id, x (price), y (consumption).
equation	Character. Equation type: "hs" (Hursh & Silberberg, 2008), "koff" (Kof-farnus et al., 2015), "simplified" (Rzeszutek et al., 2025; simplified expo-nential with normalized decay, no k parameter), or "linear". The modern aliases "exponential" (equivalent to "hs") and "exponentiated" (equivalent to "koff") are also accepted. Default "hs".
k	Scaling constant. Numeric value (fixed), "ind" (individual), "fit" (free param-eter), or "range" (data-driven). Default 2.
agg	Character. Aggregation method: "Mean", "Pooled", or NULL for individual fits. Default NULL.
x_var	Character. Name of the price column. Default "x".

y_var	Character. Name of the consumption column. Default "y".
id_var	Character. Name of the subject identifier column. Default "id".
param_space	Character. Parameterization used for fitting. One of: • "natural": fit Q_0, α (and k if $k = \text{"fit"}$) on their natural scale • "log10": fit $\log_{10}(Q_0)$, $\log_{10}(\alpha)$ (and $\log_{10}(k)$ if $k = \text{"fit"}$)
by	Optional character vector of column names to group by. When supplied, fits are run separately within each unique combination of the by columns. Returns a beezdemand_fixed_grouped object with per-group child fits. Default NULL (no grouping).
multistart	Logical. If TRUE (the default), subjects whose production-heuristic fit is not strict-converged (converged_strict; see Details) are automatically re-fit from $S - 1$ additional sampled starting values. Subjects that strict-converge on the production start are never refit, so their results are byte-identical whether multistart is TRUE or FALSE. Set to FALSE (or $S = 1$) to reproduce the legacy single-start behavior exactly. Not applicable to equation = "linear" (closed-form; never multistarted).
S	Integer or NULL. Total number of starts to try per subject (including the production start), when multistart = TRUE. Default NULL uses a tiered budget: 8 for 2-parameter forms (hs/koff/simplified with a fixed k), 32 when $k = \text{"fit"}$. Ignored for equation = "linear". If supplied, must be a single finite integer ≥ 1 . Note: multistart and S were added after by in the argument list so that pre-existing positional calls (e.g., fit_demand_fixed(data, "hs", 2, NULL, "x", "y", "id", "natural", "group_col")), where the 9th positional argument is by) continue to bind correctly. Always pass multistart/S by name.
...	Additional arguments passed to the underlying FitCurves() engine.

Details

This function is a modern wrapper around the legacy FitCurves() function. It provides the same fitting capabilities but returns a structured S3 object with standardized methods for model interrogation.

Multi-start rescue protocol:

fit_demand_fixed() always runs FitCurves()'s existing heuristic start exactly as before (the "production start"). A subject whose production fit is strict-converged (converged_strict: the optimizer's own convergence flag AND finite coefficients/objective AND not sitting on a user-supplied bound) is accepted immediately; no sampled starts are ever run for it, so its row, fitted model, predictions, and data are byte-identical to the multistart = FALSE / $S = 1$ protocol by construction. Only subjects whose production fit is not strict-converged are re-fit from $S - 1$ additional starts, sampled log-uniformly in interpretable (Q_0 , Pmax) coordinates and mapped to each equation's native (Q_0 , α) parameterization via the same closed forms used by beezdemand_calc_pmax_omax(). Among the sampled attempts that themselves strict-converge, the minimum-residual-SS start wins (ties broken by draw order). If none of the sampled starts strict-converge, the original (non-converged) production row is kept. equation = "linear" is a closed-form fit and is never multistarted. FitCurves() itself is unchanged; sampling draws from the ambient RNG stream (call set.seed() before fit_demand_fixed() for reproducibility; the helpers never call set.seed() themselves).

Value

An object of class `beezdemand_fixed` with components:

results Data frame of fitted parameters for each subject. Gains `n_starts_tried`, `n_starts_converged`, and `start_source` ("production", "sampled", or "none") from the multi-start protocol; see Details.

fits List of model fit objects (if `detailed = TRUE` internally)

predictions List of prediction data frames

data_used List of data frames used for each fit

multistart List describing the multi-start protocol: `multistart`, `S` (resolved budget), `equation`, `eligible` (whether this equation supports rescue), and `summary` (per-subject start metadata, or `NULL` when not applicable)

call The original function call

equation The equation form used

k_spec Description of `k` specification

agg Aggregation method used

n_total Total number of subjects/fits attempted

n_success Number of successful fits

n_fail Number of failed fits

See Also

[fit_demand_tmb\(\)](#) for TMB mixed-effects models, [fit_demand_mixed\(\)](#) for NLME mixed-effects models, [fit_demand_hurdle\(\)](#) for hurdle models.

Other demand-fitting: [fit_demand_hurdle\(\)](#), [fit_demand_mixed\(\)](#), [fit_demand_tmb\(\)](#)

Examples

```
data(apt)
fit <- fit_demand_fixed(apt, equation = "hs", k = 2)
print(fit)
summary(fit)
tidy(fit)
glance(fit)

# Grouped analysis: fit separately by gender (subset keeps it fast)
data(apt_full)
ids <- unique(apt_full[, c("id", "gender")])
ids <- ids[ids$gender %in% c("Male", "Female"), ]
keep <- unlist(lapply(split(ids$id, ids$gender), head, 40))
dat <- apt_full[apt_full$id %in% keep, ]
fit_g <- fit_demand_fixed(dat, equation = "hs", k = 2, by = "gender")
tidy(fit_g)  # group column prepended
glance(fit_g) # one row per group
```

fit_demand_hurdle	<i>Fit Two-Part Mixed Effects Hurdle Demand Model</i>
-------------------	---

Description

Fits a two-part hurdle model for demand data using TMB (Template Model Builder). Part I models the probability of zero consumption using logistic regression. Part II models log-consumption given positive response using a nonlinear mixed effects model.

Usage

```
fit_demand_hurdle(
  data,
  y_var,
  x_var,
  id_var,
  random_effects = c("zeros", "q0", "alpha"),
  epsilon = 0.001,
  start_values = NULL,
  tmb_control = list(max_iter = 200, eval_max = 1000, trace = 0),
  verbose = 1,
  part2 = c("zhao_exponential", "exponential", "simplified_exponential"),
  ...
)
```

Arguments

<code>data</code>	A data frame containing the demand data.
<code>y_var</code>	Character string specifying the column name for consumption values.
<code>x_var</code>	Character string specifying the column name for price.
<code>id_var</code>	Character string specifying the column name for subject IDs.
<code>random_effects</code>	Character vector specifying which random effects to include. Options are "zeros" (<code>a_i</code> for Part I), "q0" (<code>b_i</code> for intensity), and "alpha" (<code>c_i</code> for elasticity). Default is <code>c("zeros", "q0", "alpha")</code> for the full 3-random-effect model. Use <code>c("zeros", "q0")</code> for the simplified 2-random-effect model (fixed alpha across subjects).
<code>epsilon</code>	Small constant added to price before log transformation in Part I. Used to handle zero prices: $\log(\text{price} + \text{epsilon})$. Default is 0.001.
<code>start_values</code>	Optional named list of starting values for optimization. If NULL (default), sensible defaults are used.
<code>tmb_control</code>	List of control parameters for TMB optimization: max_iter Maximum number of optimization iterations (default 200) eval_max Maximum number of function evaluations (default 1000) trace Print optimization trace: 0 = none, 1 = some (default 0)

verbose	Integer controlling output verbosity: 0 = silent, 1 = progress messages, 2 = detailed optimization trace. Default is 1.
part2	Character string selecting the Part II mean function. Options are "zhao_exponential" (default; no Q0 normalization in the exponent), "exponential" (HS-standardized; Q0 inside the exponent), and "simplified_exponential" (SND/log-linear; no k parameter).
...	Additional arguments (reserved for future use).

Details

The model structure is:

Part I (Binary - probability of zero consumption):

$$\text{logit}(\pi_{ij}) = \beta_0 + \beta_1 \cdot \log(\text{price} + \epsilon) + a_i$$

Part II (Continuous - log consumption given positive):

With 3 random effects (random_effects = c("zeros", "q0", "alpha")):

$$\log(Q_{ij}) = (\log Q_0 + b_i) + k \cdot (\exp(-\alpha_i \cdot \text{price}) - 1) + \epsilon_{ij}$$

where $\alpha_i = \exp(\log(\alpha) + c_i)$ and $k = \exp(\log(k))$.

With 2 random effects (random_effects = c("zeros", "q0")):

$$\log(Q_{ij}) = (\log Q_0 + b_i) + k \cdot (\exp(-\alpha \cdot \text{price}) - 1) + \epsilon_{ij}$$

where $\alpha = \exp(\log(\alpha))$ and $k = \exp(\log(k))$.

Random effects follow a multivariate normal distribution with unstructured covariance matrix. Use [compare_hurdle_models](#) for likelihood ratio tests comparing nested models.

Value

An object of class `beezdemand_hurdle` containing:

model List with coefficients, se, variance_components, correlations

random_effects Matrix of empirical Bayes random effect estimates

subject_pars Data frame of subject-specific parameters including Q0, alpha, breakpoint, Pmax, Omax

tmb_obj TMB objective function object

opt Optimization result from `nlmminb`

sdr TMB `sdreport` object

call The matched call

data Original data used for fitting

param_info List with y_var, x_var, id_var, n_subjects, n_obs, etc.

converged Logical indicating convergence

re_cov_fallback Logical. TRUE when the estimated random-effects covariance was not positive definite and the subject-level effects were computed from an uncorrelated (diagonal) approximation; the reported correlations do not apply to those outputs. `summary()` and `print()` carry a note when it is TRUE.

loglik Log-likelihood at convergence

AIC, BIC Information criteria

error_message Error message if fitting failed, NULL otherwise

Parameterization and comparability

The TMB backend estimates positive-constrained parameters on the natural-log scale: $\log(Q_0)$, $\log(\alpha)$, and $\log(k)$. Reporting methods (`summary()`, `tidy()`, `coef()`) can back-transform to the natural scale or present parameters on the $\log_{10}$ scale.

To compare α estimates with models fit in $\log_{10}$ space, use:

$$\log_{10}(\alpha) = \log(\alpha) / \log(10).$$

The default `part2 = "zhao_exponential"` places no Q_0 inside the exponent ($\exp(-\alpha_{Zhao} P)$), whereas the Hursh & Silberberg form used by `fit_demand_fixed(equation = "hs")` and `fit_demand_tmb(equation = "exponential")` uses $\exp(-\alpha_{HS} Q_0 P)$. The two α values are therefore on different scales and relate as

$$\alpha_{HS} = \alpha_{Zhao} / Q_0,$$

so a Zhao α is not directly comparable with an HS α unless divided by the subject's (or group's) Q_0 . Use `part2 = "exponential"` (alias `"hs_stdq0"`) for an HS-scaled α .

Convergence

The 3-random-effect spec (`random_effects = c("zeros", "q0", "alpha")`) can *false-converge* on real purchase task data: `nlminb()` reports a nonzero convergence code (e.g. "false convergence (8)") while the Hessian is not positive definite (`fit$hessian_pd == FALSE`). This reflects weak identification of the alpha random effect rather than a broken model specification (companion datasets from the same source can converge cleanly with the identical spec). `fit$opt$message` and `fit$opt$convergence` retain `nlminb`'s own diagnostic fields alongside `fit$converged` and `fit$hessian_pd`; `print()` and `summary()` display a prominent warning block (quoting the optimizer message) whenever either signals a problem, so a non-converged or non-PD-Hessian fit is hard to miss even without inspecting these fields directly.

The recommended **stability check** when the 3RE spec reports non-convergence or a non-PD Hessian is to refit with `random_effects = c("zeros", "q0")` (dropping the alpha random effect, which converges more readily) and compare the empirical-Bayes subject parameters between the two fits. Broadly similar per-subject estimates support treating the 2RE fit's conclusions as stable across the two specifications. This is a diagnostic comparison rather than a claim that the 2RE spec is generally preferred, and `beezdemand` does not refit automatically.

See Also

[summary.beezdemand_hurdle](#), [predict.beezdemand_hurdle](#) (and its *Scoring predictions* section: score zero-inclusive consumption with `type = "demand"`, the default since 0.3.0), [plot.beezdemand_hurdle](#), [compare_hurdle_models](#), [simulate_hurdle_data](#)

`fit_demand_tmb()` for continuous mixed-effects models, `fit_demand_mixed()` for NLME-based fitting, `fit_demand_fixed()` for individual NLS curves.

Other demand-fitting: `fit_demand_fixed()`, `fit_demand_mixed()`, `fit_demand_tmb()`

Examples

```
# Load example data
data(apt)

# Fit full model with 3 random effects
fit3 <- fit_demand_hurdle(apt, y_var = "y", x_var = "x", id_var = "id",
  random_effects = c("zeros", "q0", "alpha"))

# Fit simplified model with 2 random effects (fixed alpha)
fit2 <- fit_demand_hurdle(apt, y_var = "y", x_var = "x", id_var = "id",
  random_effects = c("zeros", "q0"))

# View results
summary(fit3)

# Compare models with likelihood ratio test
compare_hurdle_models(fit3, fit2)
```

fit_demand_mixed

Fit Nonlinear Mixed-Effects Demand Model

Description

Fits a nonlinear mixed-effects model for behavioral economic demand data. The function allows Q0 and alpha parameters to vary by specified factors and supports different demand equation forms.

Usage

```
fit_demand_mixed(
  data,
  y_var,
  x_var,
  id_var,
  factors = NULL,
  factor_interaction = FALSE,
  equation_form = c("zben", "simplified", "exponentiated"),
  param_space = c("log10", "natural"),
  k = NULL,
  custom_model_formula = NULL,
  fixed_rhs = NULL,
  continuous_covariates = NULL,
```

```

    start_value_method = c("heuristic", "pooled_nls"),
    random_effects = Q0 + alpha ~ 1,
    covariance_structure = c("pdDiag", "pdSymm"),
    start_values = NULL,
    collapse_levels = NULL,
    nlme_control = list(msMaxIter = 200, niterEM = 100, maxIter = 200, pnlsTol = 0.001,
      tolerance = 1e-06, apVar = TRUE, minScale = 1e-09, opt = "nlminb", msVerbose = FALSE),
    method = "ML",
    ...
  )

```

Arguments

<code>data</code>	A data frame.
<code>y_var</code>	Character string, the name of the dependent variable column. For <code>equation_form = "zben"</code> , this should be log-transformed consumption (e.g., "y_ll4"). For <code>equation_form = "simplified"</code> or <code>"exponentiated"</code> , this should be raw, untransformed consumption (e.g., "y").
<code>x_var</code>	Character string, the name of the independent variable column (e.g., "x", price).
<code>id_var</code>	Character string, the name of the subject/group identifier column for random effects.
<code>factors</code>	A character vector of factor names (up to two) by which <code>Q0</code> and <code>alpha</code> are expected to vary (e.g., <code>c("dose", "treatment")</code>).
<code>factor_interaction</code>	Logical. If TRUE and two factors are provided, their interaction term is included in the fixed effects for <code>Q0</code> and <code>alpha</code> . Defaults to FALSE (additive effects).
<code>equation_form</code>	Character string specifying the demand equation form. Options are: "zben" (default): Assumes <code>y_var</code> is log-transformed. Equation: $y_var \sim Q0 * \exp(-(10^{\alpha} / Q0) * (10^{Q0}) * x_var)$. Model parameter <code>Q0</code> represents $\log_{10}(\text{True Max Consumption})$. Model parameter <code>alpha</code> represents $\log_{10}(\text{True Alpha Sensitivity})$. "simplified": Assumes <code>y_var</code> is raw (untransformed) consumption. Equation: $y_var \sim (10^{Q0}) * \exp(-(10^{\alpha}) * (10^{Q0}) * x_var)$. Model parameter <code>Q0</code> represents $\log_{10}(\text{True Max Consumption})$. Model parameter <code>alpha</code> represents $\log_{10}(\text{True Alpha Sensitivity})$. "exponentiated": Koffarnus et al. (2015) exponentiated equation. Assumes <code>y_var</code> is raw (untransformed) consumption. Requires the <code>k</code> parameter. Equation (log10 param_space): $y_var \sim (10^{Q0}) * 10^{(k * (\exp(-(10^{\alpha}) * (10^{Q0}) * x_var) - 1))}$. Equation (natural param_space): $y_var \sim Q0 * 10^{(k * (\exp(-\alpha * Q0 * x_var) - 1))}$.
<code>param_space</code>	Character. Parameterization used for fitting core demand parameters. One of: "log10": treat <code>Q0</code> and <code>alpha</code> as $\log_{10}(Q0)$ and $\log_{10}(\alpha)$ (default) "natural": treat <code>Q0</code> and <code>alpha</code> as natural-scale parameters

Notes:

- For `equation_form = "zben"`, only "log10" is currently supported.

	For <code>equation_form = "simplified"</code> or <code>"exponentiated"</code>, both <code>"log10"</code> and <code>"natural"</code> are supported.
<code>k</code>	Numeric. Range parameter (in log10 units) used with <code>equation_form = "exponentiated"</code> . If NULL (default), <code>k</code> is calculated from the data range: $\log_{10}(\max(y)) - \log_{10}(\min(y)) + 0.5$. Ignored for other equation forms.
<code>custom_model_formula</code>	An optional custom nonlinear model formula (nlme format). If provided, this overrides <code>equation_form</code> . The user is responsible for ensuring the <code>y_var</code> scale matches the formula and that starting values are appropriate. The formula should use parameters named <code>Q0</code> and <code>alpha</code> .
<code>fixed_rhs</code>	Optional one-sided formula or character string specifying the right-hand side (RHS) for the fixed-effects linear models of <code>Q0</code> and <code>alpha</code> . When provided, this RHS is used for both parameters and overrides <code>factors</code> , <code>factor_interaction</code> , and <code>continuous_covariates</code> for building the fixed-effects design matrix. Example: <code>"~ 1 + drug * dose + session"</code> .
<code>continuous_covariates</code>	Optional character vector of continuous (numeric) predictor names to be included additively in the fixed-effects RHS when <code>fixed_rhs</code> is NULL. These variables are not coerced to factors and are stored for downstream functions (e.g., plotting) to condition on.
<code>start_value_method</code>	Character, method to generate starting values if <code>start_values</code> is NULL. Options: <code>"heuristic"</code> (default, uses data-driven heuristics) or <code>"pooled_nls"</code> (fits a simpler pooled NLS model first; falls back to heuristic if NLS fails).
<code>random_effects</code>	A formula or a list of formulas for the random effects structure. Default <code>nlme::pdDiag(Q0 + alpha ~ 1)</code> .
<code>covariance_structure</code>	Character, covariance structure for random effects. Options: <code>"pdDiag"</code> (default) or <code>"pdSymm"</code> .
<code>start_values</code>	Optional named list of starting values for fixed effects. If NULL, defaults are estimated based on <code>equation_form</code> and <code>y_var</code> scale.
<code>collapse_levels</code>	Optional named list specifying factor level collapsing separately for <code>Q0</code> and <code>alpha</code> parameters. Structure: <pre>list(Q0 = list(factor_name = list(new_level = c(old_levels), ...)), alpha = list(factor_name = list(new_level = c(old_levels), ...)))</pre> Either <code>Q0</code> or <code>alpha</code> (or both) can be omitted to use original factor levels for that parameter. This allows different collapsing schemes for each parameter. Ignored if <code>fixed_rhs</code> is provided.
<code>nlme_control</code>	Control parameters for <code>nlme::nlme()</code> .
<code>method</code>	Fitting method for <code>nlme::nlme()</code> (<code>"ML"</code> or <code>"REML"</code>). Default <code>"ML"</code> .
<code>...</code>	Additional arguments passed to <code>nlme::nlme()</code> .

Value

An object of class `beezdemand_nlme` containing:

model The fitted nlme model object
data Data frame used for fitting
call The original function call
equation_form Equation form used
param_space Parameterization space
factors Factor variables in the model
y_var, x_var, id_var Variable name strings
start_values Starting values used

See Also

[fit_demand_tmb\(\)](#) for TMB-based mixed-effects (preferred for new work), [fit_demand_fixed\(\)](#) for individual NLS curves, [fit_demand_hurdle\(\)](#) for hurdle models, [get_demand_param_emms\(\)](#) for estimated marginal means, [get_demand_comparisons\(\)](#) for pairwise comparisons.

Other demand-fitting: [fit_demand_fixed\(\)](#), [fit_demand_hurdle\(\)](#), [fit_demand_tmb\(\)](#)

Examples

```
# Basic mixed-effects demand fit with apt data
# Transform consumption using LL4 for the zben equation
apt_ll4 <- apt |> dplyr::mutate(y_ll4 = ll4(y))

fit <- fit_demand_mixed(
  data = apt_ll4,
  y_var = "y_ll4",
  x_var = "x",
  id_var = "id",
  equation_form = "zben"
)
print(fit)
summary(fit)
```

fit_demand_tmb

Fit Mixed-Effects Demand Models via TMB

Description

Fits nonlinear mixed-effects demand models using Template Model Builder (TMB) for exact automatic differentiation and Laplace approximation. In practice this converges on demand equations where the PNLS-based `nlme::nlme()` fails (see Details).

Usage

```
fit_demand_tmb(
  data,
  y_var = "y",
  x_var = "x",
  id_var = "id",
  equation = c("exponentiated", "exponential", "simplified", "zben"),
  estimate_k = FALSE,
  k = NULL,
  random_effects = Q0 + alpha ~ 1,
  covariance_structure = c("pdSymm", "pdDiag"),
  factors = NULL,
  factor_interaction = FALSE,
  continuous_covariates = NULL,
  collapse_levels = NULL,
  start_values = NULL,
  tmb_control = list(iter_max = 1000, eval_max = 2000),
  multi_start = TRUE,
  validate_subject_pars = TRUE,
  verbose = 1,
  ...,
  store_report_cov = FALSE
)
```

Arguments

<code>data</code>	A data frame in long format with columns for subject ID, price, and consumption.
<code>y_var</code>	Character. Name of the consumption/response variable.
<code>x_var</code>	Character. Name of the price variable.
<code>id_var</code>	Character. Name of the subject identifier variable.
<code>equation</code>	Character. The demand equation to fit. One of: "exponentiated" Koffarnus et al. (2015). Gaussian on raw Q. Zeros allowed. Has k parameter. "exponential" Hursh & Silberberg (2008). Gaussian on log(Q). Observations with Q = 0 are automatically dropped. Has k parameter. "simplified" Simplified exponential (no k). Gaussian on raw Q. Zeros allowed. "zben" Zero-bounded exponential (no k). Gaussian on LL4- transformed Q. User must pass LL4-transformed y_var. Note: Q0 on the log10 scale is clamped to a minimum of 0.001 to avoid a singularity at Q0 = 1 (where log10(Q0) = 0 causes division by zero in the decay rate). Subjects with estimated Q0 near 1 may have biased parameter estimates.
<code>estimate_k</code>	Logical. If FALSE (default), k is held fixed at the value given in k. If TRUE, k is estimated as a free parameter; see Details for when the data support that. Only relevant for the "exponentiated" and "exponential" equations.

k	Numeric or NULL. Fixed value of k when estimate_k = FALSE. NULL (the default) means k = 2, the conventional value of Hursh & Silberberg (2008) and the default of <code>fit_demand_fixed()</code> .
random_effects	Specification of subject-level random effects. Accepts any of the following, in order of generality: formula (default) $Q0 + \alpha \sim 1$: random intercepts on both parameters (equivalent to the legacy <code>c("q0", "alpha")</code> shortcut). $Q0 \sim 1$ limits REs to $Q0$. Formulas with a factor-expanded RHS (e.g., $Q0 + \alpha \sim \text{condition}$ or $Q0 + \alpha \sim \text{condition} - 1$) are supported, giving each subject a random effect per factor level. The within-subject factor must vary within each id; pure between-subject factors belong in factors, not in the RE formula. continuous within-id covariate (random slope) A <i>numeric</i> RHS term such as $Q0 + \alpha \sim \text{dose_c}$ gives each subject a random <i>slope</i> on that covariate (dose-response demand). The covariate must vary within id for enough subjects and should be centered (and, for dose ladders, typically log10-transformed); see Details. Pair it with <code>continuous_covariates</code> to also estimate the population (fixed) dose slope. <code>nlme::pdMat</code> e.g., <code>nlme::pdDiag(Q0 + alpha ~ 1)</code> or <code>nlme::pdSymm(Q0 + alpha ~ condition)</code>. Pre-constructed <code>pdMat</code> objects are accepted and their covariance class is honored (overrides <code>covariance_structure</code>). list of <code>pdMat</code> / <code>nlme::pdBlocked</code> Multi-block covariance structures like <code>list(pdSymm(Q0+alpha~1), pdDiag(Q0+alpha~cond-1))</code> are fully supported. character vector (deprecated) <code>c("q0", "alpha")</code> or <code>"q0"</code>. Soft-deprecated in 0.3.0; emits a <code>lifecycle::deprecate_soft()</code> message. Translated internally to the formula $Q0 + \alpha \sim 1$ or $Q0 \sim 1$.
covariance_structure	"pdSymm" (default; unstructured) or "pdDiag" (diagonal). Applies only when <code>random_effects</code> is a formula; ignored for pre-constructed <code>pdMat</code> / <code>list</code> / <code>pdBlocked</code> inputs.
factors	Character vector of factor variable names for group comparisons.
factor_interaction	Logical. If TRUE and two factors provided, include their interaction.
continuous_covariates	Character vector of continuous covariate names entered as fixed (population) effects on $Q0$ and α . To also let the per-subject dose-response vary, add the same (centered) covariate as a random slope in <code>random_effects</code> (e.g. $Q0 + \alpha \sim \text{dose_c}$); the fixed and random parts are sourced separately and recovering the population dose slope requires both.
collapse_levels	Named list for asymmetric factor collapsing. Structure: <code>list(Q0 = list(factor = list(new = c(old))), alpha = list(...))</code> .
start_values	Named list of starting values. If NULL, data-driven defaults are used.
tmb_control	List of control parameters for the optimizer: optimizer Character. "nlminb" (default) or "L-BFGS-B". L-BFGS-B can sometimes recover where <code>nlminb</code> reports convergence code 1 with a "false

convergence (8)" message (R's `nlminb()` reports only codes 0/1; the PORT status is in `opt$message`).

rescue Logical (default TRUE). When `nlminb` exits with false convergence, retry automatically: restart `nlminb` from the stalled point, then L-BFGS-B from it, and, for a fixed-k fit, warm-start from a free-k refit. A candidate is accepted only if it reports convergence code 0, its gradient satisfies $\max(\text{abs}(\text{grad})) \leq \text{rescue_grad_tol}$, and its Hessian is positive definite; the lowest-NLL accepted candidate replaces the stalled result and `fit$opt$rescued_from` / `fit$opt$rescue_method` record it. If no candidate qualifies the fit fails loudly as before. See `vignette("convergence-guide")` for the worked example.

rescue_grad_tol Positive number (default $1e-2$): the maximum absolute gradient component a rescue candidate may have.

iter_max Maximum iterations (default 1000).

eval_max Maximum function evaluations (default 2000). Only applies to `nlminb`; L-BFGS-B has no function evaluation limit.

rel_tol Relative convergence tolerance (default $1e-10$). Only applies to `nlminb`.

lower Named numeric vector of lower bounds on optimizer-scale parameters (default NULL = no bounds). Names must match optimizer parameter names (e.g., `log_k`, `beta_q0`, `logsigma_b`). Note that most parameters are in log-space: e.g., to constrain `k` between 0.14 and 55, use `lower = c(log_k = -2)`, `upper = c(log_k = 4)` (which bind only when `estimate_k = TRUE`, since a fixed `k` is not a free parameter). A bound name applies to *all* occurrences of that parameter (e.g., both elements of `beta_q0`).

upper Named numeric vector of upper bounds (see `lower`).

warm_start Named numeric vector of starting values in optimizer space (e.g., from a previous `fit$opt$par`). When provided, `multi_start` is automatically disabled. This differs from `start_values`, which operates in parameter space before `TMB::MakeADFun()`. Length must match the number of free parameters.

trace Non-negative integer controlling optimizer trace output (default 0). When not explicitly set, inherits from `verbose >= 2`.

multi_start Logical. If TRUE (default), try 3 starting value sets and select the best.

validate_subject_pars Logical. If TRUE (default), validate that every column of the fixed-effect design matrices is constant within each `id` before computing `subject_pars`. When a factor or continuous covariate varies within subject, `Q0/alpha/Pmax/Omax` are set to `NA_real_` for affected subjects and a warning names the offending columns. Set to FALSE to force row-order-dependent values (not recommended; prefer a factor-expanded random-effects formula instead).

verbose Integer. Verbosity level: 0 = silent, 1 = progress, 2 = debug.

... Additional arguments (currently unused).

store_report_cov Logical. Advanced storage control. When FALSE (default), the full covariance matrix of all ADREPORT'd quantities (`$sdr$cov`) is not materialized, shrinking the saved fit substantially (often >80% on large datasets) with no loss of

functionality: no method reads it. Standard errors, `cov.fixed`, variance components, and all inference are identical either way. Set `TRUE` only if you need the full joint covariance of derived ADREPORT'd quantities.

Details

Traditional NLME approaches using `nlme::nlme()` universally fail for demand equations because the PNLs algorithm with numerical finite-difference gradients cannot navigate the likelihood surface. TMB succeeds using exact automatic differentiation, Laplace approximation, and joint marginal likelihood optimization.

Fixed versus estimated k. By default `k` is held at 2, the convention of Hursh & Silberberg (2008) and the default of `fit_demand_fixed()`. It is a convention rather than an estimate, so fits at a second value (say `k = 1.5` or `k = 3`) are worth reporting as a sensitivity check. `alpha` and the derived `Pmax / Omax / EV` move most; `Q0` enters the likelihood jointly with `alpha`, so it can shift too.

