Package 'asympDiag'

May 7, 2026

Type Package

Title Diagnostic Tools for Asymptotic Theory

Version 0.3.2

Description Leveraging Monte Carlo simulations, this package provides tools for diagnosing regression models. It implements a parametric bootstrap framework to compute statistics, generates diagnostic envelopes to assess goodness-of-fit, and evaluates type I error control for Wald tests. By simulating data under the assumption that the model is true, it helps to identify model mis-specifications and enhances the reliability of the model inferences.

License MIT + file LICENSE

URL https://github.com/Alvaro-Kothe/asympDiag

BugReports https://github.com/Alvaro-Kothe/asympDiag/issues

Imports cli

Suggests glmmTMB, lme4, MASS, Matrix, rlang, sandwich, survival, testthat (>= 3.0.0), vctrs, withr

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.3

NeedsCompilation no

Author Álvaro Kothe [aut, cre, cph] (ORCID: <https://orcid.org/0000-0003-2114-3193>), Alexandre Patriota [aut]

Maintainer Álvaro Kothe <kothe65@gmail.com>

Repository CRAN

Date/Publication 2026-02-15 19:00:02 UTC

Contents

concat_pvalues . . . 2
envelope . . . 3
envelope_residual . . . 5
get_converged . . . 6
get_fixef . . . 6
get_model_response . . . 7
get_refit . . . 8
get_vcov . . . 8
parametric_bootstrap . . . 9
plot.AD_envelope . . . 11
plot.AD_pvalues . . . 12
plot_cook . . . 13
plot_ecdf_pvalue . . . 14
plot_res_vs_linear_predictor . . . 15
refit_model . . . 16
select_covariates . . . 17
simulate_wald_pvalues . . . 19

Index **22**

`concat_pvalues` *Concatenate AD_pvalues object*

Description

Concatenate AD_pvalues object

Usage

```
concat_pvalues(ld_pvalues)
```

Arguments

`ld_pvalues`	list of elements of class `AD_pvalues`.

Value

Object of class `AD_pvalues`

envelope *Generate Simulated Envelope*

Description

Generates QQ-plot with simulated envelope residuals.

Usage

```
envelope(
  model,
  residual_fn = envelope_residual(model),
  alpha = 0.05,
  nsim = 100,
  responses = NULL,
  no_warnings = FALSE,
  no_messages = FALSE,
  converged_only = FALSE,
  show_progress = TRUE,
  plot.it = TRUE,
  refit_fn = NULL,
  ...
)
```

Arguments

model	A model to generate responses and compute the observed residuals.
residual_fn	A function to calculate model residuals. The default is envelope_residual() for an absolute residual.
alpha	The significance level for constructing the envelope bounds. Defaults to 0.05.
nsim	The number of simulations to perform for envelope construction. Defaults to 100.
responses	An optional list of values to be used as response variables to refit the model.
no_warnings	If TRUE, ignore simulations that threw warnings.
no_messages	If TRUE, ignore simulations that shown messages.
converged_only	Use p-values from converged models only.
show_progress	Display a progress bar for the simulation iteration.
plot.it	Logical. Generate envelope plot.
refit_fn	Function to refit the model with new responses. If NULL, defaults to get_refit(model, y, ...).
...	Extra arguments to get_refit()

Details

Simulates new responses using `stats::simulate()` and refits the model for each vector of new responses using `get_refit()`. The function then computes residuals for each simulation, sorts them, and constructs envelope bands and a median line based on the quantiles of these residuals.

`refit_fn` is a function that supposedly compute the refit of `model`. Use this method if the default `get_refit()` doesn't work. If `refit_fn` is `NULL`, it's value is defined as `function(y, ...) get_refit(model, y, ...)`.

Value

An object of class `AD_envelope`, which contains the following components:

observed A vector of observed quantiles from the model residuals.

outside A logical vector indicating whether each observation falls outside the constructed envelope bounds.

lower The lower bounds of the envelope for each observation.

med The median bounds of the envelope for each observation.

upper The upper bounds of the envelope for each observation.