Setting `estimate_k = TRUE` estimates `k` alongside `Q0` and `alpha`, which fits better on data that carry the information to support it. Many do not. The response depends on `k` through $k(e^{-\alpha Q_0 C} - 1)$, so while $\alpha Q_0 C$ stays small the curve is a straight line of slope $k\alpha$ and only that product is identified. What pins `k` down is the curvature that appears as consumption approaches its floor. On data whose consumption never gets there, a free `k` can drift to arbitrarily large values with a compensating `alpha`, giving a non-positive-definite Hessian and meaningless `Pmax / Omax`. `check_demand_model()` screens a free-`k` fit for that signature and `summary()` carries the note. The screen reports what it detects; passing it is no evidence that `k` is identified.

Continuous within-subject random slopes (dose-response). A numeric term in the random-effects formula (e.g. `Q0 + alpha ~ dose_c`) gives each subject a random *slope* on a continuous within-id covariate, so intensity and elasticity change with the covariate (dose) at a subject-specific rate. The population (fixed) slope is sourced separately from `continuous_covariates`; recovering it requires both. The covariate must vary within `id` for enough subjects (a hard error below 2 informative subjects; a warning below 80\ ladders typically a centered `log10` dose) so the random intercept is the subject deviation at the reference value and the intercept/slope covariance is interpretable. No silent transform is applied: an uncentered covariate is still fit, but the intercept/slope correlation is reference-dependent and a warning is emitted. Per-subject parameters at a chosen covariate value are available via `get_subject_pars(fit, at = c(dose_c = value))` and `predict(fit, type = "parameters", at = ...)`; the per-subject slope deviations appear as `q0_<term> / alpha_<term>` columns there and in `ranef()`, and the variance components are labelled by the covariate term in `summary()` / `VarCorr()`. See `vignette("tmb-advanced-random-effects")` for a worked example.

Error model considerations: The exponentiated and simplified equations use a Gaussian error model on raw consumption (`Q`), which assigns non-zero density to negative values. For data with many near-zero observations, prefer exponential (Gaussian on `log Q`, zeros dropped) or `zben` (Gaussian on LL4-transformed `Q`, zeros handled by the transformation).

Random-effect variance components are reported by `summary()` on the `log10` scale; see `?summary.beezdemand_tmb` for the scale convention and its `nlme::VarCorr()` equivalence.

Value

An object of class `beezdemand_tmb` containing:

model List with coefficients, `se`, `variance_components`

subject_pars Data frame of subject-specific Q0, alpha, Pmax, Omax

tmb_obj TMB objective function object

opt Optimization result (normalized across optimizers)

sdr TMB sdreport object. Its \$cov (full covariance of all ADREPORT'd quantities) is not materialized (a scalar NA) unless store_report_cov = TRUE.

converged Logical convergence indicator

loglik Log-likelihood at convergence

AIC Akaike Information Criterion

BIC Bayesian Information Criterion

data Original data (after any filtering)

param_info List of model metadata

formula_details Design matrix and formula information

collapse_info Collapse levels information (if used)

See Also

[fit_demand_mixed\(\)](#) for NLME-based fitting, [fit_demand_hurdle\(\)](#) for two-part hurdle models, [fit_demand_fixed\(\)](#) for individual NLS curves.

Other demand-fitting: [fit_demand_fixed\(\)](#), [fit_demand_hurdle\(\)](#), [fit_demand_mixed\(\)](#)

Examples

```
data(apt)

# Exponential (HS) on log(Q)
fit <- fit_demand_tmb(apt, y_var = "y", x_var = "x", id_var = "id",
                     equation = "exponential")

summary(fit)
plot(fit)

# Exponentiated (Koffarnus) on raw Q
fit2 <- fit_demand_tmb(apt, y_var = "y", x_var = "x", id_var = "id",
                     equation = "exponentiated")

# With covariates (a 30-per-gender subset keeps the example fast)
data(apt_full)
ids <- unique(apt_full[c("id", "gender")])
ids <- ids[ids$gender %in% c("Male", "Female"), ]
keep <- unlist(lapply(split(ids$id, ids$gender), head, 30))
dat <- apt_full[apt_full$id %in% keep, ]
fit3 <- fit_demand_tmb(dat, y_var = "y", x_var = "x", id_var = "id",
                     equation = "exponential", factors = "gender")
get_demand_param_emms(fit3, param = "alpha")

# Factor-expanded random slopes on a within-subject factor are supported
# through the `random_effects` formula interface, e.g.
```

```
# random_effects = nlme::pdDiag(Q0 + alpha ~ cond)
# so each subject contributes a Q0 / alpha random effect per factor level.
# See vignette("tmb-advanced-random-effects", package = "beezdemand").
```

FitCurves

FitCurves

Description

Analyzes purchase task data

Usage

```
FitCurves(
  dat,
  equation,
  k,
  agg = NULL,
  detailed = FALSE,
  xcol = "x",
  ycol = "y",
  idcol = "id",
  groupcol = NULL,
  lobound,
  hibound,
  constrainq0 = NULL,
  startq0 = NULL,
  startalpha = NULL,
  param_space = c("natural", "log10")
)
```

Arguments

<code>dat</code>	data frame (long form) of purchase task data.
<code>equation</code>	Character vector of length one. Accepts either "hs" for Hursh and Silberberg (2008) or "koff" for Koffarnus, Franck, Stein, and Bickel (2015).
<code>k</code>	A numeric (or character) vector of length one. Reflects the range of consumption in log10 units. If none provided, k will be calculated based on the max/min of the entire sample + .5. If k = "ind", k will be calculated per individual using max/min + .5. If k = "fit", k will be a free parameter on an individual basis. If k = "range", k will be calculated based on the max/min of the entire sample + .5.
<code>agg</code>	Character vector of length one accepts either "Mean" or "Pooled". If not NULL (default), data will be aggregated appropriately and analyzed in the specified way.

detailed	If TRUE, output will be a 3 element list including (1) dataframe of results, (2) list of model objects, (3) list of individual dataframes used in fitting. Default value is FALSE, which returns only the dataframe of results.
xcol	The column name that should be treated as "x" data
ycol	The column name that should be treated as "y" data
idcol	The column name that should be treated as dataset identifier
groupcol	The column name that should be treated as the groups
lobound	Optional. A named vector of length 2 ("q0", "alpha") or 3 ("q0", "k", "alpha"), the latter length if k = "fit", specifying the lower bounds.
hibound	Optional. A named vector of length 2 ("q0", "alpha") or 3 ("q0", "k", "alpha"), the latter length if k = "fit", specifying the upper bounds.
constrainq0	Optional. A number that will be used to constrain Q0 in the fitting process. Currently experimental and only works with a fixed k value.
startq0	Optional. A number that will be used to start Q0 in the fitting process. Currently experimental.
startalpha	Optional. A number that will be used to start Alpha in the fitting process. Currently experimental.
param_space	Character. One of "natural" (default) or "log10". Specifies whether parameters (Q0, alpha) are estimated in natural space or log10-transformed space.

Details

[Superseded]

FitCurves() has been superseded by [fit_demand_fixed\(\)](#), which provides a modern S3 interface with standardized methods (summary(), tidy(), glance(), predict()). FitCurves() will continue to work but is no longer recommended for new code. See vignette("migration-guide") for migration instructions.

Value

If detailed == FALSE (default), a dataframe of results. If detailed == TRUE, a 3 element list consisting of (1) dataframe of results, (2) list of model objects, (3) list of individual dataframes used in fitting

Author(s)

Brent Kaplan bkaplan.ku@gmail.com Shawn Gilroy shawn.gilroy@temple.edu

See Also

[fit_demand_fixed\(\)](#) for the modern interface

Examples

```
## Analyze using Hursh & Silberberg, 2008 equation with a k fixed to 2
FitCurves(apt[sample(apt$id, 5), ], "hs", k = 2)
```

FitMeanCurves

*Fit Pooled/Mean Curves***Description**

Fits curve to pooled or mean data

Usage

```
FitMeanCurves(
  dat,
  equation,
  k,
  remq0e = FALSE,
  replfree = NULL,
  rem0 = FALSE,
  nrepl = NULL,
  replnum = NULL,
  plotcurves = FALSE,
  method = NULL,
  indpoints = TRUE,
  vartext = NULL
)
```

Arguments

<code>dat</code>	data frame (long form) of purchase task data.
<code>equation</code>	Character vector of length one. Accepts either "hs" for Hursh and Silberberg (2008) or "koff" for Koffarnus, Franck, Stein, and Bickel (2015).
<code>k</code>	A numeric vector of length one. Reflects the range of consumption in log10 units. If none provided, k will be calculated based on the max/min of the entire sample. If k = "fit", k will be a free parameter
<code>remq0e</code>	If TRUE, removes consumption and price where price == 0. Default value is FALSE
<code>replfree</code>	Optionally replaces price == 0 with specified value. Note, if fitting using equation == "hs", and 0 is first price, 0 gets replaced by replfree. Default value is .01
<code>rem0</code>	If TRUE, removes all 0s in consumption data prior to analysis. Default value is FALSE.
<code>nrepl</code>	Number of zeros to replace with replacement value (replnum). Can accept either a number or "all" if all zeros should be replaced. Default is to replace the first zero only.
<code>replnum</code>	Value to replace zeros. Default is .01
<code>plotcurves</code>	Boolean whether to create plot. If TRUE, a "plots/" directory is created one level above working directory. Default is FALSE.

method	Character string of length 1. Accepts "Mean" to fit to mean data or "Pooled" to fit to pooled data
indpoints	Boolean whether to plot individual points in gray. Default is TRUE.
vartext	Character vector specifying indices to report on plots. Valid indices include "Q0d", "Alpha", "Q0e", "EV", "Pmaxe", "Omaxe", "Pmaxd", "Omaxd", "K", "Q0se", "Alphase", "R2", "AbsSS"

Details

[Superseded]

FitMeanCurves() has been superseded by `fit_demand_fixed()` with the `agg` parameter. FitMeanCurves() will continue to work but is no longer recommended for new code. See `vignette("migration-guide")` for migration instructions.

Value

Data frame

Author(s)

Brent Kaplan bkaplan.ku@gmail.com

See Also

`fit_demand_fixed()` for the modern interface with `agg` parameter

Examples

```
## Fit aggregated data (mean only) using Hursh & Silberberg, 2008 equation with a k fixed at 2
FitMeanCurves(apt[sample(apt$id, 5), ], "hs", k = 2, method = "Mean")
```

```
fitted.beezdemand_hurdle
```

Fitted values for a beezdemand_hurdle fit

Description

Returns marginal expected consumption $P(y > 0 \mid x) * E(y \mid y > 0, x)$ by default (the `.fitted` column of `predict(fit, type = "demand")`). With `marginal = FALSE`, returns the conditional-on-positive expectation (the `.fitted` column of `predict(fit, type = "response")`).

Usage

```
## S3 method for class 'beezdemand_hurdle'
fitted(object, marginal = TRUE, ...)
```

Arguments

object	A beezdemand_hurdle object.
marginal	If TRUE (default), returns marginal expected consumption (type = "demand" in predict). If FALSE, returns conditional-on-positive consumption (type = "response").
...	Unused.

Value

Numeric vector of length nobs(object).

See Also

[predict.beezdemand_hurdle\(\)](#), [augment.beezdemand_hurdle\(\)](#).

fitted.beezdemand_tmb *Fitted values for a beezdemand_tmb fit*

Description

Default returns fitted values on the model's native likelihood scale (log scale for "exponential", natural/LL4 scale for others), matching `augment(fit)$fitted`. Set `scale = "natural"` to back-transform.

Usage

```
## S3 method for class 'beezdemand_tmb'
fitted(
  object,
  scale = c("model", "natural"),
  level = c("subject", "population"),
  ...
)
```

Arguments

object	A beezdemand_tmb object.
scale	One of "model" (default) or "natural".
level	One of "subject" (default; conditions on the subject random effects) or "population" (random effects set to zero, giving the population-mean values). See predict.beezdemand_tmb() .
...	Unused.

Value

Numeric vector of length nobs(object).

See Also

[predict.beezdemand_tmb\(\)](#), [augment.beezdemand_tmb\(\)](#), [residuals.beezdemand_tmb\(\)](#).

Examples

```
data(apt)
fit <- fit_demand_tmb(apt, equation = "exponential", verbose = 0)
head(fitted(fit))
```

fixed-demand	<i>Fixed-Effect Demand Curve Fitting</i>
--------------	--

Description

Modern wrapper for fitting individual demand curves via nonlinear least squares. Returns a structured S3 object with standard methods.

<code>fixef.beezdemand_nlme</code>	<i>Extract Fixed Effects from a beezdemand_nlme Model</i>
------------------------------------	---

Description

S3 method for `fixef` for objects of class `beezdemand_nlme`. Extracts the fixed-effect coefficients from the fitted `nlme` model.

Usage

```
## S3 method for class 'beezdemand_nlme'
fixef(object, ...)
```

Arguments

- `object` A `beezdemand_nlme` object.
- `...` Additional arguments passed to `nlme::fixef()`.

Value

A named numeric vector of fixed-effect coefficients.

See Also

[coef.beezdemand_nlme](#), [ranef.beezdemand_nlme](#)

fixef.beezdemand_tmb *Extract Fixed Effects from TMB Model*

Description

Extract Fixed Effects from TMB Model

Usage

```
## S3 method for class 'beezdemand_tmb'
fixef(object, ...)
```

Arguments

object	A beezdemand_tmb object.
...	Additional arguments.

Value

Named numeric vector of fixed effects.

Examples

```
data(apt)
fit <- fit_demand_tmb(apt, equation = "exponential", verbose = 0)
nlme::fixef(fit)
```

fixef.cp_model_lmer *Extract Fixed Effects from Mixed-Effects Cross-Price Model*

Description

Extract Fixed Effects from Mixed-Effects Cross-Price Model

Usage

```
## S3 method for class 'cp_model_lmer'
fixef(object, ...)
```

Arguments

object	A cp_model_lmer object
...	Additional arguments passed to fixef

Value

Named vector of fixed effects

formula.beezdemand_hurdle

Formula for a beezdemand_hurdle fit

Description

Returns the (currently intercept-only) component formulas plus the random-effect specification preserved at fit time. Hurdle does not yet support factor or covariate effects on its Part I (logit participation) or Part II (log-consumption intensity) components; both binary and consumption are ~ 1 . The API is intentionally aligned with [formula.beezdemand_tmb\(\)](#) so the shape extends without breaking when hurdle components gain factor/covariate support.

Usage

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

Arguments

x	A beezdemand_hurdle object.
...	Unused.

Value

Named list list(binary, consumption, random).

See Also

[model.matrix.beezdemand_hurdle\(\)](#).

formula.beezdemand_tmb

Formula for a beezdemand_tmb fit

Description

Returns the fixed-effect RHS formulas for Q0 and alpha plus the original random-effect specification preserved at fit time. The Q0 and alpha formulas may differ when collapse_levels was used to collapse factor levels asymmetrically.

Usage

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

Arguments

x A beezdemand_tmb object.
... Unused.

Value

Named list list(Q0, alpha, random). Q0 and alpha are one-sided formulas built from fit\$formula_details. random is the original random_effects argument value (round-trippable back to fit_demand_tmb()).

See Also

`model.matrix.beezdemand_tmb()`, `update.beezdemand_tmb()`.

Examples

```
data(apt)
fit <- fit_demand_tmb(apt, equation = "exponential", verbose = 0)
formula(fit)
```

get_demand_comparisons

Get Pairwise Comparisons for Demand Parameters

Description

Conducts pairwise comparisons for Q0 and/or alpha parameters from a beezdemand_nlme model across levels of specified factors. Comparisons are performed on the log10 scale of the parameters. Results include estimates of differences (on log10 scale) and optionally, ratios (on the natural scale by applying 10^difference).

Usage

```
get_demand_comparisons(fit_obj, ...)

## Default S3 method:
get_demand_comparisons(fit_obj, ...)

## S3 method for class 'beezdemand_nlme'
get_demand_comparisons(
  fit_obj,
  param = c("Q0", "alpha"),
```

```

compare_specs = NULL,
contrast_type = "pairwise",
contrast_by = NULL,
adjust = "holm",
at = NULL,
ci_level = 0.95,
report_ratios = TRUE,
params_to_compare = lifecycle::deprecated(),
...
)

```

Arguments

fit_obj	A beezdemand_nlme object.
...	Additional arguments passed to <code>emmeans::emmeans()</code> or <code>emmeans::contrast()</code> .
param	Character vector: "Q0", "alpha", or <code>c("Q0", "alpha")</code> . Default <code>c("Q0", "alpha")</code> (both). This is the canonical argument name, shared with the TMB backend (<code>get_demand_comparisons.beezdemand_tmb()</code>).
compare_specs	A formula specifying the factors whose levels are to be included in the EMM calculation prior to contrasting. This defines the "cells" of your design for EMMs. E.g., <code>~ factor1</code> (EMMs for levels of factor1, averaging over others), <code>~ factor1 * factor2</code> (EMMs for all cells of factor1 x factor2). If NULL, it defaults to an interaction of all factors in <code>fit_obj\$param_info\$factors</code> .
contrast_type	Character string specifying the type of contrast (passed to method in <code>emmeans::contrast</code>). Commonly "pairwise", "revpairwise", "eff", "consec", "poly". Default "pairwise".
contrast_by	Optional character vector of factor names to condition the contrasts by (passed to <code>by</code> in <code>emmeans::contrast</code>). If NULL (default), contrasts are performed over the primary terms implied by <code>compare_specs</code> and <code>contrast_type</code> . Example: If <code>compare_specs = ~ dose * drug</code> , <code>contrast_type = "pairwise"</code> , and <code>contrast_by = "dose"</code> , this will perform pairwise comparisons of drug levels within each level of dose. Note: If the original <code>fit_obj</code> model is additive for the factors involved (i.e., no interaction term was fitted), specifying <code>contrast_by</code> will result in identical contrast estimates across the levels of the <code>contrast_by</code> variable(s). In such cases, consider analyzing main effects directly (e.g., <code>compare_specs = ~drug</code> , <code>contrast_by = NULL</code>).
adjust	P-value adjustment method. Default "holm" (changed from "tukey" in 0.3.0 for cross-backend reproducibility; pass <code>adjust = "tukey"</code> to retain the previous default).
at	Optional named list for <code>emmeans::ref_grid()</code> .
ci_level	Confidence level. Default 0.95.
report_ratios	Logical. If TRUE, reports contrasts as ratios. Default TRUE.
params_to_compare	[Deprecated] Use <code>param</code> instead (deprecated in 0.3.0 to harmonize with the TMB backend).

Value

A list named by parameter. Each element contains:

- emmeans Tibble of EMMs (internal scale, i.e., log10 for param_space = "log10" fits, natural for param_space = "natural" fits) with CIs.
- contrasts_log10 Tibble of comparisons (differences on the fit's internal scale, despite the name; see the Details on param_space = "natural") with CIs and p-values.
- contrasts_ratio (If report_ratios=TRUE and successful) Tibble of comparisons with the same column shape (ratio_estimate, LCL_ratio, UCL_ratio) for both spaces, but different CONTENT (TICKET-075): for param_space = "log10" fits, a multiplicative ratio ($10^{\text{difference}}$, fold-change on the natural scale); for param_space = "natural" fits, the difference again (unchanged from contrasts_log10), because there is no log-scale quantity to exponentiate for an already-natural-scale difference. The returned object's contrasts_ratio_scale attribute is "ratio" or "difference" accordingly.

S3 class beezdemand_comparison is assigned. When contrast_by is active, the nested contrast tables carry leading by-column(s) named with the user-requested *original* factor name (e.g. dose, not the collapse-mapped dose_alpha), harmonized with the TMB backend and the flat [tidy\(\)](#) output.

Examples

```
data(ko, package = "beezdemand")
ko$y_ll4 <- ll4(ko$y, lambda = 4)
fit <- fit_demand_mixed(ko, y_var = "y_ll4", x_var = "x",
  id_var = "monkey", factors = "dose", equation_form = "zben")
get_demand_comparisons(fit)
```

```
get_demand_comparisons.beezdemand_tmb
```

Get Demand Parameter Comparisons for TMB Model

Description

Computes factor-level contrasts for demand parameters from a beezdemand_tmb model. Returns a classed beezdemand_comparison object (the same container the NLME backend returns), so [tidy.beezdemand_comparison\(\)](#) gives a backend-agnostic flat frame.

Usage

```
## S3 method for class 'beezdemand_tmb'
get_demand_comparisons(
  fit_obj,
  param = c("Q0", "alpha"),
```

```

compare_specs = NULL,
contrast_type = c("pairwise", "trt.vs.ctrl"),
contrast_by = NULL,
adjust = "holm",
at = NULL,
ci_level = 0.95,
report_ratios = TRUE,
...
)

```

Arguments

fit_obj	A beezdemand_tmb object.
param	Character vector. Which parameter(s) to compare: any of "Q0", "alpha". Default c("Q0", "alpha") (both).
compare_specs	Optional one-sided formula naming the factor subset to contrast (e.g. ~ gender). Omitted fitted factors are marginalized over with equal weights across the full crossing of their levels (matching the NLME backend). If NULL (default), all fitted factors are retained. Under asymmetric collapse_levels, name the original factor (e.g. ~ age_group); it resolves to that parameter's collapsed column (age_group_Q0 / age_group_alpha), as on the NLME backend.
contrast_type	Character. "pairwise" (all pairs, factor-level order) or "trt.vs.ctrl" (each level vs. the first/reference level).
contrast_by	Optional NULL (default) or character vector of factor name(s) within compare_specs to condition the contrasts on. Within each observed combination of by-level(s), pairwise (or trt.vs.ctrl) contrasts are computed over the remaining (non-by) factors, with p-value adjustment applied per by-cell . The by-variable(s) must be named in compare_specs (per parameter after collapse-mapping); a contrast_by factor absent from compare_specs aborts. Unlike compare_specs (which aborts on an unresolvable name), contrast_by soft-skips a parameter for which it does not resolve under asymmetric collapse_levels. Numeric results match the NLME backend in shape and direction (TMB uses asymptotic z vs. NLME's t). Continuous covariates are held at the global training mean within by-levels (the same convention as at), not recomputed per by-level. Multi-by is accepted but currently untested on TMB (its two-factor fixed-effect cap precludes a compare_specs with the 3+ factors a multi-by Cartesian would require).
adjust	Character. P-value adjustment method; must be one of stats::p.adjust.methods (default "holm"). emmeans-only methods such as "tukey"/"sidak" are rejected (the TMB backend uses asymptotic z + stats::p.adjust()).
at	Named list specifying factor levels and/or continuous-covariate values to condition on, as in get_demand_param_emms.beezdemand_tmb() .
ci_level	Numeric. Confidence level for intervals. Default 0.95.
report_ratios	Logical. If TRUE (default), include a contrasts_ratio block (multiplicative ratios) per parameter.
...	Additional arguments (reserved; factors_in_emm is accepted as a lower-level alternative to compare_specs).

Value

A `beezdemand_comparison` object: a list named by parameter, each element a list with `emmeans` (native cell means), `contrasts_log10` (log10-scale contrasts with `contrast`, `estimate`, `std.error`, `statistic`, `df`, `conf.low`, `conf.high`, `p.value`), and (if `report_ratios`) `contrasts_ratio`. When `contrast_by` is active, the contrast tables gain leading by-column(s) (user-requested original names) before contrast. Attributes `backend`, `adjustment_method`, `compare_specs_used`, `contrast_type_used`, `contrast_by_used`, and `contrast_by_map` (per-parameter original -> effective by-name map) describe the call.

See Also

`tidy.beezdemand_comparison()` for the backend-agnostic frame.

Examples

```
data(apt_full)
# 40 subjects per gender keep the example fast; use the full data in practice
ids <- unique(apt_full[,c("id", "gender")])
ids <- ids[ids$gender %in% c("Male", "Female"), ]
keep <- unlist(lapply(split(ids$id, ids$gender), head, 40))
dat <- apt_full[apt_full$id %in% keep, ]
fit <- fit_demand_tmb(dat, equation = "exponential",
                     factors = "gender", verbose = 0)
res <- get_demand_comparisons(fit, param = "Q0")
tidy(res)
```

`get_demand_param_emms` *Get Estimated Marginal Means for Demand Parameters*

Description

Calculates Estimated Marginal Means (EMMs) for Q0 and alpha parameters from a `beezdemand_nlme` model for all combinations of specified factor levels. Reports parameters on both their estimation scale (log10) and their natural, back-transformed scale. Optionally includes Essential Value (EV).

Usage

```
get_demand_param_emms(fit_obj, ...)

## Default S3 method:
get_demand_param_emms(fit_obj, ...)

## S3 method for class 'beezdemand_nlme'
get_demand_param_emms(
  fit_obj,
  factors_in_emm = NULL,
```

```

  at = NULL,
  ci_level = 0.95,
  include_ev = FALSE,
  param = c("both", "Q0", "alpha"),
  ...
)

```

Arguments

<code>fit_obj</code>	A <code>beezdemand_nlme</code> object.
<code>...</code>	Additional arguments passed to <code>emmeans::emmeans()</code> .
<code>factors_in_emm</code>	Character vector of factor names to compute EMMs over. Defaults to all factors present in the <code>fit_obj</code> .
<code>at</code>	Optional named list specifying levels of conditioning variables for <code>emmeans::ref_grid()</code> .
<code>ci_level</code>	Confidence level for the EMMs (default 0.95).
<code>include_ev</code>	Logical. If TRUE, calculates and includes Essential Value (EV) derived from alpha, along with its confidence interval (calculated by back-transforming the CI of <code>alpha_param_log10</code>). Default FALSE.
<code>param</code>	Character, one of "both" (default), "Q0", or "alpha". Controls which demand parameter's EMM columns are returned. "both" preserves the historical four-column-block structure (Q0 and alpha together). "Q0" returns only Q0 columns (and drops EV, since EV is a function of alpha); a warning is emitted if <code>include_ev = TRUE</code> is requested alongside <code>param = "Q0"</code> . "alpha" returns only alpha columns plus the EV block when <code>include_ev = TRUE</code> . Mirrors the <code>param</code> argument on the <code>beezdemand_tmb</code> method.

Value

A tibble containing:

Factor levels	Columns for each factor in <code>factors_in_emm</code> .
<code>Q0_param_log10</code> , <code>alpha_param_log10</code>	EMMs for the model parameters (log10 scale) with their respective confidence intervals (LCL_Q0_param, UCL_Q0_param, etc.).
<code>Q0_natural</code> , <code>alpha_natural</code>	EMMs back-transformed to the natural scale (10^{param}) with their respective confidence intervals (LCL_Q0_natural, UCL_Q0_natural, etc.).
<code>EV</code> , <code>LCL_EV</code> , <code>UCL_EV</code>	(If <code>include_ev=TRUE</code>) Essential Value and its CI.

When `param = "Q0"` or `param = "alpha"`, only the columns associated with the requested parameter (plus factor columns and, for "alpha", the EV block) are returned.

For a `param_space = "natural"` fit (see `fit_demand_mixed()`), the `emmeans` reference-grid summary IS already natural-scale, so `*_natural` columns are populated directly and `*_param_log10` columns are filled with `log10()` of them for column-set parity with `param_space = "log10"` fits. Because that fit is an unconstrained parameterization, a Wald CI bound (or, rarely, the point estimate itself) can be non-positive; `*_param_log10` is NA wherever the corresponding `*_natural` value is ≤ 0 (log10 is undefined there), without raising a warning. `*_natural` itself is never NA from this.

Examples

```
data(ko, package = "beezdemand")
ko$y_ll4 <- ll4(ko$y, lambda = 4)
fit <- fit_demand_mixed(ko, y_var = "y_ll4", x_var = "x",
  id_var = "monkey", factors = "dose", equation_form = "zben")
get_demand_param_emms(fit)

# Request only Q0 columns (convenient for pivoting and plotting)
get_demand_param_emms(fit, param = "Q0")
```

```
get_demand_param_emms.beezdemand_tmb
```

Get Demand Parameter Estimated Marginal Means for TMB Model

Description

Computes estimated marginal means (EMMs) for demand parameters from a beezdemand_tmb model. Uses design matrices and beta vectors with vcov from TMB::sdreport().

Usage

```
## S3 method for class 'beezdemand_tmb'
get_demand_param_emms(
  fit_obj,
  param = c("Q0", "alpha"),
  factors_in_emm = NULL,
  at = NULL,
  ci_level = 0.95,
  ...
)
```

Arguments

fit_obj	A beezdemand_tmb object.
param	Character. Which parameter to compute EMMs for: "Q0" or "alpha".
factors_in_emm	Character vector of factors to retain in the EMM reference grid. If it names a strict subset of the fitted factors, the omitted factors are marginalized over using equal weights across the full crossing of their levels (emmeans' default weights = "equal"), matching the NLME backend. If NULL (default), all fitted factors are retained (no marginalization). Under asymmetric collapse_levels you may name either the original factor or its collapsed per-parameter column; a name that resolves to neither for this parameter is rejected with an error.
at	Named list specifying factor levels and continuous-covariate values for conditional EMMs. For continuous covariates, a single numeric value per covariate; multiple values produce a warning and only the first is used. at on a marginalized (omitted) factor restricts the level set averaged over.

ci_level Numeric. Confidence level for intervals.
 ... Additional arguments.

Value

A tibble with columns: level, estimate, std.error, conf.low, conf.high.

Note

Marginalization is exact because Q_0/α are linear in the fixed-effect coefficients on the log scale, so averaging the reference-grid design rows and then multiplying by the coefficient vector equals averaging the per-cell parameter predictions.

Examples

```
data(apt_full)
# 40 subjects per gender keep the example fast; use the full data in practice
ids <- unique(apt_full[c("id", "gender")])
ids <- ids[ids$gender %in% c("Male", "Female"), ]
keep <- unlist(lapply(split(ids$id, ids$gender), head, 40))
dat <- apt_full[apt_full$id %in% keep, ]
fit <- fit_demand_tmb(dat, equation = "exponential",
  factors = "gender", verbose = 0)
get_demand_param_emms(fit, param = "Q0")
get_demand_param_emms(fit, param = "alpha")
```

```
get_demand_param_trends
```

Get Trends (Slopes) of Demand Parameters with respect to Continuous Covariates

Description

Computes the trend (slope) of Q_0 and/or α with respect to one or more continuous covariates using `emmeans::emtrends()` on a fitted `beezdemand_nlme` model. Trends are computed on the parameter estimation scale (log10), consistent with how parameters are modeled.