See Also

`get_refit`, `simulate`, `rstudent`, `plot.AD_envelope`, `parametric_bootstrap()`

Examples

```
fit <- lm(mpg ~ cyl, data = mtcars)

envelope(fit)

# Use pearson residuals, and plot it agains the expected normal quantiles.
env_measures <- envelope(fit,
  residual_fn = function(x) residuals.lm(x, type = "pearson"), plot.it = FALSE
)
plot(env_measures, distribution = stats::qnorm, colors = c("gray", "black"))

## Using custom refit_fn
if (require("survival")) {
  fit <- survreg(Surv(futime, fustat) ~ ecog.ps + rx, ovarian,
    dist = "exponential"
  )
  fitted_rate <- 1 / fitted(fit)
 new_responses <- replicate(100, rexp(length(fitted_rate), fitted_rate), simplify = FALSE)
  refit_surv_ovarian <- function(.y) {
    survreg(Surv(.y, fustat) ~ ecog.ps + rx, ovarian, dist = "exponential")
  }
  env_measures <- envelope(fit,
    responses = new_responses,
    residual_fn = function(x) abs(residuals(x, type = "deviance")),
    refit_fn = refit_surv_ovarian, plot.it = FALSE
```

```
  )
  # Absolute residuals are best shown with halfnormal quantiles
  plot(env_measures, distribution = function(p) qnorm((1 + p) / 2))
}
```

`envelope_residual` *Recommended Residuals for Envelope Plots*

Description

This function returns a function that computes residuals for envelope plots. These residuals are typically absolute values to be compared against the half-normal distribution.

Usage

```
envelope_residual(object, ...)

## Default S3 method:
envelope_residual(object, ...)

## S3 method for class 'glm'
envelope_residual(object, ...)

## S3 method for class 'lm'
envelope_residual(object, ...)
```

Arguments

`object`	An object for which model residuals can be extracted.
`...`	Additional arguments passed to the residual function.

Details

For objects of class `glm`, the default residuals are:

- Deviance residuals, except for `poisson` and `binomial` families.
- For `poisson` and `binomial` families it uses deletion residual adapted from `rstudent()`.

For objects of class `lm`, the default residuals is `rstudent()`.

For other classes, the default is `stats::residuals()`, meaning no specialized recommendation is currently provided.

Value

A function that computes residuals from an object

get_converged *Did the model converged*

Description

Retrieves if the model converged.

Usage

```
get_converged(object)

## Default S3 method:
get_converged(object)
```

Arguments

object	A model to check for convergence.

Details

The default method retrieves `object$converged`. For models of class merMod it verifies if the infinity norm of the Newton–Raphson step is less than 0.0001.

Value

A boolean value.

get_fixef *Get fixed effects*

Description

Extracts the fixed effects coefficients from a model.

Usage

```
get_fixef(object)

## Default S3 method:
get_fixef(object)

## S3 method for class 'merMod'
get_fixef(object)

## S3 method for class 'glmmTMB'
get_fixef(object)
```

Arguments

object	A model object for which fixed effects are to be retrieved.

Details

By default it calls `stats::coef()`. If the model class is `merMod` calls `fixef`.

Value

A numeric vector of fixed effects coefficients.

---

get_model_response *Extract the Response Variable from a Model Object*

Description

This function is designed to extract the response variable from a fitted model object.

Usage

```
get_model_response(object)
```

Arguments

object	A fitted model from which the response variable should be extracted.

Details

The default method attempts to create a model frame using `model.frame()`. Any error encountered during this process is caught and results in a `NULL` return value.

Value

The response variable extracted from the model. If it couldn't be extracted, the function returns `NULL`.

Examples

```
lm_model <- lm(mpg ~ wt + cyl, data = mtcars)
response <- get_model_response(lm_model)
print(response)
```

get_refit *Refit the Model*

Description

Refit an existing regression model using a new response variable.

Usage

```
get_refit(object, newresp, ...)
```

Arguments

object	A model.
newresp	the new response, may be a vector or a matrix.
...	other arguments passed to refit or update.

Details

This generic method refit the object with the newresp. It shold maintain the object properties by default, but some aspects of the default fitting method may be overwritten with the ....

Value

An updated model object, refitted using the new response variable.

See Also

stats::update(), lme4::refit()

get_vcov *Get covariance matrix*

Description

Retrieves the covariance matrix for the fixed effects.

Usage

```
get_vcov(object)

## Default S3 method:
get_vcov(object)

## S3 method for class 'glmmTMB'
get_vcov(object)
```

Arguments

`object`	A model object for which the covariance matrix is to be retrieved.

Details

By default it calls `stats::vcov()` to retrieve the covariances.

Value

A covariance matrix of the model object.

`parametric_bootstrap` *Perform Parametric Bootstrap Simulations*

Description

This function performs parametric bootstrap simulations by generating new response values based on the fitted model, refitting the model on these new responses, and computing a user-defined statistic for each simulated model.