Usage

```
get_demand_param_trends(
  fit_obj,
  params = c("Q0", "alpha"),
  covariates,
  specs = ~1,
  at = NULL,
  ci_level = 0.95,
  ...
)
```

Arguments

<code>fit_obj</code>	A <code>beezdemand_nlm</code> object from <code>fit_demand_mixed()</code> .
<code>params</code>	Character vector of parameters to analyze: any of "Q0", "alpha". Default <code>c("Q0", "alpha")</code> .
<code>covariates</code>	Character vector of continuous covariate names for which to compute trends.
<code>specs</code>	A formula specifying the factors over which to produce trends (e.g., <code>~ drug</code> for trends by drug; <code>~ 1</code> for overall). Default <code>~ 1</code> .
<code>at</code>	Optional named list to condition variables (factors or continuous) when computing trends (passed through to <code>emmeans::ref_grid</code>).
<code>ci_level</code>	Confidence level for intervals. Default 0.95.
<code>...</code>	Additional args passed to <code>emmeans::emtrends()</code> .

Value

A tibble combining trends for each requested parameter and covariate, including columns for grouping factors (from `specs`), parameter, covariate, trend (slope on log10 scale), and its CI (lower.CL, upper.CL).

Examples

```
data(ko)
ko$dose_num <- as.numeric(as.character(ko$dose))
fit <- fit_demand_mixed(ko, y_var = "y_ll4", x_var = "x",
  id_var = "monkey", factors = "drug",
  equation_form = "zben")
trends <- get_demand_param_trends(fit, covariates = "dose_num",
  specs = ~ drug)
```

get_descriptive_summary

Calculate Descriptive Statistics by Price

Description

Calculates summary statistics for consumption data at each price point, including measures of central tendency (mean, median), variability (SD), range (min, max), and data quality (proportion of zeros, missing values).

This is the modern replacement for `GetDescriptives()`, returning a structured S3 object with dedicated methods for printing, summarizing, and visualizing.

Usage

```
get_descriptive_summary(
  data,
  x_var = "x",
  y_var = "y",
  id_var = "id",
  by = NULL
)
```

Arguments

<code>data</code>	A data frame in long format with columns for subject ID, price, and consumption
<code>x_var</code>	Character string specifying the column name for price (default: "x")
<code>y_var</code>	Character string specifying the column name for consumption (default: "y")
<code>id_var</code>	Character string specifying the column name for subject ID (default: "id")
<code>by</code>	Optional character vector of column names to group by. When supplied, statistics are computed separately within each unique combination of the by columns. Group columns are prepended to <code>\$statistics</code> and <code>\$data</code> . Default NULL (no grouping).

Details

For each unique price in the dataset, the function calculates:

- **Mean** - Average consumption across subjects (rounded to 2 decimals)
- **Median** - Median consumption (rounded to 2 decimals)
- **SD** - Standard deviation (rounded to 2 decimals)
- **PropZeros** - Proportion of subjects with zero consumption (0-1)
- **NAs** - Count of missing values
- **Min** - Minimum consumption value (rounded to 2 decimals)
- **Max** - Maximum consumption value (rounded to 2 decimals)

Value

An S3 object of class `beezdemand_descriptive` containing:

- **statistics** - Data frame with 8 columns (Price, Mean, Median, SD, PropZeros, NAs, Min, Max) and one row per unique price
- **call** - The matched call
- **data_summary** - List with `n_subjects`, `n_prices`, and `prices` vector

See Also

- [GetDescriptives\(\)](#) - Legacy function (superseded)
- [plot.beezdemand_descriptive\(\)](#) - Visualization method
- [summary.beezdemand_descriptive\(\)](#) - Extended summary

Examples

```
data(apt, package = "beezdemand")

# Calculate descriptive statistics
desc <- get_descriptive_summary(apt)
print(desc)

# View statistics table
desc$statistics

# Create visualization
plot(desc)

# Extended summary with distribution info
summary(desc)

# Grouped summary: statistics and plots faceted by group
data(apt_full)
desc_g <- get_descriptive_summary(apt_full, by = "gender")
desc_g$statistics
plot(desc_g)
```

get_empirical_measures

Calculate Empirical Demand Measures

Description

Calculates empirical (model-free) measures of demand from purchase task data. These metrics characterize consumption patterns without fitting a demand curve model.

This is the modern replacement for [GetEmpirical\(\)](#), returning a structured S3 object with dedicated methods for printing, summarizing, and visualizing.

Usage

```
get_empirical_measures(data, x_var = "x", y_var = "y", id_var = "id")
```

Arguments

data	A data frame in long format with columns for subject ID, price, and consumption
x_var	Character string specifying the column name for price (default: "x")
y_var	Character string specifying the column name for consumption (default: "y")
id_var	Character string specifying the column name for subject ID (default: "id")

Details

Empirical Measures:

- **Intensity** - The consumption value at the lowest price point. Reflects unrestricted demand or preferred level of consumption.
- **BP0** (Breakpoint 0) - The first price at which consumption drops to zero. If consumption never reaches zero, BP0 = NA. Indicates the price threshold where the commodity becomes unaffordable or undesirable.
- **BP1** (Breakpoint 1) - The highest price at which consumption is still non-zero. If all consumption is zero, BP1 = NA. Represents the upper limit of the commodity's value to the consumer.
- **Omaxe** (Empirical Omax) - The maximum observed expenditure across all prices (max of price $\times$ consumption). Represents peak spending on the commodity.
- **Pmaxe** (Empirical Pmax) - The price at which maximum expenditure occurs. If maximum expenditure is zero, Pmaxe = 0. If multiple prices have the same maximum expenditure, the highest price is returned.

Value

An S3 object of class `beezdemand_empirical` containing:

- **measures** - Data frame with one row per subject and columns:
 - *id* - Subject identifier
 - *Intensity* - Consumption at lowest price (demand intensity)
 - *BP0* - Breakpoint 0: first price where consumption = 0
 - *BP1* - Breakpoint 1: last price with non-zero consumption
 - *Omaxe* - Empirical Omax: maximum total expenditure (price $\times$ consumption)
 - *Pmaxe* - Empirical Pmax: price at which maximum expenditure occurs
- **call** - The matched call
- **data_summary** - List with `n_subjects`, `has_zeros`, and `complete_cases`

Note

- Data must not contain duplicate prices within a subject (will error)
- Missing values (NA) in consumption are preserved in calculations where applicable
- Breakpoints require at least one zero consumption value to be meaningful

See Also

- [GetEmpirical\(\)](#) - Legacy function (superseded)
- [plot.beezdemand_empirical\(\)](#) - Visualization method
- [summary.beezdemand_empirical\(\)](#) - Extended summary

Examples

```

data(apt, package = "beezdemand")

# Calculate empirical measures
emp <- get_empirical_measures(apt)
print(emp)

# View measures table
emp$measures

# Extended summary with distribution info
summary(emp)

# Visualize distribution of measures
plot(emp) # histogram by default
plot(emp, type = "matrix") # scatterplot matrix

```

```
get_hurdle_param_summary
```

Get Hurdle Model Parameter Summary

Description

Provides summary statistics for subject-level demand parameters from a hurdle demand model. This is analogous to EMMs but based on empirical Bayes estimates of subject-specific parameters.

Usage

```
get_hurdle_param_summary(fit_obj, ci_level = 0.95)
```

Arguments

fit_obj	A beezdemand_hurdle object.
ci_level	Confidence level for intervals (default 0.95).

Value

A data frame with summary statistics for each parameter:

- parameter** Parameter name
- mean** Mean across subjects
- sd** Standard deviation across subjects
- median** Median across subjects
- lcl** Lower confidence limit (based on percentiles)
- ucl** Upper confidence limit (based on percentiles)
- min** Minimum value
- max** Maximum value

See Also

[fit_demand_hurdle](#), [get_subject_pars](#)

Examples

```
data(apt)
fit <- fit_demand_hurdle(apt, y_var = "y", x_var = "x", id_var = "id")
get_hurdle_param_summary(fit)
```

```
get_individual_coefficients
```

Calculate Individual-Level Predicted Coefficients from beezdemand_nlme Model

Description

Extracts and combines fixed and random effects to calculate individual-level predicted coefficients for all parameter-factor combinations from a beezdemand_nlme model object. It automatically detects the factor structure and calculates coefficients for each individual and factor level.

Usage

```
get_individual_coefficients(
  fit_obj,
  params = c("Q0", "alpha"),
  format = c("wide", "long")
)
```

Arguments

fit_obj	A beezdemand_nlme object returned by fit_demand_mixed().
params	Character vector specifying which parameters to calculate. Options are "Q0", "alpha", or c("Q0", "alpha"). Default is c("Q0", "alpha").
format	Character, output format. "wide" returns one row per individual with separate columns for each parameter-factor combination. "long" returns one row per individual-parameter-factor combination. Default is "wide".

Details

Individual-level coefficients represent the predicted parameter values for each subject in the study. For models with factors, these coefficients combine:

1. The baseline intercept effect (fixed + random)
2. The factor-specific effect (fixed + random) for each factor level

This is equivalent to manually calculating: $\text{coefficient} = \text{intercept_fixed} + \text{intercept_random} + \text{factor_fixed} + \text{factor_random}$

The function automatically handles:

- Models with or without factors
- Any number of factor levels
- Missing random effects (defaults to 0)
- Complex factor structures with multiple factors

For models without factors, only intercept coefficients are calculated. For models with factors, both intercept and factor-level coefficients are provided.

Value

A data frame with individual-level predicted coefficients.

- In "wide" format: rows are individuals, columns are parameter-factor combinations
- In "long" format: columns are id, parameter, condition, coefficient_value

Column naming convention for wide format:

- `estimated_{param}_intercept`: Baseline/reference level coefficient
- `estimated_{param}_{factor}_{level}`: Factor level-specific coefficient

All coefficients are on the log10 scale (same as model estimation scale). To convert to natural scale, use $10^{\text{coefficient}}$.

See Also

[fit_demand_mixed](#) for fitting the original model [coef.beezdemand_nlme](#) for extracting model coefficients [get_demand_param_emms](#) for estimated marginal means

Examples

```
data(ko)
fit <- fit_demand_mixed(ko, y_var = "y_ll4", x_var = "x",
  id_var = "monkey", factors = "drug",
  equation_form = "zben")
individual_coefs <- get_individual_coefficients(fit)
head(individual_coefs)
```

get_k

*Calculate K Scaling Parameter for Demand Curve Fitting***Description**

Calculates the k scaling parameter used in demand curve equations to normalize consumption across different units or ranges. The k value is derived from the logarithmic range of consumption values.

This is the modern replacement for `GetK()`, with explicit parameters for the adjustment value and optional verbose output.

Usage

```
get_k(
  data,
  use_means = TRUE,
  adjustment = 0.5,
  x_var = "x",
  y_var = "y",
  verbose = FALSE
)
```

Arguments

data	A data frame in long format with columns for price and consumption
use_means	Logical indicating whether to calculate k from mean consumption by price (TRUE, default) or from individual consumption values (FALSE)
adjustment	Numeric adjustment added to the log range (default: 0.5). This value ensures k is slightly larger than the observed range.
x_var	Character string specifying the column name for price (default: "x")
y_var	Character string specifying the column name for consumption (default: "y")
verbose	Logical indicating whether to print calculation details (default: FALSE)

Details

The k parameter is calculated as:

$$k = \log_{10}(\max) - \log_{10}(\min) + \text{adjustment}$$

where max and min are the maximum and minimum non-zero consumption values.

Use in Demand Equations:

The k parameter appears in several demand curve equations:

- **Hursh & Silberberg (2008)**: Scales the exponential term
- **Koffarnus et al. (2015)**: Normalizes the exponentiated model

Calculation Modes:

- **use_means = TRUE** (default): Calculates k from mean consumption at each price point. Recommended when data has multiple subjects, as it reduces influence of individual outliers.
- **use_means = FALSE**: Calculates k from the full range of individual consumption values. May be preferable for single-subject data or when individual variability is theoretically important.

Value

A single numeric value representing the k scaling parameter

Note

- Only non-zero consumption values are used (zero values are excluded)
- Missing values (NA) are automatically removed via `na.rm = TRUE`
- The default adjustment of 0.5 is conventional but can be modified

See Also

- [GetK\(\)](#) - Legacy function (superseded)
- [FitCurves\(\)](#) - Uses k parameter in demand curve fitting

Examples

```
data(apt, package = "beezdemand")

# Calculate k using default settings (mean range + 0.5)
k_val <- get_k(apt)

# Calculate k from individual values
k_ind <- get_k(apt, use_means = FALSE)

# Calculate with custom adjustment
k_custom <- get_k(apt, adjustment = 1.0)

# Show calculation details
k_verbose <- get_k(apt, verbose = TRUE)
```

Description

Wraps `get_demand_param_emms`: it first computes Estimated Marginal Means (EMMs) for Q0 and alpha over all combinations of the specified factor levels, then filters these results to the combinations of factor levels that were actually present in the original dataset used to fit the `beezdemand_nlme` model.

Usage

```
get_observed_demand_param_emms(
  fit_obj,
  factors_in_emm = NULL,
  at = NULL,
  ci_level = 0.95,
  include_ev = FALSE,
  ...
)
```

Arguments

<code>fit_obj</code>	A <code>beezdemand_nlme</code> object returned by <code>fit_demand_mixed()</code> .
<code>factors_in_emm</code>	Character vector of factor names to compute EMMs over. Defaults to all factors present in the <code>fit_obj</code> . These factors define the grid over which EMMs are initially calculated and then filtered.
<code>at</code>	Optional named list specifying levels of conditioning variables for <code>emmeans::ref_grid()</code> . Passed to <code>get_demand_param_emms</code> .
<code>ci_level</code>	Confidence level for the EMMs (default 0.95). Passed to <code>get_demand_param_emms</code> .
<code>include_ev</code>	Logical. If TRUE, calculates and includes Essential Value (EV) derived from alpha. Passed to <code>get_demand_param_emms</code> . Default FALSE.
<code>...</code>	Additional arguments passed to <code>get_demand_param_emms</code> and subsequently to <code>emmeans::emmeans()</code> .

Value

A tibble similar to the output of `get_demand_param_emms`, but filtered to include only rows corresponding to factor level combinations that were observed in the original `fit_obj$data`. Contains:

<code>Factor levels</code>	Columns for each factor in <code>factors_in_emm</code> .
<code>Q0_param_log10</code> , <code>alpha_param_log10</code>	EMMs for model parameters (log10 scale) and CIs.
<code>Q0_natural</code> , <code>alpha_natural</code>	EMMs back-transformed to natural scale and CIs.
<code>EV</code> , <code>LCL_EV</code> , <code>UCL_EV</code>	(If <code>include_ev=TRUE</code>) Essential Value and its CI.

See Also

[get_demand_param_emms](#)

Examples

```
data(ko, package = "beezdemand")
ko$y_ll4 <- ll4(ko$y, lambda = 4)
fit <- fit_demand_mixed(ko, y_var = "y_ll4", x_var = "x",
  id_var = "monkey", factors = "dose", equation_form = "zben")
get_observed_demand_param_emms(fit)
```

get_subject_pars	<i>Get Subject-Specific Parameters</i>
------------------	--

Description

Convenience function to extract subject-specific demand parameters from a fitted demand model. For beezdemand_hurdle fits this is equivalent to accessing `object$subject_pars`. For beezdemand_tmb fits the return shape depends on the expanded argument: it may expand across within-id factor levels or condition within-id numeric covariates at the subject mean rather than return the cached `object$subject_pars`; see [get_subject_pars.beezdemand_tmb](#).

Usage

```
get_subject_pars(object, ...)
```

Arguments

<code>object</code>	A fitted beezdemand_hurdle or beezdemand_tmb object.
<code>...</code>	Method-specific arguments. The beezdemand_tmb method accepts an expanded argument (default NULL = auto-detect) that controls whether output is the wide one-row-per-subject shape or the long per-(subject, factor-level) shape; see get_subject_pars.beezdemand_tmb .

Value

Data frame with subject-specific parameters including:

- id** Subject identifier
- a_i** Random effect for Part I (zeros)
- b_i** Random effect for Part II (Q0)
- c_i** Random effect for alpha (3-RE model only)
- Q0** Subject-specific intensity (consumption at price 0)
- alpha** Subject-specific elasticity
- breakpoint** Price where $P(\text{quit}) = 0.5$
- Pmax** Price at maximum expenditure
- Omax** Maximum expenditure

See Also[fit_demand_hurdle](#)**Examples**

```
data(apt)
fit <- fit_demand_hurdle(apt, y_var = "y", x_var = "x", id_var = "id")
pars <- get_subject_pars(fit)
head(pars)
```

```
get_subject_pars.beezdemand_nlme
```

Get Subject-Specific Parameters from an NLME Demand Model

Description

Subject-level demand parameters for a beezdemand_nlme fit, matching the column / scale / expanded contract of [get_subject_pars.beezdemand_tmb](#). Combines the population fixed effects with each subject's random-effect deviations and back-transforms to the natural scale.

Usage

```
## S3 method for class 'beezdemand_nlme'
get_subject_pars(object, expanded = NULL, ...)
```

Arguments

object	A beezdemand_nlme object.
expanded	Controls the return shape for fits with within-id-varying design columns (within-subject factors, within-id covariates, or multi-block pdBlocked specs). • NULL (default): auto-detect. Expands to one row per (subject, factor-level) cell when within-id variation is present; otherwise returns the wide one-row-per-subject shape. • TRUE: always attempt expansion (no-op when there is no within-id variation). • FALSE: always return the wide shape; emits a one-line warning when within-id variation is present (the affected subjects' Q0, alpha, Pmax, Omax are NA).
...	Currently unused.

Value

A data frame. Wide form: id, b_i, c_i (if alpha has random effects), Q0, alpha, Pmax, Omax. Expanded form additionally includes the within-subject factor column(s) with one row per (subject, factor-level) cell. Q0, alpha, Pmax, Omax are on the natural scale.

Random-effect aliases (b_i / c_i)

b_i / c_i are the subject's first-block random-effect deviation for $Q0$ / α . For parity with the TMB method these are reported on the natural-log linear-predictor scale: for the default `param_space = "log10"` the stored log10 deviation is multiplied by $\log(10)$; for `param_space = "natural"` the deviation is returned on the natural parameter scale. The full per-coefficient random effects remain available via `ranef()`.

See Also

[get_subject_pars.beezdemand_tmb](#)

get_subject_pars.beezdemand_tmb

Get Subject-Specific Parameters from TMB Model

Description

Get Subject-Specific Parameters from TMB Model

Usage

```
## S3 method for class 'beezdemand_tmb'
get_subject_pars(object, expanded = NULL, at = NULL, ...)
```

Arguments

object	A beezdemand_tmb object.
expanded	Controls return shape for fits with within-id-varying design columns (factor-expanded random effects, within-id continuous covariates, or multi-block pdBlocked specs). • NULL (default): auto-detect. When fit-time within-id variation caused NA in cached <code>subject_pars\$Q0</code>, runs the expansion machinery: rows are expanded across within-id factor levels (one row per (subject, factor-level) cell), and within-id numeric covariates are conditioned at the subject's mean (no row expansion from numerics). When the cached <code>Q0</code> has no NA, returns the wide one-row-per-subject shape unchanged. • TRUE: always attempt expansion. On a fit with no within-id variation, silently returns the wide shape. • FALSE: always return the wide shape. Emits a one-line warning on a fit with within-id variation (the returned <code>Q0</code>, <code>alpha</code>, <code>Pmax</code>, <code>Omax</code> are NA).
at	Optional named numeric vector/list (e.g. <code>c(dose_c = 1)</code>) giving the covariate value(s) at which to evaluate per-subject <code>Q0/alpha</code> for continuous random-effect slope terms (TICKET-051). Defaults to each subject's mean of the covariate (which equals the reference 0 for a centered, balanced design). The per-subject slope deviations are always returned as <code>q0_<term> / alpha_<term></code> columns regardless of <code>at</code> . Ignored (with a warning) for fits without a continuous random slope.

... Additional arguments (currently unused).

Value

When the resolved expanded is FALSE: data frame with columns id, b_i, c_i (if 2 RE), Q0, alpha, Pmax, Omax. When the resolved expanded is TRUE, the shape depends on the kind of within-id variation: for fits with within-id factors, the within-subject factor columns are added and rows are expanded to one per (subject, factor-level) cell with per-cell Q0, alpha, Pmax, Omax; for fits whose only within-id variation is numeric, the numerics are conditioned at the subject's mean and the return is one row per subject (no added factor columns) with finite Q0 / alpha.

Per-block random-effect matrices

For factor-expanded or multi-block fits, the wide table's b_i / c_i columns hold the first RE column from each block (intercept slot for the M1 baseline block, for example) for backward compatibility with downstream consumers. Power users who need the full per-block RE structure can access attr(subject_pars, "re_q0_mat") and attr(subject_pars, "re_alpha_mat") as n_subjects x re_dim matrices ordered by block.

Examples

```
data(apt)
fit <- fit_demand_tmb(apt, equation = "exponential", verbose = 0)
head(get_subject_pars(fit))
```

GetAnalyticPmax	<i>Get pmax</i>
-----------------	-----------------

Description

...

Usage

```
GetAnalyticPmax(Alpha, K, Q0)
```

Arguments

Alpha	alpha parameter
K	k parameter (> lower limit)
Q0	Q0

Details

...

Value

Numeric

Author(s)

Shawn Gilroy sgilroy1@lsu.edu

Examples

```
GetAnalyticPmax(Alpha = 0.001, K = 3, Q0 = 10)
```

GetAnalyticPmaxFallback

Analytic Pmax Fallback

Description

Fallback method for Analytic Pmax

Usage

```
GetAnalyticPmaxFallback(K_, A_, Q0_)
```

Arguments

K_	k parameter
A_	alpha parameter
Q0_	q0 parameter

Details

Derivative-based optimization strategy

Value

numeric

Author(s)

Shawn Gilroy sgilroy1@lsu.edu

Examples

```
GetAnalyticPmaxFallback(K_ = 1, A_ = 0.001, Q0_ = 10)
```

GetDescriptives

*Get Purchase Task Descriptive Summary***Description**

Calculates descriptive statistics from purchase task data.

Usage

```
GetDescriptives(
  dat,
  bwplot = FALSE,
  outdir = "../plots/",
  device = "png",
  filename = "bwplot"
)
```

Arguments

<code>dat</code>	Dataframe (long form)
<code>bwplot</code>	Boolean. If TRUE, a ggplot2 box and whisker plot is saved. Default is FALSE.
<code>outdir</code>	Character. Directory where plot will be saved. Be sure to include trailing '/'. Default location is one level up in "../plots/".
<code>device</code>	Character. Type of file. Default is "png". Can be "pdf".
<code>filename</code>	Character. Specify filename. Default is "bwplot".

Details**[Superseded]**

GetDescriptives() has been superseded by [get_descriptive_summary\(\)](#), which provides a modern S3 interface with standardized methods (`print()`, `summary()`, `plot()`). GetDescriptives() will continue to work but is no longer recommended for new code.

Provides the following descriptive statistics from purchase task data at each price: mean consumption, median consumption, standard deviation of consumption, proportion of 0 values, number of NAs, minimum consumption, and maximum consumption.

Value

Dataframe with descriptive statistics

Author(s)

Brent Kaplan bkaplan.ku@gmail.com

See Also

[get_descriptive_summary\(\)](#) for the modern interface

Examples

```
GetDescriptives(apt)
```

GetEmpirical

GetEmpirical

Description

Calculates empirical measures for purchase task data

Usage

```
GetEmpirical(dat, xcol = "x", ycol = "y", idcol = "id")
```

Arguments

dat	data frame (long form) of purchase task data.
xcol	The column name that should be treated as "x" data
ycol	The column name that should be treated as "y" data
idcol	The column name that should be treated as dataset identifier

Details**[Superseded]**

GetEmpirical() has been superseded by [get_empirical_measures\(\)](#), which provides a modern S3 interface with standardized methods (print(), summary(), plot()). GetEmpirical() will continue to work but is no longer recommended for new code.

Will calculate and return the following empirical measures: Intensity, BP0, BP1, Omax, and Pmax

Value

Data frame of empirical measures

Author(s)

Brent Kaplan bkaplan.ku@gmail.com

See Also

[get_empirical_measures\(\)](#) for the modern interface

Examples

```
## Obtain empirical measures
GetEmpirical(apt)
```

`GetK`*Get K*

Description

Calculates a k value by looking for the max/min consumption across entire dataset and adds .5 to that range

Usage

```
GetK(dat, mnrange = TRUE)
```

Arguments

<code>dat</code>	Dataframe (long form)
<code>mnrange</code>	Boolean for whether k should be calculated based on the mean range + .5

Details

[Superseded]

`GetK()` has been superseded by `get_k()`, which provides explicit parameters for the adjustment value and optional verbose output for better transparency. `GetK()` will continue to work but is no longer recommended for new code.

Will look for maximum/minimum greater zero

Value

Numeric

Author(s)

Brent Kaplan bkaplan.ku@gmail.com

See Also

[get_k\(\)](#) for the modern interface

Examples

```
GetK(apt)
```

`GetSharedK`*Get Shared K*

Description

Finds shared k among selected datasets using global regression

Usage

```
GetSharedK(dat, equation, sharecol = "group")
```

Arguments

<code>dat</code>	Dataframe (longform)
<code>equation</code>	Character vector. Accepts either "hs" or "koff"
<code>sharecol</code>	Character for column to find shared k. Default to "group" but can loop based on id.

Details

Uses global regression to fit a shared k among datasets. Assumes the dataset is in its final form. Used within FitCurves

Value

Numeric value of shared k

Author(s)

Brent Kaplan bkaplan.ku@gmail.com Shawn P Gilroy shawn.gilroy@temple.edu

Examples

```
## Find a shared k value across datasets indicated by id  
GetSharedK(apt, "hs", sharecol = "id")
```

`GetValsForSim`*Get Values for SimulateDemand*

Description

Gets values used in SimulateDemand

Usage

```
GetValsForSim(dat)
```

Arguments

<code>dat</code>	Dataframe (long form)
------------------	-----------------------

Details

Gets values used in SimulateDemand. Per-subject residuals are matched to the returned `x` price columns by EXACT identity (`make.names(as.character(x))`) – no numeric tolerance), so a subject's price values must match the global price set bit-for-bit. A subject missing a price contributes NA for that price (tolerated by `sdindex`'s `na.rm = TRUE`); a subject with a duplicated price is rejected with an informative error naming the subject and the duplicated price(s), since duplicate rows have no well-defined single residual to place in that column.

Value

List of 3: `setaparams`, `sdindex`, `x`

Author(s)

Brent Kaplan bkaplan.ku@gmail.com

Examples

```
GetValsForSim(apt)
```

`glance.beezdemand_fixed`*Glance Method for beezdemand_fixed*

Description

Glance Method for beezdemand_fixed

Usage

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

Arguments

x	A beezdemand_fixed object
...	Additional arguments (ignored)

Value

A one-row tibble of model statistics

```
glance.beezdemand_fixed_grouped
```

Glance Method for beezdemand_fixed_grouped

Description

Returns one row per group with model-level summaries.

Usage

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

Arguments

x	A beezdemand_fixed_grouped object
...	Additional arguments passed to glance.beezdemand_fixed()

Value

A tibble with group columns prepended

```
glance.beezdemand_hurdle
```

Glance at a beezdemand_hurdle Model

Description

Returns a one-row tibble of model-level statistics.

Usage

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

Arguments

x An object of class beezdemand_hurdle.
... Additional arguments (currently unused).

Value

A one-row tibble with columns:

model_class "beezdemand_hurdle"

backend "TMB"

nobs Number of observations

n_subjects Number of subjects

n_random_effects Number of random effects

converged Convergence status

logLik Log-likelihood

AIC Akaike Information Criterion

BIC Bayesian Information Criterion

```
glance.beezdemand_nlme
```

Glance method for beezdemand_nlme

Description

Glance method for beezdemand_nlme

Usage

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

Arguments

x	A beezdemand_nlme object
...	Additional arguments (ignored)

Value

A one-row tibble of model statistics with columns:

- `model_class`: "beezdemand_nlme"
- `backend`: "nlme"
- `equation_form`: The equation form used
- `nobs`: Number of observations
- `n_subjects`: Number of subjects
- `n_random_effects`: Number of random-effect terms (e.g. 2 for $Q_0 + \alpha \sim 1$)
- `converged`: Operational convergence status. TRUE when the final fit is usable for inference, i.e., `apVar` (nlme's approximate covariance of the variance-covariance parameters) is positive-definite and there is no terminal error. Alias for `final_fit_ok`. It is not flipped to FALSE by iteration-level optimizer warnings (see `fit_warned`).
- `final_fit_ok`: The canonical usable-for-inference gate (`apVar` PD and no terminal error); identical to `converged`. NLME-only.
- `fit_warned`: Diagnostic flag that is TRUE when nlme emitted iteration-level convergence warnings (false convergence, singular, step-halving, iteration limit, ...) during PNLS-LME alternation. Informational only; does not gate `converged`. NLME-only.
- `logLik`, `AIC`, `BIC`: Model fit statistics
- `sigma`: Residual standard error (NLME-only)

The shared canonical columns (through `converged`, `logLik`, `AIC`, `BIC`) match `glance.beezdemand_tmb()`, so backend-agnostic code needs no dispatch glue; `final_fit_ok` and `fit_warned` are additive NLME-only diagnostics.

```
glance.beezdemand_systematicity
```

Glance Method for beezdemand_systematicity

Description

Glance Method for beezdemand_systematicity

Usage

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

Arguments

x	A beezdemand_systematicity object
...	Additional arguments (ignored)

Value

A one-row tibble of overall statistics

glance.beezdemand_tmb *Glance at a beezdemand_tmb Model*

Description

Glance at a beezdemand_tmb Model

Usage

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

Arguments

x	A beezdemand_tmb object.
...	Additional arguments.

Value

A one-row tibble of model-level statistics with columns:

- model_class: "beezdemand_tmb"
- backend: "TMB_mixed"
- equation_form: The demand equation that was fitted
- nobs: Number of observations
- n_subjects: Number of subjects
- n_random_effects: Total number of random-effect columns per subject
- converged: Convergence status
- logLik, AIC, BIC: Model fit statistics

The canonical columns match [glance.beezdemand_nlme\(\)](#), so backend-agnostic code needs no dispatch glue.

Examples

```
data(apt)
fit <- fit_demand_tmb(apt, equation = "exponential", verbose = 0)
glance(fit)
```

glance.cp_model_lm	<i>Get model summaries from a linear cross-price model</i>
--------------------	--

Description

Get model summaries from a linear cross-price model

Usage

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

Arguments

x	A cp_model_lm object.
...	Additional arguments (unused).

Value

A tibble with model summary statistics.

glance.cp_model_lmer	<i>Get model summaries from a mixed-effects cross-price model</i>
----------------------	---

Description

Get model summaries from a mixed-effects cross-price model

Usage

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

Arguments

x	A cp_model_lmer object.
...	Additional arguments passed to broom.mixed::glance.

Value

A tibble with model summary statistics: the columns of broom.mixed::glance() for the underlying merMod fit, followed by converged (logical; TRUE when lme4 reported no convergence problem, FALSE when it did, NA when the fit predates the stored metadata or failed). converged mirrors broom::glance.nls()'s isConv so a batch of fits can be screened programmatically; print() and summary() report the same flag.

glance.cp_model_nls	<i>Get model summaries from a cross-price model</i>
---------------------	---

Description

Extracts model summary statistics (goodness-of-fit measures and other model information) from a cross-price demand model into a single-row data frame, following the conventions of the broom package.

Usage

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

Arguments

x	A model object from fit_cp_nls or fit_cp_linear
...	Additional arguments (unused)

Value

A one-row data frame with model summary statistics:

r.squared	R-squared value indicating model fit
aic	Akaike Information Criterion
bic	Bayesian Information Criterion
equation	The equation type used in the model
method	The method used to fit the model
transform	The transformation applied to the data, if any

Examples

```
data(etm)  
fit <- fit_cp_nls(etm, equation = "exponentiated")  
glance(fit)
```

ko	<i>Example nonhuman demand data with drug and dose</i>
----	--

Description

Data from Ko et al. (2002)

Usage

ko

Format

A tibble with 135 rows and 6 columns:

- monkey** unique identifier
- x** fixed-ratio requirement
- y** consumption
- y_ll4** consumption transformed via ll4
- drug** drug
- dose** dose

lambertW	<i>Lambert W</i>
----------	------------------

Description

Ben Bolker’s port of Lambert W from GNU Scientific Library (GPLV3)

Usage

lambertW(z, b = 0, maxiter = 10, eps = .Machine\$double.eps, min.imag = 1e-09)

Arguments

- z input value
- b branch, set to principal by default
- maxiter Halley iteration count
- eps error precision
- min.imag minimum for imaginary solution

Details

Ben Bolker’s port of Lambert W from GNU Scientific Library

Value

numeric

Author(s)

Benjamin Bolker (port)

Examples

```
## Principal branch: W(1) ~ 0.5671
lambertW(1)

## Verify: W(z) * exp(W(z)) == z
w <- lambertW(2)
w * exp(w)
```

Description

Applies a log-logistic like transformation, specifically $\log_{\text{base}}(x^{\lambda} + 1) / \lambda$. This transformation is useful for compressing data that spans several orders of magnitude while remaining defined at zero ($x = 0$ yields 0). It is related to the Box-Cox transformation and to the generalized logarithm.

Usage

```
l14(x, lambda = 4, base = 10)
```

Arguments

<code>x</code>	A numeric vector or scalar of non-negative values to be transformed.
<code>lambda</code>	A positive numeric scalar, the lambda parameter of the transformation. Controls the curvature. Default is 4.
<code>base</code>	A positive numeric scalar, the base of the logarithm. Default is 10.

Value

A numeric vector or scalar of the transformed values. Returns NaN for $x < 0$ if lambda results in non-real numbers (e.g., even root of negative). However, the intended domain is $x \geq 0$.

Examples

```
# Basic usage
ll4(0)
ll4(1)
ll4(10)
ll4(c(0, 1, 10, 100, 1000))

# Using a different lambda or base
ll4(10, lambda = 2)
ll4(10, base = exp(1)) # Natural log base

# Typical workflow: transform -> fit -> back-transform
data(apt)
apt$y_ll4 <- ll4(apt$y)
# Verify round-trip: ll4_inv(ll4(x)) == x
all.equal(apt$y, ll4_inv(apt$y_ll4))
```

ll4_inv

Inverse Log-Logistic Transformation (Inverse LL4-like)

Description

Applies the inverse of the ll4 transformation. Given $y = \text{ll4}(x)$, this function calculates $x = (\text{base}^{(y * \text{lambda})} - 1)^{(1/\text{lambda})}$.

Usage

```
ll4_inv(y, lambda = 4, base = 10)
```

Arguments

y	A numeric vector or scalar of transformed values (output from ll4).
lambda	A positive numeric scalar, the lambda parameter used in the original ll4 transformation. Must match the one used for the forward transform. Default is 4.
base	A positive numeric scalar, the base of the logarithm used in the original ll4 transformation. Must match. Default is 10.

Details

Domain and boundary behavior. The inverse LL4 transformation is defined for $y \geq 0$ (when base = 10 and lambda = 4). For $y < 0$, the intermediate quantity $\text{base}^{(y * \text{lambda})} - 1$ becomes negative, and raising a negative number to the fractional power $1/\text{lambda}$ is undefined in real arithmetic. In this case, the function returns 0 (consumption cannot be negative).

This boundary condition arises in practice when a model predicts fitted values below zero on the LL4 scale, typically for extrapolation to very high prices. The mapping to zero is the natural floor because $\text{ll4}(0) = 0$ and the LL4 transformation is monotonically increasing on $[0, \text{Inf})$.

Value

A numeric vector or scalar of the original, untransformed values. Returns 0 when the intermediate quantity $\text{base}^y - 1$ is negative (i.e., when $y < 0$ for $\text{base} = 10$ and even λ), since consumption cannot be negative.

Examples

```
original_values <- c(0, 1, 10, 100, 1000)
transformed_values <- ll4(original_values)
back_transformed_values <- ll4_inv(transformed_values)
print(data.frame(original_values, transformed_values, back_transformed_values))
all.equal(original_values, back_transformed_values) # Should be TRUE or very close

# Negative y values are mapped to 0 (consumption floor)
ll4_inv(-0.5, lambda = 4, base = 10) # Returns 0
```

```
logLik.beezdemand_hurdle
```

Extract Log-Likelihood from Hurdle Demand Model

Description

Extracts the log-likelihood from a fitted hurdle demand model. Useful for likelihood ratio tests comparing nested models.

Usage

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

Arguments

<code>object</code>	An object of class <code>beezdemand_hurdle</code> .
<code>...</code>	Additional arguments (currently unused).

Value

An object of class `logLik` with the log-likelihood value and attributes for degrees of freedom and number of observations.

Examples

```
data(apt)
fit <- fit_demand_hurdle(apt, y_var = "y", x_var = "x", id_var = "id")
logLik(fit)
```

lowNicClean	<i>Low-nicotine cigarette purchase task</i>
-------------	---

Description

Long-form purchase task data across nicotine conditions.

Usage

```
lowNicClean
```

Format

A data frame with 5 columns: id, condition, x, y, commodity.

Examples

```
# data(lowNicClean)
```

model.matrix.beezdemand_hurdle	<i>Design matrices for a beezdemand_hurdle fit</i>
--------------------------------	--

Description

Hurdle currently has intercept-only Part I and Part II linear predictors, so `X_binary` and `X_consumption` are each a single column of ones with `nobs(fit)` rows. Returned for parity with `model.matrix.beezdemand_tmb()`. Future support for factor / covariate effects on hurdle components will enrich these matrices without changing the API.

Usage

```
## S3 method for class 'beezdemand_hurdle'
model.matrix(object, what = NULL, ...)
```

Arguments

<code>object</code>	A <code>beezdemand_hurdle</code> object.
<code>what</code>	NULL (default) returns the full named list. Otherwise one of "X_binary" or "X_consumption".
<code>...</code>	Unused.

Value

Named list of numeric matrices, or a single matrix when `what` is set.

```
model.matrix.beezdemand_tmb
```

Design matrices for a beezdemand_tmb fit

Description

By default returns a named list of all four design matrices the TMB template consumed: `X_q0`, `X_alpha`, `Z_q0`, `Z_alpha`. Use `what` to select a single matrix. `X_q0` and `X_alpha` are zero-copy references to `fit$formula_details`; `Z_q0` and `Z_alpha` are recomputed via the internal builder.

Usage

```
## S3 method for class 'beezdemand_tmb'
model.matrix(object, what = NULL, ...)
```

Arguments

<code>object</code>	A <code>beezdemand_tmb</code> object.
<code>what</code>	NULL (default) returns the full named list. Otherwise one of " <code>X_q0</code> ", " <code>X_alpha</code> ", " <code>Z_q0</code> ", " <code>Z_alpha</code> ".
<code>...</code>	Unused.

Details

Returning a named list (vs the single matrix `lm/lme4` return) is intentional: the TMB tier has two fixed-effect linear predictors (one per nonlinear parameter), not one.

Value

Named list of numeric matrices, or a single numeric matrix when `what` is set. NULL (with a message) when a degenerate `Z` is requested.

See Also

[formula.beezdemand_tmb\(\)](#).

Examples

```
data(apt)
fit <- fit_demand_tmb(apt, equation = "exponential", verbose = 0)
str(model.matrix(fit))
head(model.matrix(fit, what = "X_q0"))
```

nobs.beezdemand_hurdle

Sample size for a beezdemand_hurdle fit

Description

Universal-accessor parity with `nobs.beezdemand_tmb()` and the cross-price classes. `broom::glance(fit)$nobs` and `BIC(fit)` were already correct via their own paths (`param_info$n_obs` and the `nobs` attribute on `logLik()`, respectively); this method closes the gap for any caller that consumes `nobs()` directly.

Usage

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

Arguments

<code>object</code>	A <code>beezdemand_hurdle</code> object.
<code>...</code>	Unused.

Value

Integer scalar, the number of observations the model was fit on.

nobs.cp_model_lm

Number of Observations in a Cross-Price Demand Model (Linear)

Description

Number of Observations in a Cross-Price Demand Model (Linear)

Usage

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

Arguments

<code>object</code>	A <code>cp_model_lm</code> object.
<code>...</code>	Additional arguments (unused).

Value

Integer; observation count or `NA_integer_` if unavailable.

nobs.cp_model_lmer	<i>Number of Observations in a Cross-Price Demand Model (Mixed-Effects)</i>
--------------------	---

Description

Number of Observations in a Cross-Price Demand Model (Mixed-Effects)

Usage

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

Arguments

object	A cp_model_lmer object.
...	Additional arguments (unused).

Value

Integer; observation count or NA_integer_ if unavailable.

nobs.cp_model_nls	<i>Number of Observations in a Cross-Price Demand Model (Nonlinear)</i>
-------------------	---

Description

Number of Observations in a Cross-Price Demand Model (Nonlinear)

Usage

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

Arguments

object	A cp_model_nls object.
...	Additional arguments (unused).

Value

Integer; observation count or NA_integer_ if unavailable.

ongoingETM	<i>Experimental Tobacco Marketplace (ETM) data</i>
------------	--

Description

ETM data across price points with product quantities.

Usage

ongoingETM

Format

A data frame with 6 columns: id, x, AdjCig, FixCig, ECig, flavor.

Examples

```
# data(ongoingETM)
```

palette_beezdemand	<i>beezdemand Color Palette</i>
--------------------	---------------------------------

Description

beezdemand Color Palette

Usage

palette_beezdemand()

Value

Named vector of brand colors.

Examples

```
palette_beezdemand()
```

`pivot_demand_data` *Reshape Demand Data Between Wide and Long Formats*

Description

Converts demand data between wide format (one row per subject, prices as columns) and long format (one row per observation with id, x, y columns). This is a convenience wrapper around `tidyr::pivot_longer()` and `tidyr::pivot_wider()` tailored for behavioral economic purchase task data.

Usage

```
pivot_demand_data(
  data,
  format = c("long", "wide"),
  id_var = "id",
  x_var = "x",
  y_var = "y",
  price_cols = NULL,
  x_values = NULL,
  drop_na = TRUE
)
```

Arguments

<code>data</code>	A data frame or tibble to reshape.
<code>format</code>	Character. Direction of reshaping: "long" (wide to long) or "wide" (long to wide). Default "long".
<code>id_var</code>	Character. Name of the subject/series identifier column. Default "id".
<code>x_var</code>	Character. Name of the price column in long-format data. Used when format = "wide". Default "x".
<code>y_var</code>	Character. Name of the consumption column in long-format data. Used when format = "wide". Default "y".
<code>price_cols</code>	Character vector of column names in wide-format data that represent prices. Used when format = "long". Default NULL, which auto-detects: columns whose names parse as numeric become price columns, and remaining columns (besides <code>id_var</code>) are preserved as identifiers. If no column names parse as numeric, all non- <code>id_var</code> columns are treated as price columns (and <code>x_values</code> must be supplied).
<code>x_values</code>	Numeric vector of actual price values corresponding to each price column in wide-format data. Used when format = "long". Must be the same length as the number of price columns. Default NULL, which parses prices from column names.
<code>drop_na</code>	Logical. When format = "long", drop rows where consumption (y) is NA after pivoting? Default TRUE. A warning is issued when rows are dropped.

Details

Wide to Long (format = "long"):

The function determines which columns are price columns by:

1. If price_cols is provided, those columns are used directly.
2. If price_cols is NULL, column names are tested with `as.numeric()`. Columns whose names successfully parse as numbers (e.g., "0", "0.5", "10") are treated as price columns. Remaining non-id_var columns are preserved as extra identifiers.
3. If no column names parse as numeric, all non-id_var columns become price columns and x_values must be supplied.

Actual numeric price values come from x_values if supplied, or from parsing column names. If column names cannot be parsed and x_values is not supplied, an error is raised.

Long to Wide (format = "wide"):

Pivots long data so that each unique value in x_var becomes a column, with values from y_var. All columns except x_var and y_var are used as identifiers.

Value

A tibble. When format = "long": columns id, any extra identifier columns, x (numeric price), and y (numeric consumption). When format = "wide": one row per subject with prices as column names.

Examples

```
# --- Wide to long ---
# Columns named as prices (auto-parsed)
wide_num <- data.frame(
  id = 1:3,
  "0" = c(10, 8, 12), "0.5" = c(9, 7, 11), "1" = c(8, 6, 9),
  check.names = FALSE
)
pivot_demand_data(wide_num, format = "long")

# Columns with non-numeric names require x_values
wide_named <- data.frame(id = 1:2, price_1 = c(10, 8), price_2 = c(5, 4))
pivot_demand_data(wide_named, format = "long",
  x_values = c(0, 0.5))

# --- Long to wide ---
data(apt, package = "beezdemand")
wide_apt <- pivot_demand_data(apt, format = "wide")
head(wide_apt)
```

plot-theme

beezdemand Plot Theme and Color Palette

Description

Central plotting style API for beezdemand, with a modern default based on the codedbx "Refined Contemporary" brand and an APA alternative.

plot.beezdemand_fixed *Plot Method for beezdemand_fixed*

Description

Plot Method for beezdemand_fixed

Usage

```
## S3 method for class 'beezdemand_fixed'
plot(
  x,
  type = c("demand", "population", "individual", "both"),
  ids = NULL,
  style = c("modern", "apa"),
  show_observed = TRUE,
  show_pred = NULL,
  x_trans = c("log10", "log", "linear", "pseudo_log"),
  y_trans = NULL,
  free_trans = 0.01,
  x_limits = NULL,
  y_limits = NULL,
  n_points = 200,
  x_lab = NULL,
  y_lab = NULL,
  xlab = NULL,
  ylab = NULL,
  facet = NULL,
  observed_point_alpha = 0.5,
  observed_point_size = 1.8,
  pop_line_alpha = 0.9,
  pop_line_size = 1,
  ind_line_alpha = 0.35,
  ind_line_size = 0.7,
  subtitle = NULL,
  ...
)
```

Arguments

<code>x</code>	A beezdemand_fixed object.
<code>type</code>	Plot type: "demand", "population", "individual", or "both".
<code>ids</code>	Optional vector of subject IDs to plot. Defaults to all subjects.
<code>style</code>	Plot styling, passed to theme_beezdemand().
<code>show_observed</code>	Logical; if TRUE, overlay observed data points where possible.
<code>show_pred</code>	Which prediction layers to plot: "population", "individual", or "both".
<code>x_trans</code>	X-axis transform: "log", "log10", "linear", or "pseudo_log".
<code>y_trans</code>	Y-axis transform: "log", "log10", "linear", or "pseudo_log".
<code>free_trans</code>	Value used to display free ($x = 0$) on log scales. Use NULL to drop $x \leq 0$ values instead.
<code>x_limits</code>	Optional numeric vector of length 2 for x-axis limits.
<code>y_limits</code>	Optional numeric vector of length 2 for y-axis limits.
<code>n_points</code>	Number of points to use for prediction curves when thinning.
<code>x_lab</code>	Optional x-axis label.
<code>y_lab</code>	Optional y-axis label.
<code>xlab</code>	Deprecated alias for <code>x_lab</code> .
<code>ylab</code>	Deprecated alias for <code>y_lab</code> .
<code>facet</code>	Faceting specification (TRUE for <code>~id</code> or a formula).
<code>observed_point_alpha</code>	Alpha for observed points.
<code>observed_point_size</code>	Size for observed points.
<code>pop_line_alpha</code>	Alpha for population curve.
<code>pop_line_size</code>	Line size for population curve.
<code>ind_line_alpha</code>	Alpha for individual curves.
<code>ind_line_size</code>	Line size for individual curves.
<code>subtitle</code>	Optional subtitle for the plot.
<code>...</code>	Additional arguments (currently unused).

Value

A ggplot2 object.

plot.beezdemand_fixed_grouped

Plot Method for beezdemand_fixed_grouped

Description

Calls `plot()` on each per-group child and combines with `patchwork::wrap_plots()` if available, otherwise returns a list.

Usage

```
## S3 method for class 'beezdemand_fixed_grouped'
plot(x, ...)
```

Arguments

`x` A `beezdemand_fixed_grouped` object
`...` Additional arguments passed to `plot.beezdemand_fixed()`

Value

A combined patchwork plot if patchwork is available, otherwise a named list of ggplot objects

plot.beezdemand_hurdle

Plot Demand Curves from Hurdle Demand Model

Description

Creates visualizations of fitted demand curves from a hurdle demand model.

Usage

```
## S3 method for class 'beezdemand_hurdle'
plot(
  x,
  type = c("demand", "population", "probability", "parameters", "individual", "both"),
  ids = NULL,
  subjects = NULL,
  parameters = c("Q0", "alpha", "breakpoint", "Pmax", "Omax"),
  prices = NULL,
  show_population = TRUE,
  show_pred = NULL,
  show_observed = TRUE,
  x_trans = c("log10", "log", "linear", "pseudo_log"),
```

```

y_trans = NULL,
free_trans = 0.01,
facet = NULL,
x_limits = NULL,
y_limits = NULL,
x_lab = NULL,
y_lab = NULL,
xlab = NULL,
ylab = NULL,
style = c("modern", "apa"),
observed_point_alpha = 0.5,
observed_point_size = 1.8,
pop_line_alpha = 0.9,
pop_line_size = 1,
ind_line_alpha = 0.35,
ind_line_size = 0.7,
marginal = TRUE,
marginal_method = c("normal", "kde", "empirical"),
par_trans = NULL,
...
)

```

Arguments

x	An object of class beezdemand_hurdle.
type	Character string specifying the plot type: "demand" Predicted demand curves (default) "population" Alias for "demand" "probability" Probability of zero consumption "parameters" Distribution of subject-specific parameters "individual" Individual demand curves for selected subjects
ids	Optional vector of subject IDs to plot (alias of subjects).
subjects	Character or numeric vector of subject IDs to plot for type = "individual". If NULL, plots first 9 subjects.
parameters	Character vector specifying which parameters to plot when type = "parameters". Options are: "Q0", "alpha", "breakpoint", "Pmax", "Omax". Default is all five.
prices	Numeric vector of prices for plotting. If NULL, uses a sequence from 0 to max observed price.
show_population	Logical; if TRUE, overlay population-level curve on individual plots. Default is TRUE.
show_pred	Which prediction layers to plot: "population", "individual", or "both".
show_observed	Logical; if TRUE, overlay observed data points.
x_trans	Character. Transformation for x-axis. Default "log".

y_trans	Character. Transformation for y-axis. Default "log".
free_trans	Value used to display free ($x = 0$) on log scales. Use NULL to drop $x \leq 0$ values instead.
facet	Faceting specification (TRUE for <code>~id</code> or a formula).
x_limits	Optional numeric vector of length 2 for x-axis limits.
y_limits	Optional numeric vector of length 2 for y-axis limits.
x_lab	Optional x-axis label.
y_lab	Optional y-axis label.
xlab	Deprecated alias for x_lab.
ylab	Deprecated alias for y_lab.
style	Plot styling, passed to <code>theme_beezdemand()</code> .
observed_point_alpha	Alpha for observed points.
observed_point_size	Size for observed points.
pop_line_alpha	Alpha for population curve.
pop_line_size	Line size for population curve.
ind_line_alpha	Alpha for individual curves.
ind_line_size	Line size for individual curves.
marginal	Logical; if TRUE (default) and <code>type = "probability"</code> , the population curve shows the marginal (population-averaged) $P(\text{zero})$ instead of the conditional ($RE = 0$) curve. Set to FALSE for the old conditional behavior.
marginal_method	Character. Method for marginal integration when <code>marginal = TRUE</code> . One of "normal" (default), "kde", or "empirical". See predict.beezdemand_hurdle() for details.
par_trans	Named list of transformations for parameter distribution plots (when <code>type = "parameters"</code>). Names are parameter names (e.g., "alpha"), values are transformation names (e.g., "log10"). Default applies log10 to alpha.
...	Additional arguments (currently unused).

Value

A ggplot2 object.

Examples

```
data(apt)
fit <- fit_demand_hurdle(apt, y_var = "y", x_var = "x", id_var = "id")

# Plot mean demand curve
plot(fit)

# Plot parameter distributions
plot(fit, type = "parameters")
```

plot.beezdemand_nlme *Plot Method for beezdemand_nlme Objects*

Description

Creates a ggplot2 visualization of a fitted beezdemand_nlme model, showing observed data points and/or model prediction lines.

Usage

```
## S3 method for class 'beezdemand_nlme'
plot(
  x,
  type = c("demand", "population", "individual", "both"),
  ids = NULL,
  show_observed = TRUE,
  observed_point_alpha = 0.6,
  show_pred = "population",
  n_points = 200,
  inv_fun = identity,
  facet = NULL,
  at = NULL,
  color_by = NULL,
  linetype_by = NULL,
  shape_by = NULL,
  x_trans = c("log10", "log", "linear", "pseudo_log"),
  y_trans = NULL,
  free_trans = 0.01,
  x_limits = NULL,
  y_limits = NULL,
  style = c("modern", "apa"),
  title = NULL,
  subtitle = NULL,
  x_lab = NULL,
  y_lab = NULL,
  xlab = NULL,
  ylab = NULL,
  observed_point_size = 2,
  pop_line_size = 1,
  ind_line_size = 0.6,
  pop_line_alpha = 0.9,
  ind_line_alpha = 0.3,
  ...
)
```

Arguments

x A beezdemand_nlme object.

type	Plot type: "demand", "population", "individual", or "both".
ids	Optional vector of subject IDs to plot.
show_observed	Logical. If TRUE, plots the original data points. Default TRUE.
observed_point_alpha	Alpha for observed points. Default 0.6.
show_pred	Which prediction layers to plot: "population", "individual", or "both".
n_points	Integer. Number of points for prediction lines. Default 100.
inv_fun	Optional function to inverse-transform y-axis and predictions. Default identity.
facet	Optional faceting formula (e.g., ~ dose).
at	Optional named list giving values for continuous covariates used in the fixed-effects RHS. When building prediction grids for population- or individual- level lines, these values will be used. If not provided, the function will default to the median of each continuous covariate found in the original model data. Factor variables are always handled as grids (population) or observed combinations (individual) as before.
color_by	Optional character string: name of a factor to color lines and/or points by. Must be a column in x\$data.
linetype_by	Optional character string: name of a factor for linetypes of population prediction lines if individual lines are also shown (otherwise applies to the shown lines). Must be a model factor in x\$param_info\$factors.
shape_by	Optional character string: name of a factor for shapes of observed points. Must be a column in x\$data.
x_trans	Character. Transformation for x-axis. Default "log".
y_trans	Character. Transformation for y-axis. Default "log".
free_trans	Value used to display free ($x = 0$) on log scales. Use NULL to drop $x \leq 0$ values instead.
x_limits	Optional numeric vector of length 2 for x-axis limits.
y_limits	Optional numeric vector of length 2 for y-axis limits.
style	Plot styling, passed to theme_beezdemand().
title	Optional plot title.
subtitle	Optional subtitle for the plot.
x_lab	Optional x-axis label.
y_lab	Optional y-axis label.
xlab	Deprecated alias for x_lab.
ylab	Deprecated alias for y_lab.
observed_point_size	Size for observed points. Default 2.
pop_line_size	Size for population prediction lines. Default 1.
ind_line_size	Size for individual prediction lines. Default 0.6.
pop_line_alpha	Alpha for population prediction lines. Default 0.9.
ind_line_alpha	Alpha for individual prediction lines. Default 0.3.
...	Additional arguments (currently unused).

Value

A ggplot2 object.

plot.beezdemand_tmb	<i>Plot TMB Mixed-Effects Demand Model</i>
---------------------	--

Description

Plot TMB Mixed-Effects Demand Model

Usage

```
## S3 method for class 'beezdemand_tmb'
plot(
  x,
  type = c("demand", "individual", "parameters"),
  ids = NULL,
  prices = NULL,
  show_population = TRUE,
  show_observed = TRUE,
  show_pred = NULL,
  x_trans = c("log10", "log", "linear", "pseudo_log"),
  y_trans = NULL,
  inv_fun = NULL,
  x_limits = NULL,
  y_limits = NULL,
  x_lab = NULL,
  y_lab = NULL,
  style = c("modern", "apa"),
  observed_point_alpha = 0.3,
  observed_point_size = 1.5,
  pop_line_alpha = 1,
  pop_line_size = 1.2,
  ind_line_alpha = 0.3,
  ind_line_size = 0.5,
  ...
)
```

Arguments

x	A beezdemand_tmb object.
type	Character. One of "demand" (population curve with data), "individual" (per-subject curves), "parameters" (parameter distributions).
ids	Character vector of subject IDs to plot (for individual type).
prices	Optional numeric vector of prices for curve generation.

show_population	Logical. Show population curve overlay.
show_observed	Logical. Show observed data points.
show_pred	Character. Which predictions to show: "population", "individual", or "both". If NULL (default), determined by type.
x_trans	Character. X-axis transformation.
y_trans	Character. Y-axis transformation. If NULL (default), uses "pseudo_log", which is defined at zero.
inv_fun	Optional function to back-transform y-axis. For zben and exponential equations, the inverse link is applied automatically by default so all demand plots are on the consumption scale.
x_limits, y_limits	Numeric length-2 vectors for axis limits.
x_lab	Character. X-axis label.
y_lab	Character. Y-axis label.
style	Character. Plot style: "modern" or "apa".
observed_point_alpha, observed_point_size	Numeric. Aesthetics for observed data points.
pop_line_alpha, pop_line_size	Numeric. Aesthetics for population curve.
ind_line_alpha, ind_line_size	Numeric. Aesthetics for individual curves.
...	Additional arguments.

Value

A ggplot2 object.

Examples

```
data(apt)
fit <- fit_demand_tmb(apt, equation = "exponential", verbose = 0)

# Population demand curve
plot(fit, type = "demand")

# Individual curves for selected subjects
plot(fit, type = "individual", ids = c("19", "51"))

# Parameter distributions
plot(fit, type = "parameters")
```

Description

Creates a ggplot2 visualization of a fitted linear cross-price demand model (of class `cp_model_lm`). The plot overlays a prediction line over the observed data points. Axis transformations (e.g., "log10") can be applied. If the model includes group effects, separate lines will be drawn for each group.

Usage

```
## S3 method for class 'cp_model_lm'
plot(
  x,
  data = NULL,
  inv_fun = identity,
  n_points = 100,
  title = NULL,
  xlab = "Price",
  ylab = "Consumption",
  x_trans = "identity",
  y_trans = "identity",
  point_size = 3,
  ...
)
```

Arguments

<code>x</code>	A <code>cp_model_lm</code> object (as returned by <code>fit_cp_linear(type = "fixed", ...)</code>).
<code>data</code>	Optional data frame containing columns <code>x</code> and <code>y</code> to plot. If not provided, the function uses <code>object\$data</code> if available.
<code>inv_fun</code>	Optional function to inverse-transform predictions. Default is <code>identity</code> . Not typically used for linear models but included for API consistency.
<code>n_points</code>	Number of points to create in the prediction grid. Default is 100.
<code>title</code>	Optional title for the plot; default is <code>NULL</code> .
<code>xlab</code>	Label for the x-axis. Default is "Price".
<code>ylab</code>	Label for the y-axis. Default is "Consumption".
<code>x_trans</code>	Transformation for the x-axis; one of "identity", "log10", or "pseudo_log". Default is "identity".
<code>y_trans</code>	Transformation for the y-axis; one of "identity", "log10", or "pseudo_log". Default is "identity".
<code>point_size</code>	Size of the data points in the plot. Default is 3.
<code>...</code>	Additional arguments passed to the generic <code>predict</code> method.