Usage

```
parametric_bootstrap(
  model,
  statistic,
  nsim,
  responses = NULL,
  refit_fn = NULL,
  show_progress = TRUE,
  simplify = TRUE,
  stat_hc = NULL,
  show_message_count = TRUE,
  show_warning_count = TRUE,
  show_not_converged_count = TRUE,
  ...
)
```

Arguments

`model`	A fitted model object that will be used to simulate responses.
`statistic`	A function that computes the desired statistic from the refitted model. It must take the refitted model as an argument.
`nsim`	The number of simulations to perform.
`responses`	An optional list of values to be used as response variables to refit the model.
`refit_fn`	Function to refit the model with new responses. If `NULL`, defaults to `get_refit(model, y, ...)`.

show_progress Display a progress bar for the simulation iteration.

simplify logical or character string; should the result be simplified to a vector, matrix or higher dimensional array if possible? If occurs any errors during the refit procedure, the results will be a list, regardless of the value of this argument.

stat_hc A function that verifies if the computed statistic is correct. It should return nothing, just throw errors to halt execution.

show_message_count
Show total of captured messages from refit_fn as a message. It only shows if the number of messages is greater than 0.

show_warning_count
Show total of captured warnings from refit_fn as a warning. It only shows if the number of warnings is greater than 0.

show_not_converged_count
Show total of models that didn't converge as a warning. It only shows if the number of models that didn't converged is greater than 0.

... Additional arguments to be passed to refit_fn.

Details

This function implements a parametric bootstrap procedure. It generates new response values from the fitted model, refits the model for each simulated response, and computes a user-defined statistic on the refitted model. The refit function can be customized through the refit_fn argument.

If show_progress is TRUE, the progress of the simulations will be displayed using a progress bar from the cli package.

Value

A list containing the following elements:

result A list of length nsim if simplify is FALSE. Otherwise an atomic vector or matrix or list of length nsim.

responses The list of simulated response values.

simulation_warning A logical vector indicating whether a warning occurred during model refitting for each simulation.

converged A logical vector indicating whether the model refit converged for each simulation.

Examples

```
model <- lm(mpg ~ wt + hp, data = mtcars)
statistic <- function(model) coef(model)["wt"]
bootstrap_results <- parametric_bootstrap(model, statistic, nsim = 100)
wt_coefs <- Reduce(c, bootstrap_results$result)
summary(wt_coefs)
```

`plot.AD_envelope` *Envelope Plot*

Description

Plot AD_envelope

Usage

```
## S3 method for class 'AD_envelope'
plot(
  x,
  colors = getOption("asympDiag.plot.AD_envelope.colors"),
  xlab = "Expected quantiles",
  ylab = "Observed quantiles",
  distribution = function(p) stats::qnorm((1 + p)/2),
 ylim = base::range(c(x$observed, x$lower, x$upper), na.rm = TRUE, finite = TRUE),
  ...
)
```

Arguments

`x`	AD_envelope object, usually the result of `envelope()`
`colors`	Vector of length 2, with color for points outside and inside the envelope band, respectively.
`xlab`	The label for the x-axis.
`ylab`	The label for the y-axis.
`distribution`	quantile function for reference theoretical distribution.
`ylim`	the y limits of the plot.
`...`	extra arguments passed to graphics::plot

Details

Create envelope plot. The expected quantile, by default, is from a half-normal distribution, as the default residuals from `envelope_residual()` are absolute. You may replace it with the `distribution` argument.

Value

No return value, called for side effects

plot.AD_pvalues *Plot Empirical Cumulative Distribution Function (ECDF) of p-values*

Description

This function creates several plots with the empirical cumulative distribution of the p-values obtained through simulation.

Usage

```
## S3 method for class 'AD_pvalues'
plot(
  x,
  which = seq_len(length(x$test_coefficients) + 1),
 caption = as.list(paste("ECDF of", c(names(x$test_coefficients), "all coefficients"))),
  ks_test = TRUE,
  signif = c(0.01, 0.05, 0.1),
  discrepancy_tol = 0.1,
  plot_uniform = TRUE,
  uniform_legend = TRUE,
  converged_only = FALSE,
  no_warnings = FALSE,
  no_messages = FALSE,
  ylab = "Empirical cumulative distribution",
  xlab = "p-value",
  ...,
 ask = prod(graphics::par("mfcol")) < length(which) && grDevices::dev.interactive()
)
```

Arguments

x	AD_pvalues object, usually the result of simulate_wald_pvalues()
which	A vector specifying the indices of coefficients to plot. If index is bigger than the number of coefficients it plots the joint p_value.
caption	A character vector or a list with caption for each plot. If it's a list, the list index must match the coefficient index used by which. If it's a vector, it's values are used in order.
ks_test	If TRUE inserts Kolmogorov-Smirnov p-value in the graphic.
signif	Points to verify discrepancy.
discrepancy_tol	Threshold to consider point discrepant.
plot_uniform	Logical. If TRUE, plot uniform distribution.
uniform_legend	Logical. If TRUE, a legend is added to the plot to distinguish between the p-value and U(0, 1) curves. Defaults to TRUE.
converged_only	Use p-values from converged models only.