Value

A ggplot2 object displaying the fitted model predictions and observed data.

plot.cp_model_lmer	<i>Plot Method for Mixed-Effects Cross-Price Demand Models</i>
--------------------	--

Description

Creates a ggplot2 visualization of a fitted mixed-effects cross-price demand model (of class cp_model_lmer). The pred_type argument selects what is drawn:

Usage

```
## S3 method for class 'cp_model_lmer'
plot(
  x,
  data = NULL,
  inv_fun = identity,
  n_points = 100,
  title = NULL,
  xlab = "Price",
  ylab = "Consumption",
  x_trans = "identity",
  y_trans = "identity",
  point_size = 3,
  pred_type = c("fixed", "random", "all"),
  ...
)
```

Arguments

x	A cp_model_lmer object (as returned by fit_cp_linear(type = "mixed", ...)).
data	Optional data frame containing columns x and y to be plotted. If not provided, object\$data is used.
inv_fun	Optional function to inverse-transform predictions. Default is identity. Not typically used for linear models but included for API consistency.
n_points	Number of points to use in creating the prediction grid. Default is 100.
title	Optional title for the plot; default is NULL.
xlab	Label for the x-axis. Default is "Price".
ylab	Label for the y-axis. Default is "Consumption".
x_trans	Transformation for the x-axis; one of "identity", "log10", or "pseudo_log". Default is "identity".

y_trans	Transformation for the y-axis; one of "identity", "log10", or "pseudo_log". Default is "identity".
point_size	Size of the observed data points. Default is 3.
pred_type	Character string specifying which prediction components to plot: "fixed" Plot only the fixed-effects (population) prediction. "random" Plot only the subject-specific predictions. "all" Plot both the fixed-effects and the subject-specific predictions. The default is "fixed".
...	Additional arguments passed to <code>predict.cp_model_lmer</code> .

Details

"fixed" Only the population-level (fixed-effects) prediction.

"random" Only the subject-specific predictions.

"all" Both: the fixed-effects and the subject-specific predictions.

If the model includes group effects, separate lines will be drawn for each group.

Value

A ggplot2 object displaying the observed data points along with the prediction curves.

plot.cp_model_nls	<i>Plot a Cross-Price Demand Model (Nonlinear)</i>
-------------------	--

Description

Plot a Cross-Price Demand Model (Nonlinear)

Usage

```
## S3 method for class 'cp_model_nls'
plot(
  x,
  data = NULL,
  inv_fun = identity,
  n_points = 100,
  title = NULL,
  xlab = "Price",
  ylab = "Consumption",
  x_trans = "identity",
  y_trans = "identity",
  point_size = 3,
  inverse_fun = deprecated(),
  ...
)
```

Arguments

<code>x</code>	A cross-price model object from <code>fit_cp_nls</code> with <code>return_all=TRUE</code> .
<code>data</code>	Optional data frame with <code>x</code> and <code>y</code> ; if <code>NULL</code> , uses <code>object\$data</code> .
<code>inv_fun</code>	Optional function to inverse-transform predictions. Default is <code>identity</code> .
<code>n_points</code>	Number of points used for prediction curve.
<code>title</code>	Optional plot title.
<code>xlab</code>	X-axis label.
<code>ylab</code>	Y-axis label.
<code>x_trans</code>	Transformation for x-axis: "identity", "log10", or "pseudo_log".
<code>y_trans</code>	Transformation for y-axis: "identity", "log10", or "pseudo_log".
<code>point_size</code>	Size of data points.
<code>inverse_fun</code>	[Deprecated] Use <code>inv_fun</code> instead.
<code>...</code>	Additional arguments (passed to <code>predict</code>).

Value

A `ggplot2` object.

`plot_alpha_distribution`

Plot Distribution of Subject-Level Alpha Estimates

Description

Visualizes the distribution of individual-level alpha (elasticity) estimates from a fitted hurdle model, with the population MLE as a reference.

Usage

```
plot_alpha_distribution(object, ...)

## S3 method for class 'beezdemand_hurdle'
plot_alpha_distribution(
  object,
  type = c("density", "histogram"),
  log_scale = TRUE,
  bins = 30,
  style = c("modern", "apa"),
  ...
)

## S3 method for class 'beezdemand_tmb'
plot_alpha_distribution(
```

```

    object,
    type = c("density", "histogram"),
    log_scale = TRUE,
    bins = 30,
    style = c("modern", "apa"),
    ...
  )

```

Arguments

object	A fitted model object with subject-level parameters.
...	Additional arguments passed to methods.
type	Character; "density" (default) or "histogram".
log_scale	Logical; plot on log10 scale (default TRUE).
bins	Integer; histogram bins (default 30).
style	Character; "modern" or "apa".

Value

A ggplot2 object.

Examples

```

data(apt)
fit <- fit_demand_hurdle(apt, y_var = "y", x_var = "x", id_var = "id",
  random_effects = c("zeros", "q0", "alpha"))
plot_alpha_distribution(fit)

```

plot_demand_overlay *Overlay Demand Curves from Multiple Models*

Description

Plots population-level demand curves from multiple fitted models on the same axes for visual comparison.

Usage

```

plot_demand_overlay(
  ...,
  model_list = NULL,
  labels = NULL,
  prices = NULL,
  n_points = 200,
  x_trans = c("log10", "log", "linear", "pseudo_log"),

```

```

    free_trans = 0.01,
    y_min = 0.001,
    inv_fun = identity,
    x_lab = "Price",
    y_lab = "Consumption",
    style = c("modern", "apa")
  )

```

Arguments

<code>...</code>	Fitted model objects (named or unnamed).
<code>model_list</code>	Optional named list of models (combined with <code>...</code>).
<code>labels</code>	Character vector of model labels for the legend.
<code>prices</code>	Numeric vector of prices. Default uses the union of observed prices across all models.
<code>n_points</code>	Integer; number of points for smooth curves (default 200).
<code>x_trans</code>	Character; x-axis transformation (default "log10").
<code>free_trans</code>	Numeric; replacement for price = 0 on log scales.
<code>y_min</code>	Numeric; minimum consumption value to display. Values below this floor are dropped to prevent extreme predictions (e.g., 1e-16 from hurdle models) from compressing the y-axis. Set to NULL to disable. Default is 0.001.
<code>inv_fun</code>	Function to back-transform consumption values (e.g., ll4_inv for LL4-transformed models). Default is identity .
<code>x_lab, y_lab</code>	Axis labels.
<code>style</code>	Character; "modern" or "apa".

Value

A ggplot2 object.

Examples

```

data(apt)
fit1 <- fit_demand_hurdle(apt, y_var = "y", x_var = "x", id_var = "id")
# Compare with a 3-RE model:
# fit2 <- fit_demand_hurdle(apt, y_var = "y", x_var = "x", id_var = "id",
#   random_effects = c("zeros", "q0", "alpha"))
# plot_demand_overlay(fit1, fit2, labels = c("2-RE", "3-RE"))
plot_demand_overlay(fit1, labels = c("Hurdle"))

```

plot_elasticity	<i>Plot Own-Price Elasticity Curve</i>
-----------------	--

Description

Computes and plots the own-price point elasticity of demand across prices. Elasticity is computed numerically via central differences on the unconditional demand curve.

Usage

```
plot_elasticity(object, ...)

## S3 method for class 'beezdemand_hurdle'
plot_elasticity(
  object,
  prices = NULL,
  n_points = 200,
  show_unit = TRUE,
  x_trans = c("log10", "log", "linear", "pseudo_log"),
  free_trans = 0.01,
  x_lab = "Price",
  y_lab = "Elasticity",
  style = c("modern", "apa"),
  ...
)

## S3 method for class 'beezdemand_tmb'
plot_elasticity(
  object,
  prices = NULL,
  n_points = 200,
  show_unit = TRUE,
  x_trans = c("log10", "log", "linear", "pseudo_log"),
  free_trans = 0.01,
  x_lab = "Price",
  y_lab = "Elasticity",
  style = c("modern", "apa"),
  ...
)
```

Arguments

object	A fitted model object.
...	Additional arguments passed to methods.
prices	Numeric vector of prices. If NULL, uses a smooth grid spanning the observed price range.

n_points	Integer; number of grid points (default 200).
show_unit	Logical; show unit elasticity reference line at -1 (default TRUE).
x_trans	Character; x-axis transformation (default "log10").
free_trans	Numeric; replacement for price = 0 on log scales.
x_lab, y_lab	Axis labels.
style	Character; "modern" or "apa".

Details

Point elasticity is computed as:

$$\eta(P) = \frac{dQ}{dP} \cdot \frac{P}{Q(P)}$$

This uses the unconditional demand Q(P) for hurdle models (which includes the probability of zero consumption), so the result is the economically relevant total elasticity.

Value

A ggplot2 object.

Examples

```
data(apt)
fit <- fit_demand_hurdle(apt, y_var = "y", x_var = "x", id_var = "id")
plot_elasticity(fit)
```

plot_expenditure	<i>Plot Expenditure Curves</i>
------------------	--------------------------------

Description

Plots the expenditure curve (Price x Consumption) derived from the fitted demand model. Optionally overlays Pmax and Omax reference lines.

Usage

```
plot_expenditure(object, ...)

## S3 method for class 'beezdemand_hurdle'
plot_expenditure(
  object,
  prices = NULL,
  n_points = 200,
  show_pmax = TRUE,
  show_omax = TRUE,
```

```

    demand_type = c("unconditional", "conditional"),
    x_trans = c("log10", "log", "linear", "pseudo_log"),
    free_trans = 0.01,
    x_lab = "Price",
    y_lab = "Expenditure (P × Q)",
    style = c("modern", "apa"),
    ...
)

## S3 method for class 'beezdemand_tmb'
plot_expenditure(
  object,
  prices = NULL,
  n_points = 200,
  show_pmax = TRUE,
  show_omax = TRUE,
  x_trans = c("log10", "log", "linear", "pseudo_log"),
  free_trans = 0.01,
  x_lab = "Price",
  y_lab = "Expenditure (P × Q)",
  style = c("modern", "apa"),
  ...
)

```

Arguments

object	A fitted model object.
...	Additional arguments passed to methods.
prices	Numeric vector of prices. If NULL, uses a smooth grid spanning the observed price range.
n_points	Integer; number of grid points (default 200).
show_pmax	Logical; show Pmax vertical line (default TRUE).
show_omax	Logical; show Omax horizontal line (default TRUE).
demand_type	Character; for beezdemand_hurdle only. "unconditional" (default) plots $p * (1 - P_0(p)) * Q(p)$ and overlays the unconditional Pmax/Omax reference lines. "conditional" plots the Part-II demand $p * Q(p)$ (consumption given positive purchase) and overlays the Part-II-only Pmax/Omax reference lines. Either way the displayed curve and the Pmax/Omax lines are guaranteed to come from the same metric set, so they always align.
x_trans	Character; x-axis transformation (default "log10").
free_trans	Numeric; replacement for price = 0 on log scales.
x_lab, y_lab	Axis labels.
style	Character; "modern" or "apa".

Value

A ggplot2 object.

Examples

```
data(apt)
fit <- fit_demand_hurdle(apt, y_var = "y", x_var = "x", id_var = "id")
plot_expenditure(fit)
```

plot_loss_profile	<i>Plot Loss Profile for a Single Parameter</i>
-------------------	---

Description

Plots 1D slices of the SSR surface, fixing one parameter at the MLE and varying the other.

Usage

```
plot_loss_profile(object, ...)

## S3 method for class 'beezdemand_hurdle'
plot_loss_profile(
  object,
  parameter = c("both", "q0", "alpha"),
  resolution = 200,
  range = c(-3, 3),
  style = c("modern", "apa"),
  ...
)

## S3 method for class 'beezdemand_tmb'
plot_loss_profile(
  object,
  parameter = c("both", "q0", "alpha"),
  resolution = 200,
  range = c(-3, 3),
  style = c("modern", "apa"),
  ...
)

## S3 method for class 'beezdemand_nlme'
plot_loss_profile(
  object,
  parameter = c("both", "q0", "alpha"),
  resolution = 200,
  range = c(-3, 3),
  style = c("modern", "apa"),
  type = c("ssr", "marginal"),
  ...
)
```

Arguments

object	A fitted model object.
...	Additional arguments passed to methods.
parameter	Character; which parameter to profile: "q0", "alpha", or "both" (default).
resolution	Integer; number of grid points (default 200).
range	Numeric vector of length 2; range in log10 units relative to MLE (default c(-3, 3)).
style	Character; plot style, "modern" or "apa".
type	Character; loss profile to plot for NLME models. "ssr" (default) profiles sum-of-squared-residuals on price-aggregated means; "marginal" profiles a linearized marginal negative log-likelihood.

Value

A ggplot2 object. If parameter = "both" and patchwork is available, returns a combined patchwork object.

Examples

```
data(apt)
fit <- fit_demand_hurdle(apt, y_var = "y", x_var = "x", id_var = "id")
plot_loss_profile(fit, parameter = "q0")
plot_loss_profile(fit, parameter = "both")
```

plot_loss_surface	<i>Plot Loss Surface for Demand Model Parameters</i>
-------------------	--

Description

Visualizes the sum-of-squared-residuals (SSR) surface over a grid of Q0 and alpha values, holding other parameters (k, variance components) fixed at their MLE. The SSR is computed on aggregated mean log-consumption by price for the Part II (continuous) component of hurdle models.

Usage

```
plot_loss_surface(object, ...)

## S3 method for class 'beezdemand_hurdle'
plot_loss_surface(
  object,
  resolution = 80,
  q0_range = NULL,
  alpha_range = NULL,
```

```

    fill_palette = "D",
    show_mle = TRUE,
    show_contours = FALSE,
    style = c("modern", "apa"),
    ...
)

## S3 method for class 'beezdemand_tmb'
plot_loss_surface(
  object,
  resolution = 80,
  q0_range = NULL,
  alpha_range = NULL,
  fill_palette = "D",
  show_mle = TRUE,
  show_contours = FALSE,
  style = c("modern", "apa"),
  ...
)

## S3 method for class 'beezdemand_nlme'
plot_loss_surface(
  object,
  resolution = 80,
  q0_range = NULL,
  alpha_range = NULL,
  fill_palette = "D",
  show_mle = TRUE,
  show_contours = FALSE,
  style = c("modern", "apa"),
  type = c("ssr", "marginal"),
  ...
)

```

Arguments

<code>object</code>	A fitted model object.
<code>...</code>	Additional arguments passed to methods.
<code>resolution</code>	Integer; grid resolution per axis (default 80).
<code>q0_range</code>	Numeric vector of length 2; Q0 range (natural scale). Default: MLE +/- 3 orders of magnitude.
<code>alpha_range</code>	Numeric vector of length 2; alpha range (natural scale). Default: MLE +/- 3 orders of magnitude.
<code>fill_palette</code>	Character; viridis palette option (default "D").
<code>show_mle</code>	Logical; overlay MLE point (default TRUE).
<code>show_contours</code>	Logical; add contour lines (default FALSE).

style	Character; plot style, "modern" or "apa".
type	Character; loss surface to plot for NLME models. "ssr" (default) uses sum-of-squared-residuals on price-aggregated means; "marginal" uses a linearized marginal negative log-likelihood.

Details

Note that this function computes SSR on price-aggregated means rather than a true profile likelihood. The resulting surface shows how well different (Q0, alpha) pairs explain the average demand pattern, but does not account for individual variation. For models with large random effects, the surface may appear sharper than the full-data objective.

Supported model classes: beezdemand_hurdle, beezdemand_tmb, and beezdemand_nlme. Models with factor covariates on Q0 or alpha are not supported; use [get_demand_param_emms\(\)](#) instead.

Value

A ggplot2 object.

See Also

[plot_loss_profile\(\)](#) for 1D profile slices

Examples

```
data(apt)
fit <- fit_demand_hurdle(apt, y_var = "y", x_var = "x", id_var = "id")
plot_loss_surface(fit)
plot_loss_surface(fit, resolution = 50, show_contours = TRUE)
```

plot_model_comparison *Compare Parameter Estimates Across Models*

Description

Creates a forest plot (coefficient plot) comparing parameter estimates and confidence intervals across multiple fitted demand models.

Usage

```
plot_model_comparison(
  ...,
  model_list = NULL,
  labels = NULL,
  parameters = c("Q0", "alpha"),
  conf_level = 0.95,
  style = c("modern", "apa")
)
```

Arguments

<code>...</code>	Fitted model objects.
<code>model_list</code>	Optional named list of models.
<code>labels</code>	Character vector of model labels.
<code>parameters</code>	Character vector of parameter names to compare (default <code>c("Q0", "alpha")</code>).
<code>conf_level</code>	Numeric; confidence level for intervals (default 0.95).
<code>style</code>	Character; "modern" or "apa".

Details

Uses `tidy()` methods to extract parameter estimates and standard errors. Confidence intervals are computed as $\text{estimate} \pm z * \text{SE}$. Parameters are matched by the `term` column from `tidy()` output.

Value

A `ggplot2` object.

Examples

```
data(apt)
fit <- fit_demand_hurdle(apt, y_var = "y", x_var = "x", id_var = "id")
plot_model_comparison(fit, labels = "Hurdle")
```

plot_qq

Plot Random Effects Q-Q

Description

Creates Q-Q plots for random effects to assess normality assumptions.

Usage

```
plot_qq(object, which = NULL, ...)

## S3 method for class 'beezdemand_hurdle'
plot_qq(object, which = NULL, ...)

## S3 method for class 'beezdemand_nlme'
plot_qq(object, which = NULL, ...)

## S3 method for class 'beezdemand_tmb'
plot_qq(object, which = NULL, ...)
```

Arguments

<code>object</code>	A fitted model object with random effects (<code>beezdemand_hurdle</code> , <code>beezdemand_nlme</code> , or <code>beezdemand_tmb</code>).
<code>which</code>	Character vector; which random effects to plot. Default is all.
<code>...</code>	Additional arguments (ignored).

Details

For `beezdemand_tmb` models, Q-Q plots display empirical Bayes predictions (BLUPs) of the random effects (`b_i` for Q_0 , `c_i` for α). These exhibit shrinkage toward the population mean: subjects with fewer observations or higher variability are pulled toward zero. Consequently, Q-Q plots may appear more normal than the true random effect distribution (Verbeke & Molenberghs, 2000). Useful for detecting gross departures (bimodality, heavy tails) but not a definitive normality test.

Value

A `ggplot2` object.

Examples

```
data(apt)
fit <- fit_demand_hurdle(apt, y_var = "y", x_var = "x", id_var = "id")
plot_qq(fit)
```

`plot_re_diagnostics` *Diagnostic Plots for Random Effects*

Description

Creates diagnostic panels for random effects: histogram and Q-Q plot for each selected random effect, plus (for the zeros RE) a comparison of observed vs predicted proportion of zeros across prices.

Usage

```
plot_re_diagnostics(object, ...)

## S3 method for class 'beezdemand_hurdle'
plot_re_diagnostics(
  object,
  which = c("all", "zeros", "q0", "alpha"),
  style = c("modern", "apa"),
  ...
)
```

```
## S3 method for class 'beezdemand_tmb'
plot_re_diagnostics(
  object,
  which = c("all", "q0", "alpha"),
  style = c("modern", "apa"),
  ...
)
```

Arguments

- object A fitted model object with random effects.
- ... Additional arguments (ignored).
- which Character; which random effects to diagnose. One of "all" (default), "zeros", "q0", or "alpha".
- style Character; "modern" or "apa".

Value

A ggplot2/patchwork object, or a list of ggplot2 objects if patchwork is not installed.

Examples

```
data(apt)
fit <- fit_demand_hurdle(apt, y_var = "y", x_var = "x", id_var = "id")
plot_re_diagnostics(fit)
plot_re_diagnostics(fit, which = "zeros")
```

plot_residuals	<i>Plot Residual Diagnostics</i>
----------------	----------------------------------

Description

Creates diagnostic plots for model residuals including residuals vs fitted, scale-location, and histogram of residuals.

Usage

```
plot_residuals(
  object,
  type = c("all", "fitted", "histogram", "qq"),
  component = c("combined", "continuous"),
  ...
)
```

Arguments

object	A fitted model object.
type	Character; type of residual plot. One of: • "fitted": Residuals vs fitted values • "histogram": Histogram of residuals • "qq": Q-Q plot of residuals • "all": All plots combined (default)
component	Character; for hurdle models, which residuals to plot: "combined" (default) uses randomized quantile residuals that assess both binary and continuous components simultaneously; "continuous" uses log-scale Part II residuals only (zeros excluded). Ignored for non-hurdle models.
...	Additional arguments passed to plotting functions.

Details

For hurdle models, diagnostic plots default to randomized quantile residuals that assess both the binary and continuous components simultaneously. Set component = "continuous" to see only Part II (log-scale) residuals.

Value

A ggplot2 object or list of ggplot2 objects.

Examples

```
data(apt)
fit <- fit_demand_hurdle(apt, y_var = "y", x_var = "x", id_var = "id")
plot_residuals(fit)
```

plot_subject

Plot Demand Curve for a Single Subject

Description

Creates a demand curve plot for a single subject with optional observed data and population reference curve.

Usage

```
plot_subject(
  object,
  subject_id,
  prices = NULL,
  show_data = TRUE,
```

```
  show_population = TRUE,  
  style = c("modern", "apa")  
)
```

Arguments

object	An object of class beezdemand_hurdle.
subject_id	The ID of the subject to plot.
prices	Numeric vector of prices for plotting. If NULL, uses a sequence from 0 to max observed price.
show_data	Logical; if TRUE, overlay observed data points. Default is TRUE.
show_population	Logical; if TRUE, show population curve. Default is TRUE.
style	Plot styling, passed to theme_beezdemand().

Value

A ggplot2 object.

Examples

```
data(apt)  
fit <- fit_demand_hurdle(apt, y_var = "y", x_var = "x", id_var = "id")  
plot_subject(fit, subject_id = "19")
```

PlotCurve	<i>Plot Curve</i>
-----------	-------------------

Description

Creates a single plot object

Usage

```
PlotCurve(adf, dfrow, newdat, yscale = "log", style = c("modern", "apa"))
```

Arguments

adf	Data frame (long form) of purchase task data.
dfrow	A row of results from FitCurves
newdat	A newdat dataframe from FitCurves
yscale	Scaling of y axis. Default is "log". Can also take "linear"
style	Plot styling, passed to theme_beezdemand().

Details

Creates individual demand curves

Value

ggplot2 graphical object

Author(s)

Shawn Gilroy shawn.gilroy@temple.edu

Examples

```
## Creates a single plot from elements of an object created by FitCurves
if (interactive()) {
  fc <- FitCurves(apt, "hs", k = 2, detailed = TRUE)
  PlotCurve(fc$adfs[[1]], fc$dfres[1, ], fc$newdates[[1]])
}
```

PlotCurves

Plot Curves

Description

Creates plots

Usage

```
PlotCurves(dat, outdir = NULL, device = "png", ending = NULL, ask = TRUE, ...)
```

Arguments

dat	FitCurves object with 4 elements (dfres, newdates, adfs, fits)
outdir	Directory where plots are saved
device	Type of file. Default is "png". Can be "pdf"
ending	Optional. Can specify to only plot through a certain number of datasets
ask	Can view plots one by one. If TRUE, plots will not save
...	Pass arguments to PlotCurve (for example yscale = c("log", "linear"))

Details

Creates and saves plots of individual demand curves

Value

Nothing

Author(s)

Brent Kaplan bkaplan.ku@gmail.com, Shawn Gilroy shawn.gilroy@temple.edu

Examples

```
## Interactively view plots from output from FitCurves
if (interactive()) {
  fc <- FitCurves(apt, "hs", k = 2, detailed = TRUE)
  PlotCurves(fc, ask = TRUE)
}
```

power_demand

Monte Carlo power analysis for two-condition demand designs

Description

Estimates statistical power to detect a single fixed-effect difference in a demand parameter (Q_0 or α) between two conditions, by simulation: each replicate (1) simulates a two-condition dataset from the mixed-effects demand model in `.simulate_within_subject_demand()` under assumed population parameters plus the effect δ , under either a within-subject design (every subject observed in both conditions) or, with `design_type = "between"`, a two-arm between-subject design (each subject in one condition); (2) refits it with `fit_demand_tmb()`; and (3) tests the condition contrast on the estimation (natural log) scale with a Wald test at level α , referred to a t distribution with df degrees of freedom (see the `df` argument). Power is the proportion of *usable* fits (converged, positive-definite Hessian, finite standard error) that reject.

Because the power estimate is a proportion from finitely many replicates, it is reported with a Wilson score confidence interval (`power_mc_ci`). Both a p-value verdict ($p < \alpha$) and a confidence-interval verdict (Wald CI excludes 0) are recorded per replicate; they use the same standard error and reference distribution, so they coincide by construction, and both rates are returned.

Usage

```
power_demand(
  n_subjects,
  effect = list(delta_q0 = NULL, delta_alpha = NULL),
  design = list(),
  n_sim = 500,
  alpha = 0.05,
  df = NULL,
  seed = NULL,
  equation = "simplified",
  random_effects = NULL,
  multi_start = FALSE,
  verbose = TRUE,
  design_type = c("within", "between"),
  ...
)
```

Arguments

n_subjects	Number of simulated subjects per replicate. For design_type = "within" each subject is observed at every price in both conditions; for "between" this is the <i>total</i> sample, split ceiling(n_subjects / 2) to condition 1 and the rest to condition 2 (an odd total therefore gives arms differing by one subject).
effect	Named list supplying exactly one of delta_q0 or delta_alpha: the true condition shift on natural-log Q0 (or natural-log alpha) for condition 2 relative to condition 1. 0 is allowed (useful for Type I error checks). E.g. delta_q0 = log(1.5) means condition 2's Q0 is 1.5 times condition 1's.
design	Named list of data-generating settings, merged over the simulator defaults: prices (vector), log_q0_pop, log_alpha_pop, sigma_b (per-condition subject SD on log Q0), sigma_d (same for log alpha), rho_bd (must be 0 in this version), and sigma_e (residual SD on log consumption). All SDs must be strictly positive.
n_sim	Number of Monte Carlo replicates. 500 (default) is suitable for interactive exploration; use 2000+ for grant-quality precision (see vignette("power-analysis")).
alpha	Nominal two-sided test level.
df	Degrees of freedom for the Wald test's t reference distribution. NULL (default) uses n_subjects - 1 for design_type = "within" and n_subjects - 2 for design_type = "between" (the two-sample df). This is an <i>empirically calibrated</i> small-sample correction rather than a model-derived df (the TMB fit has no exact t sampling theory): the asymptotic z-test was measurably anticonservative in the package's Type I calibration battery (empirical rate 0.089 at nominal .05 with 15 subjects), while the t reference passes the battery's null checks across the tested sample sizes, target parameters, residual-SD settings, and both designs. Inf gives the asymptotic z-test.
seed	Optional integer seed; identical seeds give identical results. The caller's RNG state is restored on exit.
equation	Demand equation passed to <code>fit_demand_tmb()</code> . The default "simplified" shares the simulator's mean function, so the simulated delta and the fitted contrast share the same scale. Note the error model is a <i>working model</i> : the simulator draws multiplicative lognormal errors while "simplified" fits additive Gaussian errors on raw consumption; the approximation is closest at small sigma_e, and Type I calibration is verified by the test suite at the default and a 3x-larger sigma_e. Other equations are sensitivity analyses with a different estimand, so their contrasts are not on the scale of the simulated delta.
random_effects	Random-effects specification passed to <code>fit_demand_tmb()</code> . NULL (default) resolves to a specification matching design_type: nlme::pdDiag(Q0 + alpha ~ condition - 1) for "within" (independent per-condition subject effects on both parameters, matching the simulator's data-generating process when rho_bd = 0), and nlme::pdDiag(Q0 + alpha ~ 1) for "between" (per-subject intercepts, which are correctly specified because each subject appears in only one condition). Supply a specification to override.
multi_start	Passed to <code>fit_demand_tmb()</code> . Defaults to FALSE for speed (roughly 3x fewer optimizations); non-convergent replicates are excluded and surfaced rather than biasing the estimate.

verbose	Logical; show a progress bar.
design_type	Either "within" (default) or "between". "within" simulates the two-condition within-subject design (every subject observed at every price in both conditions) and tests the within-subject condition contrast. "between" assigns each subject to exactly one of two arms (ceiling($n/2$) to condition 1, the rest to condition 2) and tests the group difference; the two-arm dataset is composed from the same simulator run once per arm with a single condition, so no new data-generating process is introduced (see Details). The df and random_effects defaults track design_type.
...	Additional arguments passed to <code>fit_demand_tmb()</code> (e.g. <code>tmb_control</code>).

Details

A replicate whose fit fails (non-convergence, non-positive-definite Hessian, unusable standard error, or an error) is excluded from the power denominator and reported through the `n_*` counts and `$replicates$status`. It is never counted as "no effect detected", which would bias power in an unpredictable direction. A warning is issued when fewer than 95% of replicates are usable, since power conditional on convergence can be selected when convergence depends on the realized data.

The `v1` scope is a single fixed-effect delta. Joint $Q_0 + \alpha$ effects, power for derived measures ($P_{\max}$, $O_{\max}$), and arbitrary designs are out of scope; see `vignette("power-analysis")`.

The within-subject simulator draws a fresh pair of subject deviations ($\log Q_0$, $\log \alpha$) for every subject in *every condition*. A subject's deviation in condition 1 is independent of their deviation in condition 2, and ρ_{bd} (the within-condition Q_0/α correlation) is pinned at 0. No subject-level effect is shared across conditions, so the within-subject power estimate describes two conditions with independent subject effects. In the usual repeated-measures design each subject's deviation carries over and the condition contrast is paired; that design is not what is simulated here. When the carried-over deviations are positively correlated across conditions (the common case) pairing reduces the contrast variance, so the power reported here is a conservative benchmark for such a design; a negative cross-condition correlation would make it optimistic instead. The default `random_effects` (per-condition effects) matches the simulated data-generating process exactly.

For `design_type = "between"`, the two arms are composed by running the within-subject simulator once per arm with a single condition, then binding the arms and refitting with per-subject intercept random effects. Because each subject appears in only one condition, that random-effects structure matches the composed data-generating process exactly, as the within-subject default matches its own. In both designs the additive-Gaussian residual likelihood is a *working model* for the simulator's multiplicative-lognormal errors (closest at small σ_e). Type I error is calibrated by the test suite for both designs.

Value

An object of class `beezdemand_power`: a list with

power Estimated power: proportion of usable replicates whose Wald CI excludes 0 (equal to `hit_rate_ci`). NA if no replicate was usable.

power_mc_ci Wilson 95% confidence interval on power, reflecting Monte Carlo uncertainty from `n_used` replicates.

hit_rate_p Proportion of usable replicates with $p < \alpha$.

- hit_rate_ci** Proportion of usable replicates whose Wald CI excludes 0 (the same decision rule as `hit_rate_p`, since both use the same SE and t reference; both reported).
- n_sim** Total replicates attempted.
- n_converged** Replicates whose fit converged.
- n_hessian_pd** Replicates with a positive-definite Hessian.
- n_used** Replicates entering the power denominator (converged, positive-definite Hessian, finite SE).
- alpha** Nominal test level.
- df** Degrees of freedom of the t reference distribution actually used (`n_subjects - 1` for "within", `n_subjects - 2` for "between", unless overridden).
- effect** The validated effect specification (name and delta).
- target_term** The tested coefficient (e.g. "Q0:conditionC2").
- design** The merged design list actually used.
- n_subjects** As supplied.
- replicates** Tibble with one row per replicate: `sim`, `status` ("ok", "nonconverged", "hessian_not_pd", "se_unusable", "error"), `converged`, `hessian_pd`, `estimate`, `se`, `statistic`, `p_value`, `ci_lower`, `ci_upper`, `hit_p`, `hit_ci`, and `message` (error text, if any). Estimates are on the natural-log scale of the simulated delta.
- seed** As supplied.
- settings** List of equation, `design_type`, `multi_start`, and the deparsed random-effects specification.
- call** The matched call.

See Also

[find_n_demand\(\)](#) to search for the smallest adequate sample size; [fit_demand_tmb\(\)](#) for the model being refit.

Other power-analysis: [find_n_demand\(\)](#)

Examples

```
# Tiny illustrative run (use n_sim >= 500 for real planning; the
# Monte Carlo interval at n_sim = 5 is uninformative by design)
res <- power_demand(
  n_subjects = 12,
  effect = list(delta_q0 = log(1.5)),
  n_sim = 5, seed = 1, verbose = FALSE
)
print(res)

# Between-subject design: group difference in Q0 across two arms
res_b <- power_demand(
  n_subjects = 16,
  effect = list(delta_q0 = log(1.5)),
  design_type = "between",
```

```

  n_sim = 5, seed = 1, verbose = FALSE
)
print(res_b)

```

predict.beezdemand_fixed

Predict Method for beezdemand_fixed

Description

Predict Method for beezdemand_fixed

Usage

```

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

```

Arguments

object	A beezdemand_fixed object.
newdata	A data frame containing a price column matching the fitted object's x_var. If NULL, uses the unique observed prices when available.
type	One of "response" (default) or "link".
se.fit	Logical; if TRUE, includes a .se.fit column (currently NA because vcov is not available from legacy fixed fits).
interval	One of "none" (default) or "confidence". When requested, .lower/.upper are returned as NA because vcov is unavailable.
level	Confidence level when interval = "confidence". Currently used only for validation.
...	Unused.

Value

A tibble containing the original newdata columns, plus .fitted and, when requested, .se.fit and .lower/.upper. If newdata does not include an id column, predictions are returned for all subjects (cross product of newdata × subjects) unless k is subject-specific (k = "ind").

predict.beezdemand_hurdle

Predict Method for Hurdle Demand Models

Description

Returns predictions from a fitted hurdle demand model.

Usage

```
## S3 method for class 'beezdemand_hurdle'
predict(
  object,
  newdata = NULL,
  type = c("demand", "response", "link", "parameters", "probability"),
  prices = NULL,
  marginal = FALSE,
  marginal_method = c("normal", "kde", "empirical"),
  correction = TRUE,
  seed = 42L,
  se.fit = FALSE,
  interval = c("none", "confidence"),
  level = 0.95,
  ...
)
```

Arguments

object	An object of class beezdemand_hurdle.
newdata	Optional data frame containing a price column matching the fitted object's x_var. If newdata includes the id column, subject-specific predictions are returned; otherwise population predictions are returned. If newdata is NULL, returns predictions for all subjects across a price grid.
type	One of: "demand" (default) Predicted expected consumption = $(1 - P_0) * \text{response}$ (the marginal expectation; see <i>Scoring predictions</i>) "response" Predicted consumption conditional on consuming (part II), $E[Y \mid Y > 0]$ "link" Predicted log-consumption (linear predictor of part II) "probability" Predicted probability of zero consumption (part I) "parameters" Subject-specific parameters (no .fitted column) The default changed from "response" to "demand" in beezdemand 0.3.0; omitting type emits a one-time-per-session message naming the change.
prices	Optional numeric vector of prices used only when newdata = NULL.

marginal	Logical; if TRUE, computes population-averaged (marginal) predictions by integrating over the random effects distribution. For type = "probability", uses KDE/Normal/Empirical integration of the binary component. For type = "response" and type = "demand", uses Monte Carlo integration over all random effects, producing the population-average demand curve (accounting for Jensen's inequality). Default is FALSE, which gives conditional (RE = 0) predictions representing a "typical" subject at the center of the RE distribution.
marginal_method	Character. Method for marginal integration; one of "normal" (default; integrate over the model-assumed $N(0, \sigma_a)$ distribution of the zero-component intercept, over the whole real line), "kde" (kernel density estimate of the shrunk BLUPs), or "empirical" (simple average over the BLUPs). "normal" is the model-consistent choice: it integrates over the same distribution the fitted likelihood integrates over. "kde" and "empirical" are descriptive summaries of the shrunk BLUPs, which understate the random-effect spread (see Details). The default was "kde" in the development versions before 0.3.0. Ignored when marginal = FALSE.
correction	Logical; if TRUE (default), applies the lognormal retransformation correction $\exp(\sigma_e^2 / 2)$ when back-transforming from the log scale to the natural consumption scale. This produces the arithmetic mean (conditional on $Q > 0$). Set to FALSE to obtain the median (geometric mean), which is useful for individual-level "most likely" predictions. Only applies to type = "response" and type = "demand".
seed	Integer or NULL. Random seed for Monte Carlo marginal predictions (default 42L). Set to NULL to use current RNG state. The global RNG state is preserved and restored after the call.
se.fit	Logical; if TRUE, includes a .se.fit column (delta-method via sdreport when available).
interval	One of "none" (default) or "confidence".
level	Confidence level when interval = "confidence".
...	Unused.

Details

Retransformation correction:

The hurdle model specifies Gaussian errors on log-consumption (Part II): $\log(Q) \sim N(\mu, \sigma_e^2)$. The conditional distribution of Q given $Q > 0$ is therefore lognormal. The arithmetic mean of a lognormal is $\exp(\mu + \sigma_e^2/2)$, not $\exp(\mu)$. Using $\exp(\mu)$ returns the median (geometric mean), which systematically underestimates the arithmetic mean by a factor of $\exp(\sigma_e^2/2)$. This correction is applied by default when type = "response" or type = "demand". Set correction = FALSE to obtain the median instead.

This is a parametric correction assuming normality of log-scale residuals (Duan, 1983). Under the model's normality assumption, this is equivalent to Duan's nonparametric smearing estimator.

Marginal P(zero):

The conditional P(zero) curve (when marginal = FALSE) sets the random intercept to zero, which produces a near step-function that misrepresents the observed fraction of zero responses. The

marginal curve integrates over the random effect distribution, answering "what fraction of the population has stopped buying at this price?"

The "kde" and "empirical" methods integrate over empirical Bayes estimates (BLUPs) of the random intercepts. They are descriptive rather than model-consistent: BLUPs are shrunk toward zero compared to the true random effects (more so for subjects with few observations), so these methods understate the RE spread, and the marginal curve they produce summarises the fitted subjects rather than the population-level quantity the model defines. The "normal" method integrates over the model-assumed $N(0, \sigma_a)$ distribution, which is the model-consistent marginal (the same one the fitted likelihood integrates over) and is the choice to use when the marginal curve is reported as an estimate. It can be wrong only in the way the model itself is wrong, that is, if the normality assumption fails. The default remains "kde" for continuity with earlier versions. Use `plot_qq()` to assess RE normality.

Conditional vs. marginal demand predictions:

Population-level demand predictions (when no subject ID is provided) can be computed in two ways:

- Conditional (default, `marginal = FALSE`): Sets all random effects to zero and evaluates the demand function at the fixed-effect (population) parameters. For nonlinear models, this corresponds to the conditional mode rather than the population-average mean.
- Marginal (`marginal = TRUE`): Integrates the prediction over the estimated random-effects distribution via Monte Carlo sampling. This gives the population-average demand curve. Due to Jensen's inequality, this curve lies above the conditional curve when the demand function is convex in the random effects (which it is for exponential demand with log-normal Q_0 and α).

The conditional prediction is appropriate for characterizing a "typical" subject. The marginal prediction is appropriate for predicting aggregate consumption in a population.

Value

For `type = "parameters"`, a tibble of subject-level parameters. For `type = "probability"` with `marginal = TRUE`, a tibble with columns for price, `prob_zero`, and `.fitted` (no subject column). Otherwise, a tibble containing the newdata columns plus `.fitted` and helper columns `predicted_log_consumption`, `predicted_consumption`, `prob_zero`, and `expected_consumption`. When requested, `.se.fit` and `.lower/.upper` are included. Marginal results carry a `marginal_method` attribute; Monte Carlo marginal results (`type = "response" / "demand"`) also carry a logical `re_cov_fallback` attribute that is TRUE when the random-effect draws had to use an uncorrelated (diagonal) covariance because the estimated covariance was not positive definite.

Scoring predictions

A hurdle model has two natural "predicted consumption" quantities and they answer different questions. `type = "response"` is the conditional positive mean $E[Y | Y > 0]$: consumption *given* that any is purchased. `type = "demand"` is the marginal expectation $(1 - p_0)E[Y | Y > 0]$, which weights in the probability of buying nothing. Observed consumption includes zeros, so predictions scored against raw data (cross-validation error, calibration, model comparison on predictions) must use `type = "demand"`; scoring with "response" systematically overstates error wherever p_0 is large (typically at high prices). This is why "demand" is the default as of beezdemand 0.3.0.

Examples

```
data(apt)
fit <- fit_demand_hurdle(apt, y_var = "y", x_var = "x", id_var = "id")

# Get subject-specific parameters
pars <- predict(fit, type = "parameters")

# Predict demand at specific prices
demand <- predict(fit, type = "demand", prices = c(0, 0.5, 1, 2, 5))
```

predict.beezdemand_nlme

Predict Method for beezdemand_nlme Objects

Description

Generates point predictions from a fitted beezdemand_nlme model. Predictions can be made at the population level (fixed effects only) or group/subject level (fixed + random effects). The output scale depends on the equation_form used during model fitting and whether inv_fun is applied.

Usage

```
## S3 method for class 'beezdemand_nlme'
predict(
  object,
  newdata = NULL,
  type = c("response", "link", "population", "individual"),
  level = 0,
  inv_fun = identity,
  se.fit = FALSE,
  interval = c("none", "confidence"),
  interval_level = 0.95,
  ...
)
```

Arguments

object	A beezdemand_nlme object.
newdata	Optional data frame for which to make predictions. Must contain x_var and all factors specified in the original model. If group-level predictions are desired (level=1), the id_var column from the original fit must also be present in newdata and its levels should correspond to those in the original data for meaningful random effect application. If NULL, predictions are made for the data used in fitting the model.

type	One of "response" (default), "link", "population", or "individual". "population" and "individual" are aliases that set level to 0 or 1, respectively.
level	Integer, prediction level for nlme::predict.nlme(): • 0: Population predictions (based on fixed effects only). • 1 (or higher, up to number of grouping levels in model): Group-specific predictions (fixed effects + random effects for the specified id_var level). Default is 0.
inv_fun	Optional function to inverse-transform the predictions. Example: If y_var was log10-transformed during fitting and equation_form like "zben" produces predictions on that log10 scale, inv_fun = function(x) 10^x would convert predictions back to the original consumption scale. If equation_form was "simplified" (which models raw Y), inv_fun might be identity or not needed if predictions are already on the desired scale.
se.fit	Logical; if TRUE, includes a .se.fit column (currently NA because standard errors are not implemented for beezdemand_nlme predictions).
interval	One of "none" (default) or "confidence". When requested, .lower/.upper are returned as NA.
interval_level	Confidence level when interval = "confidence". Currently used only for validation.
...	Additional arguments passed to nlme::predict.nlme().

Value

A tibble containing the original newdata columns plus .fitted. When requested, .se.fit and .lower/.upper are included (currently NA).

See Also

[predict.nlme](#)

Examples

```
data(ko)
fit <- fit_demand_mixed(ko, y_var = "y_ll4", x_var = "x",
                        id_var = "monkey", equation_form = "zben")
# Population-level predictions
preds <- predict(fit, level = 0)

# Subject-level predictions
preds_subj <- predict(fit, level = 1)
```

predict.beezdemand_tmb

Predict from TMB Mixed-Effects Demand Model

Description

Predict from TMB Mixed-Effects Demand Model

Usage

```
## S3 method for class 'beezdemand_tmb'
predict(
  object,
  newdata = NULL,
  type = c("response", "parameters", "demand"),
  level = "subject",
  prices = NULL,
  scale = c("model", "natural"),
  correction = TRUE,
  at = NULL,
  ...
)
```

Arguments

object	A beezdemand_tmb object.
newdata	Optional data frame. If NULL, predicts for original data.
type	Character. One of "response" (fitted values on response scale), "parameters" (subject-specific parameters), or "demand" (population demand curve).
level	Character, used when type = "response". "subject" (default) conditions on each subject's random effects and requires the model's ID column in newdata (named id unless a custom id_var was set at fit time); "population" evaluates at the fixed-effect coefficients with all random effects set to zero on the fitting scale (the population-mean curve) and does not require the ID column. Pass c("population", "subject") to return both predictions in one call. Note that "population" is the random-effects-at-zero curve, not a marginal prediction integrating over the random-effects distribution (see Note). Unlike predict.beezdemand_nlme() , which forwards the nlme-style numeric level (0 / 1) to nlme::predict.lme(), this method accepts the character form only; a numeric level is rejected.
prices	Optional numeric vector of prices for population prediction.
scale	Character. Output scale for predictions: "model" returns values on the model's native scale (e.g., LL4-transformed for zben, log for exponential), while "natural" automatically back-transforms to the natural consumption scale. Default is "model" for backward compatibility. When scale = "natural" and equation = "exponential", the lognormal re-transformation correction $\exp(\sigma_e^2/2)$ is applied by default to produce

	the conditional mean (not median). Set <code>correction = FALSE</code> to obtain the median (geometric mean) instead. For "exponentiated" and "simplified" equations, predictions are already on the natural scale and no correction is needed. For "zben", <code>ll4_inv()</code> is applied; because <code>ll4_inv</code> is nonlinear, this gives the value corresponding to the conditional mean on the LL4 scale (approximately the median on the natural scale).
<code>correction</code>	Logical. If <code>TRUE</code> (default), applies the lognormal retransformation correction when <code>scale = "natural"</code> . Set to <code>FALSE</code> to obtain the median prediction. Only affects the "exponential" equation.
<code>at</code>	Optional named numeric vector/list (e.g. <code>c(dose_c = 1)</code>) giving the covariate value(s) at which to evaluate per-subject Q_0/α when <code>type = "parameters"</code> and the fit has a continuous random-effect slope (TICKET-051). Defaults to each subject's mean of the covariate (= the reference 0 for a centered, balanced design). Ignored (with a warning) otherwise.
<code>...</code>	Additional arguments.

Value

Depends on type:

- "response", level = "subject": tibble of newdata plus a `.fitted` column (the historical column name, retained for backward compatibility).
- "response", level = "population": tibble of newdata plus a `predict.fixed` column.
- "response", level = c("population", "subject"): tibble of newdata plus `predict.fixed` and `predict.id` columns, matching the `nlme::predict.lme(level = 0:1)` schema so `nlme`-based plotting code runs unchanged.
- "parameters": tibble of subject-specific parameters.
- "demand": tibble with `price` and `.fitted` columns.

Note

Population-averaged (marginal) predictions integrating over the random effects distribution are not yet implemented for this model tier. The `type = "demand"` prediction and `level = "population"` both use `RE = 0` (population fixed effects only). For marginal integration accounting for Jensen's inequality, use `predict.beezdemand_hurdle()` with `marginal = TRUE`.

See Also

`predict.beezdemand_nlme()` for the `nlme`-backed equivalent, which uses the numeric level convention.

Examples

```
data(apt)
fit <- fit_demand_tmb(apt, equation = "exponential", verbose = 0)

# Fitted values (subject-conditional, the default)
head(predict(fit, type = "response"))
```

```
# Population-mean predictions: no `id` column needed in newdata
nd <- data.frame(x = c(0.01, 1, 5, 10))
predict(fit, newdata = nd, level = "population")

# Subject-conditional and population side by side in one call
nd_id <- data.frame(x = c(0.01, 1, 5, 10), id = unique(apt$id)[1])
predict(fit, newdata = nd_id, level = c("population", "subject"))

# Population demand curve at specific prices
predict(fit, type = "demand", prices = c(0, 1, 5, 10, 20))

# Subject-level parameters
head(predict(fit, type = "parameters"))
```

predict.cp_model_lm	<i>Predict method for cp_model_lm objects.</i>
---------------------	--

Description

Predict method for cp_model_lm objects.

Usage

```
## S3 method for class 'cp_model_lm'
predict(object, newdata = NULL, ...)
```

Arguments

object	A cp_model_lm object.
newdata	Data frame containing new x values.
...	Additional arguments.

Value

Data frame with predictions.

predict.cp_model_lmer *Predict from a Mixed-Effects Cross-Price Demand Model*

Description

Generates predictions from a mixed-effects cross-price demand model (of class `cp_model_lmer`). The function supports two modes:

Usage

```
## S3 method for class 'cp_model_lmer'
predict(object, newdata = NULL, pred_type = c("fixed", "random"), ...)
```

Arguments

<code>object</code>	A <code>cp_model_lmer</code> object (as returned by <code>fit_cp_linear(type = "mixed", ...)</code>).
<code>newdata</code>	A data frame containing at least an <code>x</code> column. For <code>pred_type = "random"</code> , an <code>id</code> column is required. If absent, the function extracts unique ids from <code>object\$data</code> and expands the grid accordingly. If no ids are available, a default id of 1 is used (with a warning).
<code>pred_type</code>	Character string specifying the type of prediction: either <code>"fixed"</code> (population-level) or <code>"random"</code> (subject-specific). The default is <code>"fixed"</code> .
<code>...</code>	Additional arguments passed to the underlying <code>predict</code> function.

Details

`"fixed"` Returns predictions based solely on the fixed-effects component (using `re.form = NA`).

`"random"` Returns subject-specific predictions (fixed plus random effects) (using `re.form = NULL`).

Value

A data frame containing all columns of `newdata` plus a column `y_pred` with the corresponding predictions.

Examples

```
data(etm)
fit <- fit_cp_linear(etm, type = "mixed")
new_prices <- data.frame(x = c(2, 4, 8, 16, 32, 64))
predict(fit, newdata = new_prices, pred_type = "fixed")
```

predict.cp_model_nls *Predict from a Cross-Price Demand Model (Nonlinear)*

Description

Predict from a Cross-Price Demand Model (Nonlinear)

Usage

```
## S3 method for class 'cp_model_nls'
predict(
  object,
  newdata = NULL,
  inv_fun = identity,
  inverse_fun = deprecated(),
  ...
)
```

Arguments

object	A cross-price model object from fit_cp_nls with return_all=TRUE.
newdata	A data frame containing an 'x' column.
inv_fun	Optional inverse transformation function. Default is identity.
inverse_fun	[Deprecated] Use inv_fun instead.
...	Additional arguments.

Value

A data frame with x values and predicted y values.

print.anova.beezdemand_hurdle
Print Method for ANOVA Comparisons

Description

Print Method for ANOVA Comparisons

Usage

```
## S3 method for class 'anova.beezdemand_hurdle'
print(x, digits = 4, ...)
```

Arguments

x	An anova.beezdemand object.
digits	Number of significant digits to print.
...	Additional arguments (ignored).

Value

Invisibly returns the input object x.

```
print.beezdemand_comparison
```

Print method for beezdemand_comparison objects

Description

Print method for beezdemand_comparison objects

Usage

```
## S3 method for class 'beezdemand_comparison'  
print(x, digits = 3, ...)
```

Arguments

x	A beezdemand_comparison object.
digits	Number of significant digits to display for estimates.
...	Additional arguments (unused).

Value

Invisibly returns the input object x.

```
print.beezdemand_diagnostic_plots
```

Print Method for Diagnostic Plots

Description

Print Method for Diagnostic Plots

Usage

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

Arguments

x A beezdemand_diagnostic_plots object (list of ggplot objects).
... Additional arguments (ignored).

Value

Invisibly returns the input object x.

print.beezdemand_diagnostics

Print Method for Model Diagnostics

Description

Print Method for Model Diagnostics

Usage

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

Arguments

x A beezdemand_diagnostics object.
... Additional arguments (ignored).

Value

Invisibly returns the input object x.

print.beezdemand_fixed

Print Method for beezdemand_fixed

Description

Print Method for beezdemand_fixed

Usage

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

Arguments

x	A beezdemand_fixed object
...	Additional arguments (ignored)

Value

Invisibly returns the input object x.

```
print.beezdemand_fixed_grouped
```

Print Method for beezdemand_fixed_grouped

Description

Print Method for beezdemand_fixed_grouped

Usage

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

Arguments

x	A beezdemand_fixed_grouped object
...	Additional arguments (ignored)

Value

Invisibly returns the input object x.

```
print.beezdemand_hurdle
```

Print Method for Hurdle Demand Model

Description

Print Method for Hurdle Demand Model

Usage

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

Arguments

x An object of class beezdemand_hurdle.
 ... Additional arguments (currently unused).

Value

Invisibly returns the input object x.

```
print.beezdemand_model_comparison
```

Print Method for Model Comparison

Description

Print Method for Model Comparison

Usage

```
## S3 method for class 'beezdemand_model_comparison'
print(x, digits = 4, ...)
```

Arguments

x A beezdemand_model_comparison object.
 digits Number of significant digits to print.
 ... Additional arguments (ignored).

Value

Invisibly returns the input object x.

```
print.beezdemand_nlme
```

Print Method for beezdemand_nlme Objects

Description

Provides a concise summary of a beezdemand_nlme object, typically displaying the call, model specifications, and key results from the nlme fit if successful.

Usage

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

Arguments

<code>x</code>	An object of class <code>beezdemand_nlme</code> .
<code>digits</code>	Minimal number of significant digits, see <code>print.default</code> .
<code>...</code>	Additional arguments passed to <code>print.nlme</code> if the model exists.

Value

Invisibly returns the original object `x`.

Examples

```
data(ko)
fit <- fit_demand_mixed(ko, y_var = "y_ll4", x_var = "x",
                        id_var = "monkey", equation_form = "zben")
print(fit)
```

```
print.beezdemand_summary
```

Print Method for beezdemand Summary Objects

Description

Fallback print method for summary objects inheriting from `beezdemand_summary`. Specific summary classes should implement their own `print.summary.*` methods for detailed output; this provides a minimal fallback.

Usage

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

Arguments

<code>x</code>	A summary object with class including <code>beezdemand_summary</code> .
<code>...</code>	Additional arguments (unused).

Value

Invisibly returns `x`.

```
print.beezdemand_systematicity
```

Print Method for beezdemand_systematicity

Description

Print Method for beezdemand_systematicity

Usage

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

Arguments

x	A beezdemand_systematicity object
...	Additional arguments (ignored)

Value

Invisibly returns the input object x.

```
print.beezdemand_tmb
```

Print Method for TMB Mixed-Effects Demand Model

Description

Print Method for TMB Mixed-Effects Demand Model

Usage

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

Arguments

x	An object of class beezdemand_tmb.
...	Additional arguments (currently unused).

Value

Invisibly returns the input object x.

Examples

```
data(apt)
fit <- fit_demand_tmb(apt, equation = "exponential", verbose = 0)
print(fit)
```

print.cp_model_lm	<i>Print a Cross-Price Demand Model (Linear)</i>
-------------------	--

Description

Print a Cross-Price Demand Model (Linear)

Usage

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

Arguments

x	A cp_model_lm object.
...	Additional arguments (unused).

Value

Invisibly returns x.

print.cp_model_lmer	<i>Print a Cross-Price Demand Model (Mixed-Effects)</i>
---------------------	---

Description

Print a Cross-Price Demand Model (Mixed-Effects)

Usage

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

Arguments

x	A cp_model_lmer object.
...	Additional arguments (unused).

Value

Invisibly returns x.

print.cp_model_nls	<i>Print a Cross-Price Demand Model (Nonlinear)</i>
--------------------	---

Description

Print a Cross-Price Demand Model (Nonlinear)

Usage

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

Arguments

x	A cp_model_nls object.
...	Additional arguments (unused).

Value

Invisibly returns x.

print.cp_posthoc	<i>Print method for cp_posthoc objects</i>
------------------	--

Description

Print method for cp_posthoc objects

Usage

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

Arguments

x	A cp_posthoc object
...	Additional arguments passed to print

Value

Invisibly returns the input object x.

```
print.summary.beezdemand_fixed
```

Print Method for summary.beezdemand_fixed

Description

Print Method for summary.beezdemand_fixed

Usage

```
## S3 method for class 'summary.beezdemand_fixed'
print(x, digits = 4, n = 20, ...)
```

Arguments

x	A summary.beezdemand_fixed object
digits	Number of significant digits to print
n	Number of subjects (ids) to print (default 20)
...	Additional arguments (ignored)

Value

Invisibly returns the input object x.

```
print.summary.beezdemand_hurdle
```

Print Summary of Hurdle Demand Model

Description

Print Summary of Hurdle Demand Model

Usage

```
## S3 method for class 'summary.beezdemand_hurdle'
print(x, digits = 4, n = Inf, ...)
```

Arguments

x	An object of class summary.beezdemand_hurdle.
digits	Number of significant digits to print. Default is 4.
n	Number of rows to print for any tables (unused for this class).
...	Additional arguments passed to print.

Value

Invisibly returns the input object `x`.

```
print.summary.beezdemand_nlme
```

Print method for summary.beezdemand_nlme

Description

Print method for summary.beezdemand_nlme

Usage

```
## S3 method for class 'summary.beezdemand_nlme'
print(x, digits = 4, n = Inf, ...)
```

Arguments

<code>x</code>	A summary.beezdemand_nlme object
<code>digits</code>	Number of significant digits to print
<code>n</code>	Number of rows to print for any tables (unused for this class).
<code>...</code>	Additional arguments (ignored)

Value

Invisibly returns the input object `x`.

```
print.summary.beezdemand_systematicity
```

Print Method for summary.beezdemand_systematicity

Description

Print Method for summary.beezdemand_systematicity

Usage

```
## S3 method for class 'summary.beezdemand_systematicity'
print(x, n = 20, ...)
```

Arguments

<code>x</code>	A summary.beezdemand_systematicity object
<code>n</code>	Number of IDs to display (default 20)
<code>...</code>	Additional arguments (ignored)

Value

Invisibly returns the input object x.

```
print.summary.beezdemand_tmb
```

Print Method for TMB Model Summary

Description

Print Method for TMB Model Summary

Usage

```
## S3 method for class 'summary.beezdemand_tmb'
print(x, digits = 4, ...)
```

Arguments

x	A summary.beezdemand_tmb object.
digits	Number of significant digits.
...	Additional arguments.

Value

Invisibly returns the input object x; called for its side effect of printing a formatted model summary to the console.

```
print.summary.cp_model_lm
```

Print method for summary.cp_model_lm objects.

Description

Print method for summary.cp_model_lm objects.

Usage

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

Arguments

x	A summary.* object
...	Unused

Value

Invisibly returns the input object x.

```
print.summary.cp_model_lmer
```

Print method for summary.cp_model_lmer objects.

Description

Print method for summary.cp_model_lmer objects.

Usage

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

Arguments

x	A summary.* object
...	Unused

Value

Invisibly returns the input object x.

```
print.summary.cp_model_nls
```

Print method for summary.cp_model_nls objects

Description

Print method for summary.cp_model_nls objects

Usage

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

Arguments

x	A summary.* object
...	Unused

Value

Invisibly returns the input object x.

```
print.summary.cp_unsystematic
```

Print Method for Cross-Price Unsystematic Summary

Description

Print Method for summary.cp_unsystematic

Usage

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

Arguments

x	A summary.cp_unsystematic object.
...	Additional arguments (ignored).

Value

Invisibly returns the input object x.

```
print_mc_summary
```

Print Monte Carlo Simulation Results

Description

Prints a formatted summary of Monte Carlo simulation results.

Usage

```
print_mc_summary(mc_results, digits = 3)
```

Arguments

mc_results	Output from run_hurdle_monte_carlo .
digits	Number of digits to display. Default is 3.

Value

Invisibly returns the input mc_results object.