no_warnings	If TRUE, ignore simulations that threw warnings.
no_messages	If TRUE, ignore simulations that shown messages.
ylab	The label for the y-axis. Defaults to "Empirical cumulative distribution".
xlab	The label for the x-axis. Defaults to "p-value".
...	extra arguments passed to graphics::plot
ask	Logical. If TRUE, the user is prompted before each plot. Defaults to TRUE if in an interactive session and the number of plots is greater than the available space; otherwise, FALSE.

Details

If the asymptotic approximation is valid the distribution of the p-values should be close to an uniform distribution. Discrepancies are highlighted, by default it verifies the significance on the most commonly used significance values are 0.01, 0.05 and 0.10.

The reported KS (Kolmogorov-Smirnov) test is the result of the "two-sided" `stats::ks.test()` function comparing the observed p-values distribution with the uniform. The test may reject the KS test due to few simulations, make sure that the lines shown in the plot are smooth before drawing any conclusions.

Value

A vector of joint p-values for all coefficients.

Examples

```
model <- lm(mpg ~ wt + hp, data = mtcars)
p_values_ld <- simulate_wald_pvalues(model, n_sim = 100)
plot(p_values_ld)
```

---

plot_cook *Plot Cook's distances*

Description

Plot Cook's distances

Usage

```
plot_cook(
  model,
  n_highlights = 0,
  cut = FALSE,
  xlab = "Index",
  ylab = "Cook's distance",
  ...
)
```

Arguments

model	Model with cooks.distance() method
n_highlights	The number of observations with the highest Cook's distance to highlight on the plot. Defaults to 0 (no highlights).
cut	Logical. If TRUE, adds a cutoff line at the mean plus four times the standard deviation of Cook's distance. Defaults to FALSE.
xlab	The label for the x-axis. Defaults to "Index".
ylab	The label for the y-axis. Defaults to "Cook's distance".
...	Further arguments for graphics::plot()

Value

An invisible object representing Cook's distance values.

See Also

cooks.distance, plot

Examples

```
fit <- lm(mpg ~ cyl, data = mtcars)

plot_cook(fit)
plot_cook(fit, n_highlights = 2)
```

---

plot_ecdf_pvalue *Plot Empirical Cumulative Distribution Function (ECDF) of p-values*

Description

Plot Empirical Cumulative Distribution Function (ECDF) of p-values

Usage

```
plot_ecdf_pvalue(
  p_values,
  ks_test = TRUE,
  signif = c(0.01, 0.05, 0.1),
  discrepancy_tol = 0.1,
  plot_uniform = TRUE,
  uniform_legend = TRUE,
  main = "",
  ylab = "Empirical cumulative distribution",
  xlab = "p-value",
  ...
)
```

Arguments

p_values	vector of p-values
ks_test	If TRUE inserts Kolmogorov-Smirnov p-value in the graphic.
signif	Points to verify discrepancy.
discrepancy_tol	Threshold to consider point discrepant.
plot_uniform	Logical. If TRUE, plot uniform distribution.
uniform_legend	Logical. If TRUE, a legend is added to the plot to distinguish between the p-value and U(0, 1) curves. Defaults to TRUE.
main	main caption passed to plot
ylab	The label for the y-axis. Defaults to "Empirical cumulative distribution".
xlab	The label for the x-axis. Defaults to "p-value".
...	extra arguments passed to graphics::plot

Value

No return value, called for side effects

---

plot_res_vs_linear_predictor

Plot Residuals against Linear Predictor

Description

Plot Residuals against Linear Predictor

Usage

```
plot_res_vs_linear_predictor(
  model,
  residual_fn = stats::rstandard,
  xlab = "Linear Predictor",
  ylab = "Standardized deviance residuals",
  ...
)
```

Arguments

model	Model with methods predict() or fitted()
residual_fn	A function to calculate model residuals. The default is stats::rstandard.
xlab	The label for the x-axis. Defaults to "Linear Predictor".
ylab	The label for the y-axis. Defaults to "Standardized deviance residuals".
...	Extra arguments to residual_fn and plot().

Details

If the model was fitted using the `glm()` function, it will use the `predict()` method with `type = link`, otherwise, it will use the `fitted()` method.

Value

An invisible list containing the linear predictor (x) and standardized deviance residuals (y).