Examples

```
# Tiny run for illustration (use n_sim >= 200 for a real calibration study)
mc_results <- run_hurdle_monte_carlo(n_sim = 5, n_subjects = 30, seed = 123,
                                   verbose = FALSE)

print_mc_summary(mc_results)
```

pseudo_ll4_trans *Create a Pseudo-Log LL4 Transformation Object for ggplot2*

Description

Generates a `scales::trans` object using the ll4 transformation. This transformation object can be passed to the `trans` argument of `ggplot2::scale_x_continuous` or `ggplot2::scale_y_continuous`. It is intended for non-negative data and is defined at zero. The "pseudo" aspect is conceptual, similar to `pseudo_log_trans` in that it handles a range including zero, but the transformation is ll4.

Usage

```
pseudo_ll4_trans(lambda = 4)
```

Arguments

`lambda` A positive numeric scalar, the lambda parameter for the ll4 transformation. Default is 4.

Value

A `trans` object (from the `scales` package).

See Also

[ll4](#), [ll4_inv](#), [trans_new](#)

Examples

```
if (require(ggplot2) && require(scales)) {
  set.seed(123)
  df <- data.frame(
    x_vals = c(0, 0.01, 0.1, 1, 10, 100, 1000, NA), # Include 0 and NA
    y_vals = c(0, 10, 50, 100, 500, 1000, 2000, 50)
  )

  # Using pseudo_ll4_trans for the y-axis
  ggplot(df, aes(x = x_vals, y = y_vals)) +
    geom_point() +
    scale_y_continuous(trans = pseudo_ll4_trans(lambda = 4),
                      name = "Y-Values (Pseudo-LL4 Scale)") +
```

```

    ggtitle("Y-Axis with Pseudo-LL4 Transformation")

    # Using pseudo_ll4_trans for the x-axis
    ggplot(df, aes(x = x_vals, y = y_vals)) +
      geom_point() +
      scale_x_continuous(trans = pseudo_ll4_trans(lambda = 2), # Different lambda
                        name = "X-Values (Pseudo-LL4 Scale)") +
      ggtitle("X-Axis with Pseudo-LL4 Transformation")
  }

```

ranef.beezdemand_nlme *Extract Random Effects from a beezdemand_nlme Model*

Description

S3 method for ranef for objects of class beezdemand_nlme. Extracts the random effects from the fitted nlme model.

Usage

```
## S3 method for class 'beezdemand_nlme'
ranef(object, ...)
```

Arguments

object	A beezdemand_nlme object.
...	Additional arguments passed to nlme::ranef().

Value

A data frame (or list of data frames if multiple levels of grouping) of random effects, as returned by ranef.nlme().

See Also

[coef.beezdemand_nlme](#), [fixef.beezdemand_nlme](#)

ranef.beezdemand_tmb *Extract Random Effects from TMB Model*

Description

Returns subject-level random effect deviations on the natural (log) scale. These are the Cholesky-transformed deviations, not standardized scores. To obtain the standardized random effects (u matrix), access `object$tmb_obj` directly.

Usage

```
## S3 method for class 'beezdemand_tmb'
ranef(object, ...)
```

Arguments

<code>object</code>	A beezdemand_tmb object.
<code>...</code>	Additional arguments.

Value

Data frame with subject-level random effects. Columns:

- `id`: subject identifier
- `b_i`, `c_i` (when present): first-column convenience aliases for `q0_(Intercept)` and `alpha_(Intercept)`. Preserved for backward compatibility with older callers.
- `q0_<term>`: per-block random-effect coefficients for log-Q0, one column per random-effects design column from the parsed block structure. For factor-expanded or multi-block fits, these expose the per-condition slope REs that `b_i` / `c_i` alone do not surface.
- `alpha_<term>`: analogous columns for log-alpha.

Examples

```
data(apt)
fit <- fit_demand_tmb(apt, equation = "exponential", verbose = 0)
head(nlme::ranef(fit))
```

ranef.cp_model_lmer	<i>Extract Random Effects from Mixed-Effects Cross-Price Model</i>
---------------------	--

Description

Extract Random Effects from Mixed-Effects Cross-Price Model

Usage

```
## S3 method for class 'cp_model_lmer'
ranef(object, ...)
```

Arguments

object	A cp_model_lmer object
...	Additional arguments passed to ranef

Value

List of random effects

RecodeOutliers	<i>Recode Outliers</i>
----------------	------------------------

Description

Recodes outliers

Usage

```
RecodeOutliers(df, outval = 3.29, unitshigher = 0)
```

Arguments

df	A dataframe of consumption values
outval	Values greater/less than or equal to this number (specified in standard deviations) will be recoded. Default is 3.29SD as specified by Tabachnick and Fidell (2013)
unitshigher	Outliers identified by outval will be coded to a certain number of units higher/lower than the greatest nonoutlier value. Default is 0 units.

Details

Recodes outliers using Tabachnick and Fidell's (2013) criteria. A variable is standardized and values that are greater/less than or equal to a specified outlier value (specified in standard deviations; default 3.29SD) are recoded to a certain number of units (default 0) higher/lower than the greatest nonoutlier value. Disregards 'NA' values.

Value

Invisibly, a dataframe with original and recoded (if any) values

Author(s)

Brent Kaplan bkaplan.ku@gmail.com

Examples

```
## If any outliers are detected, they would be coded as 1 unit higher

emp <- GetEmpirical(apt)
RecodeOutliers(emp[, c(2:6)], unitshigher = 1)
```

ReplaceZeros	<i>Replace Zeros</i>
--------------	----------------------

Description

Replaces 0 values

Usage

```
ReplaceZeros(dat, nrepl = 1, replnum = 0.01)
```

Arguments

dat	Dataframe (long form)
nrepl	Number of zeros to replace with replacement value (replnum). Can accept either a number or "all" if all zeros should be replaced. Default is to replace the first zero only.
replnum	Value to replace zeros. Default is .01

Details

Replaces specified number of 0s with replacement value.

Value

Dataframe (long form)

Author(s)

Brent Kaplan bkaplan.ku@gmail.com

Examples

```
## Replace all zeros with .01
ReplaceZeros(apt, nrepl = "all", replnum = .01)
```

```
residuals.beezdemand_hurdle
```

Residuals for a beezdemand_hurdle fit

Description

Response-scale residuals against the marginal (default) or conditional fitted values. `type = "pearson"` returns the Part-II standardized residual on the model (log-consumption) scale, $(\log(y) - \mu_i) / \sigma_e$, where μ_i is the subject-conditional linear predictor (`predict(type = "link")`) and σ_e is `exp(coef[["logsigma_e"]])`; observations with $y = 0$ have no Part-II residual and are NA. (Before 0.3.0 the raw-scale residual was divided by the log-scale σ_e , which changed with the consumption unit.)

Usage

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

Arguments

<code>object</code>	A <code>beezdemand_hurdle</code> object.
<code>type</code>	One of "response" (default) or "pearson".
<code>marginal</code>	Passed to <code>fitted.beezdemand_hurdle()</code> . Default TRUE.
<code>...</code>	Unused.

Value

Numeric vector of length `nobs(object)`.

See Also

[fitted.beezdemand_hurdle\(\)](#).

residuals.beezdemand_tmb

Residuals for a beezdemand_tmb fit

Description

Default returns response residuals ($y_{\text{on_scale}} - \text{fitted}$) on the model's native scale. `type = "pearson"` divides by the residual SD on the model scale ($\exp(\text{coef}[[\text{"logsigma_e"}]])$). Requesting `type = "pearson"` with `scale = "natural"` falls back to `type = "response"` with a message because a response-scale residual SD is not identified for the exponential/zben variants without a separate variance assumption.

Usage

```
## S3 method for class 'beezdemand_tmb'
residuals(
  object,
  type = c("response", "pearson"),
  scale = c("model", "natural"),
  level = c("subject", "population"),
  ...
)
```

Arguments

<code>object</code>	A <code>beezdemand_tmb</code> object.
<code>type</code>	One of "response" (default) or "pearson".
<code>scale</code>	One of "model" (default) or "natural".
<code>level</code>	One of "subject" (default; conditions on the subject random effects) or "population" (random effects set to zero, giving the population-mean values). See predict.beezdemand_tmb() .
<code>...</code>	Unused.

Value

Numeric vector of length `nobs(object)`.

See Also

[fitted.beezdemand_tmb\(\)](#), [augment.beezdemand_tmb\(\)](#).

Examples

```
data(apt)
fit <- fit_demand_tmb(apt, equation = "exponential", verbose = 0)
head(residuals(fit))
head(residuals(fit, type = "pearson"))
```

run_hurdle_monte_carlo

Run Monte Carlo Simulation Study for Hurdle Demand Model

Description

Runs a Monte Carlo simulation study to assess model performance, including bias, standard error estimates, and confidence interval coverage.

Usage

```
run_hurdle_monte_carlo(
  n_sim = 100,
  n_subjects = 100,
  true_params = NULL,
  n_random_effects = 2,
  prices = seq(0, 11, by = 0.5),
  stop_at_zero = TRUE,
  verbose = TRUE,
  seed = NULL
)
```

Arguments

n_sim	Number of simulated datasets. Default is 100.
n_subjects	Number of subjects per dataset. Default is 100.
true_params	Named list of true parameter values. If NULL, defaults are used from simulate_hurdle_data .
n_random_effects	Number of random effects (2 or 3). Default is 2.
prices	Numeric vector of prices. Default is seq(0, 11, by = 0.5).
stop_at_zero	Logical; if TRUE in simulation, subjects stop after first zero. Default is TRUE.
verbose	Logical; print progress. Default is TRUE.
seed	Random seed for reproducibility.

Value

A list with:

estimates Data frame of parameter estimates from each converged simulation (includes non-PD-Hessian replicates, flagged via a hessian_pd column, for callers that want them)

true_params True parameter values used

summary Summary statistics including bias, SE ratio, and coverage, computed only from replicates that converged with a positive-definite Hessian (diagnostics\$status == "clean"); converged-but-non-PD and converged-but-Hessian-unavailable replicates are excluded since their SEs are unreliable or unknown

n_converged Number of simulations that converged (regardless of Hessian positive-definiteness/availability; unchanged definition)

n_sim Total number of simulations attempted

diagnostics Data frame with one row per simulation: `sim_id`, `status` ("error", "nonconverged", "converged_non_pd", "converged_hessian_unavailable", or "clean"), `converged`, `hessian_pd` (TRUE/FALSE/NA; NA means `sdreport()` itself failed, a different condition from an explicit non-PD Hessian), `opt_convergence`, and `opt_message`

n_hessian_not_pd Number of converged replicates excluded from summary because `hessian_pd` was explicitly FALSE

n_hessian_unavailable Number of converged replicates excluded from summary because `hessian_pd` was NA (Hessian PD status unavailable)

See Also

[simulate_hurdle_data](#), [fit_demand_hurdle](#)

Examples

```
# Tiny simulation study for demonstration (use n_sim >= 200 in practice)
mc_results <- run_hurdle_monte_carlo(n_sim = 5, n_subjects = 30, seed = 123,
                                     verbose = FALSE)

# View summary
print(mc_results$summary)

# Check convergence rate
cat("Convergence rate:", mc_results$n_converged / mc_results$n_sim, "\n")
```

scale_color_beezdemand

beezdemand Color Scale (Discrete)

Description

beezdemand Color Scale (Discrete)

Usage

```
scale_color_beezdemand(...)
```

Arguments

... Additional arguments passed to `ggplot2::scale_color_manual`.

Value

A ggplot2 discrete color scale.

Examples

```
library(ggplot2)
ggplot(iris, aes(Sepal.Length, Sepal.Width, color = Species)) +
  geom_point() +
  scale_color_beezdemand()
```

scale_fill_beezdemand *beezdemand Fill Scale (Discrete)*

Description

beezdemand Fill Scale (Discrete)

Usage

```
scale_fill_beezdemand(...)
```

Arguments

... Additional arguments passed to `ggplot2::scale_fill_manual`.

Value

A ggplot2 discrete fill scale.

Examples

```
library(ggplot2)
ggplot(iris, aes(Species, Sepal.Length, fill = Species)) +
  geom_boxplot() +
  scale_fill_beezdemand()
```

scale_ll4

*Create an LL4-like Scale for ggplot2 Axes***Description**

This function generates a ggplot2 continuous scale that applies the ll4 transformation (and its inverse ll4_inv) to an axis. This is useful for visualizing data spanning multiple orders of magnitude while handling zeros.

Usage

```
scale_ll4(..., lambda = 4)
```

Arguments

...	Arguments passed on to <code>ggplot2::scale_y_continuous</code> or <code>ggplot2::scale_x_continuous</code> (e.g., <code>name</code> , <code>breaks</code> , <code>labels</code>).
lambda	A positive numeric scalar, the lambda parameter for the ll4 transformation. Default is 4.

Value

A ggplot2 scale object.

See Also

[ll4](#), [ll4_inv](#), [trans_new](#)

Examples

```
if (require(ggplot2) && require(scales)) {
  set.seed(123)
  df <- data.frame(
    x = 1:100,
    y_raw = c(0, 0.1, 0.5, 1, 5, 10, 50, 100, 500, 1000, sample(1:2000, 90, replace = TRUE))
  )

  # Plot with y-axis on LL4 scale
  ggplot(df, aes(x = x, y = y_raw)) +
    geom_point() +
    scale_ll4(name = "Y-axis (LL4 Scale)", lambda = 4) +
    ggtitle("Data with LL4 Transformed Y-Axis")

  # Can also be used for x-axis by replacing scale_y_continuous in its definition
  # Or by creating a scale_x_ll4 variant.
}
```

simulate_hurdle_data	<i>Simulate Data from Two-Part Mixed Effects Hurdle Demand Model</i>
----------------------	--

Description

Generates simulated demand data from the two-part hurdle model. Useful for Monte Carlo simulation studies, power analyses, and model validation.

Usage

```
simulate_hurdle_data(  
  n_subjects = 100,  
  prices = seq(0, 11, by = 0.5),  
  beta0 = -2,  
  beta1 = 1,  
  log_q0 = log(10),  
  logQ0 = deprecated(),  
  k = 2,  
  alpha = 0.5,  
  sigma_a = 1,  
  sigma_b = 0.5,  
  sigma_c = 0.1,  
  rho_ab = 0.3,  
  rho_ac = 0,  
  rho_bc = 0,  
  sigma_e = 0.3,  
  epsilon = 0.001,  
  n_random_effects = 2,  
  stop_at_zero = TRUE,  
  seed = NULL,  
  part2 = c("koff", "snd"),  
  rho_ab_raw = NULL,  
  rho_ac_raw = NULL,  
  rho_bc_raw = NULL  
)
```

Arguments

n_subjects	Number of subjects to simulate. Default is 100.
prices	Numeric vector of prices at which to simulate consumption. Default is seq(0, 11, by = 0.5).
beta0	Intercept for Part I (logistic). Default is -2.
beta1	Slope for Part I (on log(price + epsilon)). Default is 1.
log_q0	Log of intensity parameter (Q0). Default is log(10), meaning Q0 = 10. To specify Q0 directly, use log_q0 = log(your_Q0).

logQ0	[Deprecated] Use log_q0 instead.
k	Scaling parameter for demand decay. Default is 2.
alpha	Elasticity parameter controlling rate of demand decay. Default is 0.5.
sigma_a	Standard deviation of random intercept for Part I. Default is 1.
sigma_b	Standard deviation of random intercept for Part II. Default is 0.5.
sigma_c	Standard deviation of random slope for alpha (only used if n_random_effects = 3). Default is 0.1.
rho_ab	Correlation between a_i and b_i. Default is 0.3.
rho_ac	Correlation between a_i and c_i. Default is 0.
rho_bc	Correlation between b_i and c_i. Default is 0.
sigma_e	Residual standard deviation. Default is 0.3.
epsilon	Small constant for log(price + epsilon). Default is 0.001.
n_random_effects	Number of random effects (2 or 3). Default is 2.
stop_at_zero	Logical; if TRUE, stop generating observations for a subject once zero consumption is observed. This means subjects will have varying numbers of observations. Set to FALSE to generate all prices for all subjects. Default is TRUE.
seed	Optional random seed for reproducibility.
part2	Character. Positive-part (Part II) generator: "koff" (the original Zhao et al. (2016) / Koffarnus-style generator; default, backward compatible) or "snd" (matches src/HurdleDemand3RE_SND.h / src/HurdleDemand2RE_SND.h exactly; see Details).
rho_ab_raw	Only used when part2 = "snd". Pre-tanh() raw value for the a/b random-effect correlation, mirroring the TMB model's own rho_ab_raw coefficient (actual correlation is tanh(rho_ab_raw)). Default NULL uses atanh(0.3) (actual correlation 0.3, matching the "koff" generator's default rho_ab).
rho_ac_raw	Only used when part2 = "snd" and n_random_effects = 3. Pre-tanh() raw value for the a/c random-effect correlation. Default NULL uses 0 (actual correlation 0).
rho_bc_raw	Only used when part2 = "snd" and n_random_effects = 3. Raw value for the b/c <i>partial</i> correlation (see Details); the actual b/c correlation is not tanh(rho_bc_raw) directly. Default NULL uses 0 (actual correlation 0).

Note: part2 and rho*_raw were added after seed in the argument list so that pre-existing positional calls (whose 19th and last positional argument was seed) continue to bind correctly instead of landing on part2 (where match.arg() would error). Always pass part2/rho*_raw by name.

Details

Part I (Zero vs Positive), shared by both part2 generators:

$$\text{logit}(P(Y = 0)) = \beta_0 + \beta_1 \cdot \log(\text{price} + \epsilon) + a_i$$

Part II, part2 = "koff" (**Zhao et al., 2016; mirrors** src/HurdleDemand3RE.h):

$$\alpha_i = \exp(\log \alpha + c_i)$$

$$\log(Y|Y > 0) = (\log Q_0 + b_i) + k \cdot (\exp(-\alpha_i \cdot price) - 1) + \epsilon$$

Part II, part2 = "snd" (**exactly mirrors** src/HurdleDemand3RE_SND.h / src/HurdleDemand2RE_SND.h, **i.e. a log-linear/SND mean with lognormal errors and no k**):

$$Q_{0,i} = \exp(\log Q_0 + b_i), \quad \alpha_i = \exp(\log \alpha + c_i)$$

$$\log(Y|Y > 0) = (\log Q_0 + b_i) - \alpha_i Q_{0,i} \cdot price + \epsilon$$

with $\epsilon \sim N(0, \sigma_\epsilon^2)$ (residual on the log-consumption scale) in both generators. When n_random_effects = 2, c_i = 0 for every subject in both generators, so alpha_i reduces to the fixed population alpha.

Random effects (a_i, b_i) or (a_i, b_i, c_i) are drawn from a multivariate normal distribution with mean zero and covariance built from sigma_a, sigma_b, sigma_c (SDs, natural scale) and the specified correlations. For part2 = "koff" (unchanged from previous releases), rho_ab, rho_ac, rho_bc are the actual (final) correlations, checked for a jointly positive-definite covariance matrix. For part2 = "snd", correlations instead mirror the TMB model's own raw-parameter -> actual correlation mapping exactly (any raw values are guaranteed to give a valid PD matrix, so no PD check is needed): rho_ab = tanh(rho_ab_raw), rho_ac = tanh(rho_ac_raw), and rho_bc via the LKJ-Cholesky partial-correlation transform

$$\rho_{bc} = \rho_{ab}\rho_{ac} + \tanh(\rho_{bc,raw})\sqrt{(1 - \rho_{ab}^2)(1 - \rho_{ac}^2)}.$$

This lets a user plug a fitted part2 = "snd" model's own rho_ab_raw/rho_ac_raw/rho_bc_raw coefficients directly into the simulator for a parametric bootstrap or recovery study.

Value

A data frame with columns:

id Subject identifier

x Price value

y Simulated consumption (may include zeros)

delta Indicator for zero consumption (1 = zero, 0 = positive)

a_i Subject-specific random effect for Part I

b_i Subject-specific random effect for Part II

c_i Subject-specific random effect for alpha (if n_random_effects = 3)

See Also

[fit_demand_hurdle](#), [run_hurdle_monte_carlo](#)

Examples

```
# Simulate with default parameters (2 RE model)
sim_data <- simulate_hurdle_data(n_subjects = 100, seed = 123)
head(sim_data)

# Simulate with custom prices
apt_prices <- c(0, 0.25, 0.5, 1, 1.5, 2, 2.5, 3, 4, 5, 6, 7, 8, 9, 10)
sim_apt <- simulate_hurdle_data(n_subjects = 100, prices = apt_prices, seed = 123)

# Simulate with custom parameters (Q0 = 15, alpha = 0.1)
sim_custom <- simulate_hurdle_data(
  n_subjects = 100,
  log_q0 = log(15),
  alpha = 0.1,
  seed = 123
)

# Simulate 3 RE model
sim_3re <- simulate_hurdle_data(
  n_subjects = 100,
  n_random_effects = 3,
  sigma_c = 0.1,
  seed = 456
)
```

SimulateDemand	<i>Simulate Demand Data</i>
----------------	-----------------------------

Description

Simulate demand data

Usage

```
SimulateDemand(nruns = 10, setparams, sdindex, x, outdir = NULL, fn = NULL)
```

Arguments

nruns	Number of runs. Default value is 10
setparams	A 6x1 matrix (or 6 element vector) containing (in order) mean log10alpha, sd log10alpha, mean log10q0, sd log10q0, k, sd of consumption values across all prices
sdindex	A vector of n length of sd consumption values for n prices
x	A vector of n prices
outdir	Optional. Directory to save results. Must end with a "/"
fn	Optional. Filename of saved RData object

Details

Generates and saves simulated datasets in the manner specified in Koffarnus, Franck, Stein, & Bickel (2015).

Value

Invisibly a list consisting of: rounded consumption values, unrounded consumption values, simulation parameters, and inState and outState of seeds.

Author(s)

Brent Kaplan bkaplan.ku@gmail.com

Examples

```
## set values
setparams <- vector(length = 4)
setparams <- c(-2.5547, .702521, 1.239893, .320221, 3.096, 1.438231)
names(setparams) <- c("alphalm", "alphalsd", "q0lm", "q0lsd", "k", "yvalssd")
sdindex <- c(2.1978, 1.9243, 1.5804, 1.2465, 0.8104, 0.1751, 0.0380, 0.0270)
x <- c(.1, 1, 3, 10, 30, 100, 300, 1000)
set.seed(1234)
sim <- SimulateDemand(nruns = 1, setparams = setparams, sdindex = sdindex, x = x)
sim
```

summary.beezdemand_fixed

Summary Method for beezdemand_fixed

Description

Summary Method for beezdemand_fixed

Usage

```
## S3 method for class 'beezdemand_fixed'
summary(object, report_space = c("natural", "log10"), ...)
```

Arguments

object	A beezdemand_fixed object
report_space	Character. Reporting space for core parameters. One of: "natural": report natural-scale parameters (default) "log10": report log10()-scale parameters when defined
...	Additional arguments (ignored)

Value

A summary.beezdemand_fixed object (inherits from beezdemand_summary)

```
summary.beezdemand_fixed_grouped
```

Summary Method for beezdemand_fixed_grouped

Description

Returns a named list of per-group summaries.

Usage

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

Arguments

object	A beezdemand_fixed_grouped object
...	Additional arguments passed to summary.beezdemand_fixed()

Value

A named list of per-group summary objects

```
summary.beezdemand_hurdle
```

Summarize a Hurdle Demand Model Fit

Description

Provides a summary of a fitted hurdle demand model, including fixed effects, variance components, correlations, and fit statistics.

Usage

```
## S3 method for class 'beezdemand_hurdle'
summary(object, report_space = c("natural", "log10", "internal"), ...)
```

Arguments

object	An object of class beezdemand_hurdle from fit_demand_hurdle .
report_space	Character. Reporting space for core demand parameters. One of: "internal": report internal/fitting parameters (default internal naming) "natural": report natural-scale parameters when a natural mapping exists "log10": report log₁₀()-scale parameters when a mapping exists estimate/std.error follow report_space; statistic/p.value are always computed on the estimation scale (the Wald test is defined there and is not recomputed after back-transforming).
...	Additional arguments (currently unused).

Value

An object of class `summary.beezdemand_hurdle` (also inherits from `beezdemand_summary`) containing:

call The original function call

model_class "beezdemand_hurdle"

backend "TMB"

coefficients Tibble of fixed effects with estimates, SEs, z-values, p-values

coefficients_matrix Matrix form for printing, on the internal (estimation) scale regardless of `report_space` (legacy compatibility)

variance_components Matrix of variance/covariance estimates

correlations Matrix of correlation estimates

n_subjects Number of subjects

nobs Number of observations

converged Logical indicating convergence

logLik Log-likelihood at convergence

AIC Akaike Information Criterion

BIC Bayesian Information Criterion

group_metrics Group-level Pmax and Omax

individual_metrics Summary of individual-level parameters

notes Character vector of warnings/notes

Examples

```
data(apt)
fit <- fit_demand_hurdle(apt, y_var = "y", x_var = "x", id_var = "id")
summary(fit)
```

summary.beezdemand_nlme

Summary method for beezdemand_nlme

Description

Returns a structured summary object containing model coefficients, fit statistics, and random effects information.

Usage

```
## S3 method for class 'beezdemand_nlme'
summary(
  object,
  report_space = c("natural", "log10"),
  df_method = c("containment", "between"),
  ...
)
```

Arguments

<code>object</code>	A beezdemand_nlme object
<code>report_space</code>	Character. Reporting space for core parameters. One of "natural" or "log10" (match.arg default "natural"). estimate/std.error follow this scale; statistic/p.value are always on the estimation scale (nlme's native containment-t test, which is transformation-invariant).
<code>df_method</code>	Character. Degrees of freedom used for the fixed-effect t tests. "containment" (default) reports nlme's own containment df unchanged. nlme's containment rule assigns the observation-level residual degrees of freedom to every fixed effect, including between-subject terms (a group factor), for which the effective sample size is the number of subjects; the default df and p-values for such terms are therefore anticonservative. "between" replaces the df of each between-subject term (a coefficient whose design column is constant within every subject, excluding the parameter intercepts) with $n_{\text{subjects}} - \text{rank}(X_{\text{between}})$, where X_{between} is that parameter's subject-level design (intercept plus its between-subject columns), and recomputes the p-value from the unchanged t statistic. Intercepts and within-subject terms keep containment df in both modes (Pinheiro & Bates, 2000, section 2.4.2). Estimates and standard errors never change. Satterthwaite / Kenward-Roger df are not available for nlme::nlme(). The TMB backend reports an asymptotic z test instead.
<code>...</code>	Additional arguments (passed to summary.nlme)

Value

A summary.beezdemand_nlme object (inherits from beezdemand_summary) with fields including:

- `call`: The original function call
- `model_class`: "beezdemand_nlme"
- `backend`: "nlme"
- `equation_form`: The equation form used ("zben" or "simplified")
- `coefficients`: Tibble of fixed effects with std.error, statistic, p.value and df (the degrees of freedom the p-value uses; see `df_method`)
- `df_method`: The `df_method` in effect
- `random_effects`: VarCorr output for random effects
- `logLik`, AIC, BIC: Model fit statistics

References

Pinheiro, J. C., & Bates, D. M. (2000). *Mixed-Effects Models in S and S-PLUS*. Springer. Section 2.4.2.

```
summary.beezdemand_systematicity
```

Summary Method for beezdemand_systematicity

Description

Summary Method for beezdemand_systematicity

Usage

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

Arguments

object	A beezdemand_systematicity object
...	Additional arguments (ignored)

Value

A summary.beezdemand_systematicity object

```
summary.beezdemand_tmb
```

Summarize a TMB Mixed-Effects Demand Model Fit

Description

Summarize a TMB Mixed-Effects Demand Model Fit

Usage

```
## S3 method for class 'beezdemand_tmb'
summary(object, report_space = c("natural", "log10", "internal"), ...)
```

Arguments

object	An object of class beezdemand_tmb.
report_space	Character. Reporting space for core demand parameters. One of "internal", "natural", "log10". estimate/std.error are reported on this scale; statistic/p.value are always computed on the estimation scale (the Wald test is defined there and is not recomputed after back-transforming, so on the natural scale statistic != estimate/std.error, by design).
...	Additional arguments (currently unused).

Value

An object of class summary.beezdemand_tmb (also inherits from beezdemand_summary). The variance_components element reports the Q0 and alpha random-effect SDs on the **log10 scale**: the random effects are estimated on the natural-log scale internally and divided by $\log(10)$ for reporting, so they are directly comparable with nlme::VarCorr() on a fit_demand_mixed() fit using the default param_space = "log10". The residual SD is reported on the model's likelihood scale (equation-dependent) and the random-effect correlations are scale-invariant; neither is rescaled.

Examples

```
data(apt)
fit <- fit_demand_tmb(apt, equation = "exponential", verbose = 0)
summary(fit)
summary(fit, report_space = "log10")
```

summary.cp_model_lm *Summary method for cp_model_lm objects.*

Description

Summary method for cp_model_lm objects.

Usage

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

Arguments

object	A cp_model_lm object.
...	Additional arguments.

Value

A list summarizing the linear model.

summary.cp_model_lmer *Summary method for cp_model_lmer objects.*

Description

Summary method for cp_model_lmer objects.

Usage

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

Arguments

object	A cp_model_lmer object.
...	Additional arguments.

Value

A list summarizing the mixed-effects model.

summary.cp_model_nls *Summarize a Cross-Price Demand Model (Nonlinear)*

Description

Summarize a Cross-Price Demand Model (Nonlinear)

Usage

```
## S3 method for class 'cp_model_nls'
summary(object, inv_fun = identity, inverse_fun = deprecated(), ...)
```

Arguments

object	A cross-price model object from fit_cp_nls with return_all=TRUE.
inv_fun	Optional function to inverse-transform predictions (e.g., ll4_inv). Default is identity.
inverse_fun	[Deprecated] Use inv_fun instead.
...	Additional arguments (unused).

Value

A list containing model summary information.

summary.cp_unsystematic

Summarize Cross-Price Unsystematic Data Check Results

Description

Summarizes systematic and unsystematic patterns from multiple calls to `check_unsystematic_cp()`. This includes overall proportions, trend and bounce direction counts, and optionally summaries by subject or group.

Usage

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

Arguments

<code>object</code>	A data frame containing results from multiple <code>check_unsystematic_cp()</code> calls, with at minimum the columns <code>'delta_direction'</code> , <code>'bounce_direction'</code> , and <code>'bounce_any'</code> . Columns <code>'id'</code> , <code>'group'</code> , <code>'reversals'</code> , and <code>'returns'</code> are optional but allow extended summaries.
<code>...</code>	Additional arguments (currently unused)

Value

A list of class `summary.cp_unsystematic` with the following elements:

- total_patterns** Number of total patterns examined.
- systematic_count** Count of systematic patterns (no bounce).
- unsystematic_count** Count of unsystematic patterns (bounce detected).
- systematic_percent** Proportion of systematic patterns.
- unsystematic_percent** Proportion of unsystematic patterns.
- trend_counts** Breakdown of trend directions.
- bounce_counts** Breakdown of bounce directions.
- reversal_summary** (Optional) Summary of zero-reversal patterns, if present in input.
- return_summary** (Optional) Summary of zero-return patterns, if present in input.
- group_summary** (Optional) Summary stats by `'group'`.
- problem_ids** (Optional) Top IDs with unsystematic patterns.

systematic-wrappers	<i>Systematicity Check Wrappers</i>
---------------------	-------------------------------------

Description

Modern wrappers for systematicity checks with unified output vocabulary.

theme_apa	<i>APA Theme</i>
-----------	------------------

Description

APA theme for ggplot

Usage

```
theme_apa(plot.box = FALSE)
```

Arguments

plot.box Boolean for a box around the plot

Details

Theme for ggplot graphics that closely align with APA formatting

Value

ggplot theme

Author(s)

Brent Kaplan bkaplan.ku@gmail.com

Examples

```
p <- ggplot2::ggplot(apt, ggplot2::aes(x = x, y = y)) +  
  ggplot2::geom_point()  
p + theme_apa()
```

theme_beezdemand	<i>beezdemand Plot Theme</i>
------------------	------------------------------

Description

beezdemand Plot Theme

Usage

```
theme_beezdemand(
  style = c("modern", "apa"),
  base_size = 11,
  base_family = "sans"
)
```

Arguments

style	Character. One of "modern" or "apa".
base_size	Base font size (default: 11).
base_family	Base font family (default: "sans").

Value

A ggplot2 theme object.

Examples

```
library(ggplot2)
ggplot(iris, aes(Sepal.Length, Sepal.Width)) +
  geom_point() +
  theme_beezdemand()
```

tidy.beezdemand_comparison	<i>Tidy a demand-parameter comparison into a flat contrasts frame</i>
----------------------------	---

Description

Backend-agnostic `broom::tidy()` method for `beezdemand_comparison` objects (returned by `get_demand_comparisons()` on both the NLME and TMB backends). This flat long tibble is the cross-backend contract: identical column names and order regardless of backend. The nested object itself keeps each backend's native dialect (see `get_demand_comparisons()`).

Usage

```
## S3 method for class 'beezdemand_comparison'
tidy(x, exponentiate = FALSE, ...)
```

Arguments

x	A beezdemand_comparison object.
exponentiate	Logical. If TRUE, return base-invariant ratios (estimate = 10^{estimate} , CIs back-transformed); std.error becomes NA following broom's convention for exponentiated fits. Default FALSE.
...	Unused.

Value

A tibble with columns param, contrast, estimate, std.error, statistic, df, conf.low, conf.high, p.value. Estimates and CIs are on the log10 scale (or ratios when exponentiate = TRUE). statistic is a t ratio with finite df on the NLME backend and an asymptotic z (df = Inf) on the TMB backend (the value differs by backend, by design).

Examples

```
data(apt_full)
# 40 subjects per gender keep the example fast; use the full data in practice
ids <- unique(apt_full[c("id", "gender")])
ids <- ids[ids$gender %in% c("Male", "Female"), ]
keep <- unlist(lapply(split(ids$id, ids$gender), head, 40))
dat <- apt_full[apt_full$id %in% keep, ]
fit <- fit_demand_tmb(dat, equation = "exponential",
                     factors = "gender", verbose = 0)
res <- get_demand_comparisons(fit, param = c("Q0", "alpha"))
tidy(res)
tidy(res, exponentiate = TRUE)
```

tidy.beezdemand_fixed *Tidy Method for beezdemand_fixed*

Description

Tidy Method for beezdemand_fixed

Usage

```
## S3 method for class 'beezdemand_fixed'
tidy(x, report_space = c("natural", "log10"), ...)
```

Arguments

x	A beezdemand_fixed object
report_space	Character. Reporting space for core parameters. One of "natural" (default) or "log10".
...	Additional arguments (ignored)

Value

A tibble of model coefficients with columns: id, term, estimate, std.error, statistic, p.value, component

tidy.beezdemand_fixed_grouped

Tidy Method for beezdemand_fixed_grouped

Description

Calls tidy() on each per-group child and prepends group columns.

Usage

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

Arguments

x	A beezdemand_fixed_grouped object
...	Additional arguments passed to tidy.beezdemand_fixed()

Value

A tibble with group columns prepended

tidy.beezdemand_hurdle

Tidy a beezdemand_hurdle Model

Description

Returns a tibble of model coefficients in tidy format.

Usage

```
## S3 method for class 'beezdemand_hurdle'
tidy(x, report_space = c("natural", "log10", "internal"), ...)
```

Arguments

<code>x</code>	An object of class <code>beezdemand_hurdle</code> .
<code>report_space</code>	Character. Reporting space for core demand parameters. One of "internal", "natural", or "log10". <code>estimate/std.error</code> follow this scale; <code>statistic/p.value</code> are always on the estimation scale (transformation-invariant).
<code>...</code>	Additional arguments (currently unused).

Value

A tibble with columns:

term Parameter name

estimate Point estimate

std.error Standard error

statistic z-value

p.value P-value

component One of "zero_probability", "consumption", "variance", or "fixed"

`tidy.beezdemand_nlme` *Tidy method for beezdemand_nlme*

Description

Tidy method for `beezdemand_nlme`

Usage

```
## S3 method for class 'beezdemand_nlme'
tidy(
  x,
  effects = c("fixed", "ran_pars"),
  report_space = c("natural", "log10"),
  df_method = c("containment", "between"),
  ...
)
```

Arguments

<code>x</code>	A <code>beezdemand_nlme</code> object
<code>effects</code>	Character. Which effects to include: "fixed", "ran_pars", or both (the default).
<code>report_space</code>	Character. Reporting space for core parameters. One of "natural" or "log10" (match.arg default "natural"). <code>estimate/std.error</code> follow this scale; <code>statistic/p.value</code> are always on the estimation scale (nlme's native containment-t test, which is transformation-invariant).

df_method	Character. "containment" (default) reports nlme's own df, which uses the observation-level residual df for between-subject terms as well, so their p-values are anticonservative; "between" substitutes $n_{\text{subjects}} - \text{rank}(X_{\text{between}})$ for between-subject terms and recomputes their p-values. See summary.beezdemand_nlme() for the rule.
...	Additional arguments (ignored)

Value

A tibble of model terms with columns:

- term: Parameter name
- estimate: Point estimate. For component == "variance" rows this is a *standard deviation* (pulled from `nlme::VarCorr(model)[, "StdDev"]`), matching [tidy.beezdemand_tmb\(\)](#) and the `broom.mixed::tidy.lme` convention.
- std.error: Standard error (NA for variance components)
- statistic: t-value (NA for variance components)
- p.value: P-value (NA for variance components)
- df: Degrees of freedom the p-value uses (NA for variance components); see `df_method`
- component: "fixed" or "variance"
- estimate_scale: Scale that estimate is reported on
- term_display: Display label for term
- estimate_internal: Pre-transform estimate; present whenever effects includes "fixed"

`tidy.beezdemand_systematicity`

Tidy Method for beezdemand_systematicity

Description

Tidy Method for beezdemand_systematicity

Usage

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

Arguments

x	A beezdemand_systematicity object
...	Additional arguments (ignored)

Value

The per-subject results tibble

tidy.beezdemand_tmb *Tidy a beezdemand_tmb Model*

Description

Tidy a beezdemand_tmb Model

Usage

```
## S3 method for class 'beezdemand_tmb'
tidy(
  x,
  effects = c("fixed", "ran_pars"),
  report_space = c("natural", "log10", "internal"),
  ...
)
```

Arguments

x	A beezdemand_tmb object.
effects	Character. Which effects to return: "fixed" for the fixed-effect (core demand parameter) rows, "ran_pars" for the random-effect variance components, or both (the default). Matches the effects argument of tidy.beezdemand_nlme() .
report_space	Character. Reporting space for the fixed-effect (core demand parameter) rows. One of "natural", "log10", or "internal". Variance-component rows are unaffected (see Details). estimate/std.error follow this scale; statistic/p.value are always on the estimation scale (transformation-invariant).
...	Additional arguments.

Details

Variance-component rows (effects = "ran_pars") are exactly the rows of `summary(x)$variance_components`: the Q0 and alpha random-effect standard deviations on the **log10 scale** and the residual standard deviation on the model's likelihood scale. They are not the raw internal logsigma optimizer coefficients and do not respond to report_space; std.error is NA for them. Random-effect *correlations* are not tidied here (see `summary(x)$correlations` or `VarCorr(x)` for those). The NLME sibling [tidy.beezdemand_nlme\(\)](#) likewise reports SDs, so backend-agnostic code can consume the estimate column without dispatch logic on either side.

Value

A tibble of model terms with columns term, estimate, std.error, statistic, p.value, df (Inf on fixed-effect rows: the Wald test is an asymptotic z, i.e. a t on infinite df; NA on variance rows), component, estimate_scale, and term_display. An estimate_internal column (the pre-transform estimate) is additionally present whenever effects includes "fixed". Fixed-effect rows carry component == "fixed" (matching [tidy.beezdemand_nlme\(\)](#) and the nlme/lme4

convention); variance-component rows carry `component == "variance"`. A `hessian_warning` attribute (character scalar, or absent) is attached depending on `x$hessian_pd`: absent (no attribute) when `hessian_pd` is `TRUE` or `NULL` (the field is missing on a legacy fit object, so there is nothing to report); a message noting the Hessian is not positive definite when `hessian_pd` is `FALSE`; a message noting positive-definiteness is unknown (because `TMB::sdreport()` failed entirely, so SEs/CIs are unavailable, not merely unreliable) when `hessian_pd` is `NA`. This attribute is not printed by an ordinary tibble print (see `summary.beezdemand_tmb()` or `check_demand_model()` for the surfaced versions of the same diagnostic).

Examples

```
data(apt)
fit <- fit_demand_tmb(apt, equation = "exponential", verbose = 0)
tidy(fit)
tidy(fit, effects = "fixed", report_space = "log10")
tidy(fit, effects = "ran_pars")
```

tidy.cp_model_lm

Extract coefficients from a linear cross-price model in tidy format

Description

Extract coefficients from a linear cross-price model in tidy format

Usage

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

Arguments

<code>x</code>	A <code>cp_model_lm</code> object.
<code>...</code>	Additional arguments (unused).

Value

A tibble with columns: `term`, `estimate`, `std.error`, `statistic`, `p.value`.

tidy.cp_model_lmer	<i>Extract coefficients from a mixed-effects cross-price model in tidy format</i>
--------------------	---

Description

Extract coefficients from a mixed-effects cross-price model in tidy format

Usage

```
## S3 method for class 'cp_model_lmer'
tidy(x, effects = c("fixed", "ran_vals", "ran_pars", "random"), ...)
```

Arguments

x	A cp_model_lmer object.
effects	Which effects to return: "fixed" (default), "ran_vals" (conditional modes of the random effects), or "ran_pars" (random-effect standard deviations and correlations). "random" is accepted as an alias for "ran_vals".
...	Additional arguments passed to broom.mixed::tidy.

Value

A tibble with tidy coefficient information. When the model is NULL (the fit failed) a zero-row tibble with the same columns as the corresponding successful call is returned.

tidy.cp_model_nls	<i>Convert a cross-price model to a tidy data frame of coefficients</i>
-------------------	---

Description

Extracts model coefficients from a cross-price demand model into a tidy data frame, following the conventions of the broom package. When model fitting failed, an empty data frame with the expected structure is returned.

Usage

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

Arguments

x	A model object from fit_cp_nls or fit_cp_linear
...	Additional arguments (unused)

Value

A data frame with one row per coefficient, containing columns:

term	The name of the model parameter
estimate	The estimated coefficient value
std.error	The standard error of the coefficient
statistic	The t-statistic for the coefficient
p.value	The p-value for the coefficient

Examples

```
data(etm)
fit <- fit_cp_nls(etm, equation = "exponentiated")
tidy(fit)
```

update.beezdemand_tmb *Update a beezdemand_tmb fit*

Description

Re-fits with named arguments substituted into the original call. Pass any argument of `fit_demand_tmb()` (e.g., `factors = NULL`, `random_effects = ~ 1`, `equation = "simplified"`). Does NOT support formula-update syntax (`.` - term) because `fit_demand_tmb()` is argument-driven, not formula-driven.

Usage

```
## S3 method for class 'beezdemand_tmb'
update(object, ..., evaluate = TRUE)
```

Arguments

object	A beezdemand_tmb object.
...	Named arguments to substitute into the original <code>fit_demand_tmb()</code> call.
evaluate	If TRUE (default), re-evaluate the updated call and return the new fit. If FALSE, return the unevaluated call.

Value

A new beezdemand_tmb object, or an unevaluated call.

See Also

[fit_demand_tmb\(\)](#), [formula.beezdemand_tmb\(\)](#).

Examples

```
data(apt)
fit <- fit_demand_tmb(apt, equation = "exponential", verbose = 0)
update(fit, equation = "simplified", evaluate = FALSE)
```

VarCorr.beezdemand_tmb

Random-Effect Variance Components for a TMB Demand Model

Description

Extracts the random-effect variance components from a beezdemand_tmb fit in the matrix layout produced by nlme::VarCorr(), so users familiar with **nlme** or **lme4** can introspect a TMB fit with the accessor they already know. The reported values are the same ones returned by [summary.beezdemand_tmb](#): the Q0 and alpha random-effect standard deviations on the **log10 scale** and the residual standard deviation on the model's likelihood scale. This is a presentation shim that formats already-computed values and recomputes nothing.

Usage

```
## S3 method for class 'beezdemand_tmb'
VarCorr(x, sigma = 1, rdig = 3, ...)
```

Arguments

x	A beezdemand_tmb object.
sigma	Present for signature compatibility with nlme::VarCorr(). The TMB summary reports variance components as absolute standard deviations, so there is no residual scale factor to apply; any value other than the default (1) is an error.
rdig	Integer. Number of significant digits used when formatting the displayed values. Default 3.
...	Unused; present for generic compatibility.

Value

A character matrix of class "VarCorr.lme" with one row per random-effect term plus a final "Residual" row, columns "Variance" and "StdDev", and, for fits with correlated random effects (pdSymm), a "Corr" column. print() dispatches to nlme's print.VarCorr.lme().

Note

The Corr column is placed using nlme's convention, with each correlation on the row of its higher-indexed random effect. For multi-block pdBlocked fits the correlations are placed on the correct global rows (each correlated block's off-diagonals are offset by the cumulative random-effect dimension of the earlier blocks); summary(x)\$correlations remains available for the labelled values.

See Also

[VarCorr](#), [summary.beezdemand_tmb](#)

Examples

```
data(apt)
fit <- fit_demand_tmb(apt, equation = "exponential", verbose = 0)
VarCorr(fit)
```

vcov.beezdemand_hurdle

Variance-covariance matrix for a beezdemand_hurdle fit

Description

Returns the joint fixed-effect VCOV across Part I (logit-link participation) and Part II (log-link demand) components, with row/col names prefixed by component (zero_probability.<term>, consumption.<term>, variance.<term>). Same component classification as `confint.beezdemand_hurdle()`.

Usage

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

Arguments

object	A beezdemand_hurdle object.
...	Unused.

Value

Numeric symmetric matrix with component-prefixed dim names.

See Also

[coef.beezdemand_hurdle\(\)](#), [confint.beezdemand_hurdle\(\)](#).

vcov.beezdemand_tmb	<i>Variance-covariance matrix for a beezdemand_tmb fit</i>
---------------------	--

Description

Returns the fixed-effect VCOV from the TMB sdreport, i.e., the inverse of the negative Hessian at the MLE after Laplace-marginalizing the random effects. Row/column names follow the optimizer's internal parameterization (matching `names(coef(object, type = "internal"))`).

Usage

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

Arguments

<code>object</code>	A beezdemand_tmb object.
<code>...</code>	Unused.

Value

Numeric symmetric matrix of dimension $p \times p$.

See Also

[coef.beezdemand_tmb\(\)](#), [confint.beezdemand_tmb\(\)](#).

Examples

```
data(apt)  
fit <- fit_demand_tmb(apt, equation = "exponential", verbose = 0)  
V <- vcov(fit)  
isSymmetric(V)
```

Index

- * **datasets**
 - apt, [11](#)
 - apt_full, [12](#)
 - cannabisCigarettes, [30](#)
 - cp, [56](#)
 - etm, [58](#)
 - ko, [125](#)
 - lowNicClean, [129](#)
 - ongoingETM, [133](#)
- * **demand-fitting**
 - fit_demand_fixed, [68](#)
 - fit_demand_hurdle, [71](#)
 - fit_demand_mixed, [74](#)
 - fit_demand_tmb, [77](#)
- * **power-analysis**
 - find_n_demand, [60](#)
 - power_demand, [164](#)
- AIC.beezdemand_hurdle, [7](#)
- annotation_logticks2, [7](#)
- anova.beezdemand_hurdle, [8](#)
- anova.beezdemand_nlme, [9](#)
- anova.beezdemand_nlme(), [11](#)
- anova.beezdemand_tmb, [10](#)
- apt, [11](#)
- apt_full, [12](#)
- augment.beezdemand_fixed, [12](#)
- augment.beezdemand_hurdle, [13](#)
- augment.beezdemand_hurdle(), [87](#)
- augment.beezdemand_nlme, [15](#)
- augment.beezdemand_tmb, [16](#)
- augment.beezdemand_tmb(), [88](#), [198](#)
- augment.cp_model_lm, [17](#)
- augment.cp_model_lmer, [17](#)
- augment.cp_model_nls, [18](#)
- beezdemand_descriptive_methods, [18](#)
- beezdemand_empirical_methods, [20](#)
- BIC.beezdemand_hurdle, [22](#)
- boot_demand, [22](#)
- broom::tidy(), [216](#)
- calc_group_metrics, [22](#), [23](#), [24](#), [24](#), [28](#)
- calc_group_metrics.beezdemand_tmb, [24](#), [25](#)
- calc_observed_pmax_omax, [26](#)
- calc_omax_pmax, [25](#), [27](#)
- calculate_amplitude_persistence, [28](#)
- cannabisCigarettes, [30](#)
- ChangeData, [31](#)
- check_demand_model, [32](#)
- check_demand_model(), [222](#)
- check_systematic_cp, [34](#)
- check_systematic_demand, [35](#)
- check_unsystematic_cp, [37](#), [67](#)
- CheckCols, [39](#)
- CheckUnsystematic, [40](#)
- coef-methods, [41](#)
- coef.beezdemand_fixed, [42](#)
- coef.beezdemand_fixed_grouped, [42](#)
- coef.beezdemand_hurdle, [43](#)
- coef.beezdemand_hurdle(), [226](#)
- coef.beezdemand_nlme, [43](#), [88](#), [105](#), [193](#)
- coef.beezdemand_tmb, [44](#)
- coef.beezdemand_tmb(), [227](#)
- coef.cp_model_lm (coef-methods), [41](#)
- coef.cp_model_lmer (coef-methods), [41](#)
- coef.cp_model_nls (coef-methods), [41](#)
- compare_hurdle_models, [45](#), [72](#), [73](#)
- compare_hurdle_models(), [48](#)
- compare_models, [46](#)
- confint.beezdemand_fixed, [48](#)
- confint.beezdemand_hurdle, [49](#)
- confint.beezdemand_hurdle(), [226](#)
- confint.beezdemand_nlme, [50](#)
- confint.beezdemand_nlme(), [53](#)
- confint.beezdemand_tmb, [24](#), [51](#)
- confint.beezdemand_tmb(), [11](#), [227](#)
- confint.cp_model_lm, [53](#)
- confint.cp_model_lmer, [54](#)

- confint.cp_model_nls, 54
- cp, 56
- cp_posthoc_intercepts, 56
- cp_posthoc_slopes, 57
- etm, 58
- extract_coefficients, 58
- ExtraF, 59
- find_n_demand, 60, 167
- find_n_demand(), 167
- fit_cp_linear, 63
- fit_cp_nls, 65
- fit_cp_nls(), 55
- fit_demand_fixed, 68, 74, 77, 82
- fit_demand_fixed(), 49, 74, 77, 79, 82, 84, 86
- fit_demand_hurdle, 25, 28, 46, 70, 71, 77, 82, 104, 110, 200, 205, 208
- fit_demand_hurdle(), 50, 70, 77, 82
- fit_demand_mixed, 70, 74, 74, 82, 105
- fit_demand_mixed(), 51, 70, 74, 82, 96
- fit_demand_tmb, 24, 26, 70, 74, 77, 77
- fit_demand_tmb(), 61, 62, 70, 74, 77, 164–167, 224
- FitCurves, 83
- FitCurves(), 107
- FitMeanCurves, 85
- fitted.beezdemand_hurdle, 86
- fitted.beezdemand_hurdle(), 197
- fitted.beezdemand_tmb, 87
- fitted.beezdemand_tmb(), 198
- fixed-demand, 88
- fixef.beezdemand_nlme, 44, 88, 193
- fixef.beezdemand_tmb, 89
- fixef.cp_model_lmer, 89
- formula.beezdemand_hurdle, 90
- formula.beezdemand_tmb, 90
- formula.beezdemand_tmb(), 90, 130, 224
- get_demand_comparisons, 91
- get_demand_comparisons(), 77, 216
- get_demand_comparisons.beezdemand_tmb, 25, 93
- get_demand_comparisons.beezdemand_tmb(), 92
- get_demand_param_emms, 95, 105, 108
- get_demand_param_emms(), 77, 157
- get_demand_param_emms.beezdemand_tmb, 25, 26, 97
- get_demand_param_emms.beezdemand_tmb(), 94
- get_demand_param_trends, 98
- get_descriptive_summary, 99
- get_descriptive_summary(), 18, 19, 114
- get_empirical_measures, 101
- get_empirical_measures(), 20, 21, 115
- get_hurdle_param_summary, 103
- get_individual_coefficients, 104
- get_k, 106
- get_k(), 116
- get_observed_demand_param_emms, 107
- get_subject_pars, 45, 104, 109
- get_subject_pars.beezdemand_nlme, 110
- get_subject_pars.beezdemand_tmb, 109, 110, 111, 111
- GetAnalyticPmax, 112
- GetAnalyticPmaxFallback, 113
- GetDescriptives, 114
- GetDescriptives(), 99, 100
- GetEmpirical, 115
- GetEmpirical(), 101, 102
- GetK, 116
- GetK(), 106, 107
- GetSharedK, 117
- GetValsForSim, 118
- glance.beezdemand_fixed, 118
- glance.beezdemand_fixed_grouped, 119
- glance.beezdemand_hurdle, 120
- glance.beezdemand_nlme, 120
- glance.beezdemand_nlme(), 122
- glance.beezdemand_systematicity, 121
- glance.beezdemand_tmb, 122
- glance.beezdemand_tmb(), 121
- glance.cp_model_lm, 123
- glance.cp_model_lmer, 123
- glance.cp_model_nls, 124
- identity, 150
- ko, 125
- lambertW, 125
- ll4, 126, 192, 202
- ll4_inv, 127, 150, 192, 202
- lme4::confint.merMod(), 54
- logLik.beezdemand_hurdle, 128

- lowNicClean, 129
- model.matrix.beezdemand_hurdle, 129
- model.matrix.beezdemand_hurdle(), 90
- model.matrix.beezdemand_tmb, 130
- model.matrix.beezdemand_tmb(), 91, 129
- nlme::VarCorr(), 33
- nobs.beezdemand_hurdle, 131
- nobs.cp_model_lm, 131
- nobs.cp_model_lmer, 132
- nobs.cp_model_nls, 132
- ongoingETM, 133
- palette_beezdemand, 133
- pivot_demand_data, 134
- plot-theme, 136
- plot.beezdemand_descriptive
 - (beezdemand_descriptive_methods), 18
- plot.beezdemand_descriptive(), 100
- plot.beezdemand_empirical
 - (beezdemand_empirical_methods), 20
- plot.beezdemand_empirical(), 102
- plot.beezdemand_fixed, 136
- plot.beezdemand_fixed_grouped, 138
- plot.beezdemand_hurdle, 73, 138
- plot.beezdemand_nlme, 141
- plot.beezdemand_tmb, 143
- plot.cp_model_lm, 145
- plot.cp_model_lmer, 146
- plot.cp_model_nls, 147
- plot_alpha_distribution, 148
- plot_demand_overlay, 149
- plot_elasticity, 151
- plot_expenditure, 152
- plot_loss_profile, 154
- plot_loss_profile(), 157
- plot_loss_surface, 155
- plot_model_comparison, 157
- plot_qq, 158
- plot_qq(), 33, 171
- plot_re_diagnostics, 159
- plot_residuals, 160
- plot_residuals(), 33
- plot_subject, 161
- PlotCurve, 162
- PlotCurves, 163
- power_demand, 62, 164
- power_demand(), 60–62
- predict.beezdemand_fixed, 168
- predict.beezdemand_hurdle, 73, 169
- predict.beezdemand_hurdle(), 87, 140, 175
- predict.beezdemand_nlme, 172
- predict.beezdemand_nlme(), 174, 175
- predict.beezdemand_tmb, 45, 174
- predict.beezdemand_tmb(), 87, 88, 198
- predict.cp_model_lm, 176
- predict.cp_model_lmer, 147, 177
- predict.cp_model_nls, 178
- predict.nlme, 173
- print.anova.beezdemand_hurdle, 178
- print.beezdemand_comparison, 179
- print.beezdemand_descriptive
 - (beezdemand_descriptive_methods), 18
- print.beezdemand_diagnostic_plots, 179
- print.beezdemand_diagnostics, 180
- print.beezdemand_empirical
 - (beezdemand_empirical_methods), 20
- print.beezdemand_fixed, 180
- print.beezdemand_fixed_grouped, 181
- print.beezdemand_hurdle, 181
- print.beezdemand_model_comparison, 182
- print.beezdemand_nlme, 182
- print.beezdemand_summary, 183
- print.beezdemand_systematicity, 184
- print.beezdemand_tmb, 184
- print.cp_model_lm, 185
- print.cp_model_lmer, 185
- print.cp_model_nls, 186
- print.cp_posthoc, 186
- print.summary.beezdemand_fixed, 187
- print.summary.beezdemand_hurdle, 187
- print.summary.beezdemand_nlme, 188
- print.summary.beezdemand_systematicity, 188
- print.summary.beezdemand_tmb, 189
- print.summary.cp_model_lm, 189
- print.summary.cp_model_lmer, 190
- print.summary.cp_model_nls, 190
- print.summary.cp_unsystematic, 191
- print_mc_summary, 191

pseudo_ll4_trans, 192
 ranef.beezdemand_nlme, 44, 88, 193
 ranef.beezdemand_tmb, 194
 ranef.cp_model_lmer, 195
 RecodeOutliers, 195
 ReplaceZeros, 196
 residuals.beezdemand_hurdle, 197
 residuals.beezdemand_tmb, 198
 residuals.beezdemand_tmb(), 88
 run_hurdle_monte_carlo, 191, 199, 205

 scale_color_beezdemand, 200
 scale_fill_beezdemand, 201
 scale_ll4, 202
 scales::transform_log10(), 19
 simulate_hurdle_data, 73, 199, 200, 203
 SimulateDemand, 206
 stats::confint.default(), 53
 summary(), 52
 summary.beezdemand_descriptive
 (beezdemand_descriptive_methods),
 18
 summary.beezdemand_descriptive(), 100
 summary.beezdemand_empirical
 (beezdemand_empirical_methods),
 20
 summary.beezdemand_empirical(), 102
 summary.beezdemand_fixed, 207
 summary.beezdemand_fixed_grouped, 208
 summary.beezdemand_hurdle, 73, 208
 summary.beezdemand_nlme, 209
 summary.beezdemand_nlme(), 220
 summary.beezdemand_systematicity, 211
 summary.beezdemand_tmb, 211, 225, 226
 summary.beezdemand_tmb(), 222
 summary.cp_model_lm, 212
 summary.cp_model_lmer, 213
 summary.cp_model_nls, 213
 summary.cp_unsystematic, 214
 systematic-wrappers, 215

 theme_apa, 215
 theme_apa(), 19
 theme_beezdemand, 216
 tidy(), 93
 tidy.beezdemand_comparison, 216
 tidy.beezdemand_comparison(), 93, 95
 tidy.beezdemand_fixed, 217
 tidy.beezdemand_fixed_grouped, 218
 tidy.beezdemand_hurdle, 218
 tidy.beezdemand_nlme, 219
 tidy.beezdemand_nlme(), 221
 tidy.beezdemand_systematicity, 220
 tidy.beezdemand_tmb, 221
 tidy.beezdemand_tmb(), 220
 tidy.cp_model_lm, 222
 tidy.cp_model_lmer, 223
 tidy.cp_model_nls, 223
 tidyr::pivot_longer(), 134
 tidyr::pivot_wider(), 134
 trans_new, 192, 202

 update.beezdemand_tmb, 224
 update.beezdemand_tmb(), 91

 validate_cp_data, 67
 VarCorr, 226
 VarCorr.beezdemand_tmb, 225
 vcov.beezdemand_hurdle, 226
 vcov.beezdemand_tmb, 227
 vcov.beezdemand_tmb(), 53