Examples

```
fit <- lm(mpg ~ cyl, data = mtcars)

plot_res_vs_linear_predictor(fit)
plot_res_vs_linear_predictor(fit, residual_fn = rstudent)
plot_res_vs_linear_predictor(fit, residual_fn = residuals)


glm_fit <- glm(cyl ~ mpg, family = poisson(), data = mtcars)

plot_res_vs_linear_predictor(glm_fit)
plot_res_vs_linear_predictor(glm_fit, type = "pearson")
```

---

refit_model *Refit Model*

Description

Refit a model with a new response.

Usage

```
refit_model(object, newresp, ...)
```

Arguments

`object`	A model.
`newresp`	the new response, may be a vector or a matrix.
`...`	other arguments passed to `refit` or `update`.

Details

This function uses `newresp` to refit `object` replacing its old response variable. If the class is `merMod` it uses `refit`, otherwise uses `stats::update()`.

The default method tries to update the model response using it's `stats::model.frame()`, if it errors it tries to update the model by inserting the `newresp` directly into the object formula.

Value

A model with same class as `object`.

`select_covariates` *Select covariates*

Description

Select covariates

Usage

```
select_covariates(
  model,
  threshold = 0.15,
  direction = c("both", "backward", "forward"),
  addable_coefs = names(get_fixef(model)),
  measure_fn = function(x) summary(x)[["coefficients"]][, 4],
  measure_one_at_time = FALSE,
  minimize_only = FALSE,
  max_steps = 1000,
  return_step_results = FALSE,
  do_not_remove = c("(Intercept)"),
  ...
)
```

Arguments

`model`	A model with `stats::update()`, `stats::coef()` methods.
`threshold`	Value threshold to remove variable. It can be a fixed value or a function. The variable is removed if `measure_fn(model) > threshold` and added if `measure_fn(model) <= threshold`.
`direction`	The direction of variable selection. Options include "backward", "forward", or "both". Defaults to "both".
`addable_coefs`	A vector of coefficients that can be added during forward selection. Defaults to all coefficients in the model.
`measure_fn`	Function with model as argument and returns values to be used by `threshold`. It can also compare two models, where during forward step it calls `measure_fn(candidate_model, current_selected_model)` and during backward step it calls `measure_fn(current_selected_model, candidate_model)`. Defaults to the p-value from the summary of the coefficients.
`measure_one_at_time`	Boolean indicating to apply `measure_fn` to each variable individually during forward and backward steps. Set this option to `TRUE` if `measure_fn` returns an atomic value, for example if `measure_fn` is `AIC`.

minimize_only	Logical indicating that during backward model update it should minimize the measure_fn instead of maximize it.
max_steps	The maximum number of steps for the variable selection process. Defaults to 1000.
return_step_results	Logical. If TRUE, the function returns a list containing the final fitted model and a log of the selection steps. Defaults to FALSE.
do_not_remove	A character vector specifying variables that should not be removed during backward selection. Defaults to "(Intercept)".
...	Extra arguments to stats::update().

Value

A fitted model with selected covariates based on the variable selection process. If return_step_results is TRUE, a list containing the final fitted model and a log of the selection steps is returned.

Examples

```
model <- lm(mpg ~ ., data = mtcars)
select_covariates(model)

## measure_fn with two parameters

lrt <- function(model1, model2) {
  lrt_stat <- 2 * (logLik(model1)[1L] - logLik(model2)[1L])
  return(1 - pchisq(lrt_stat, 1))
}

select_covariates(model, measure_fn = lrt)

## AICc selection

AICc <- function(model) {
  loglike <- logLik(model)
  df <- attr(loglike, "df")
  nobs <- attr(loglike, "nobs")
  aic <- -2 * as.numeric(loglike) + 2 * df

  aicc <- aic + (2 * (df^2) + 2 * df) / (nobs - df - 1)

  return(aicc)
}

selection <- select_covariates(model,
  measure_fn = AICc,
  threshold = AICc,
  measure_one_at_time = TRUE,
  minimize_only = TRUE,
  direction = "both",
  data = mtcars
)
```

simulate_wald_pvalues *Generate Wald test P-Values with Monte Carlo Simulations*

Description

This function performs Monte Carlo simulations to generate p-values for model coefficients by refitting the model with new simulated responses and computing the Wald test statistics for each simulation. It's standard behavior verify if the type I error from Wald tests are under control, considering the provided model as "true", i.e., the model assumptions are valid. It supports univariate and joint Wald tests, using chi-squared distributions to calculate p-values.

Usage

```
simulate_wald_pvalues(
  model,
  nsim = 1000,
  responses = NULL,
  test_coefficients = NULL,
  refit_fn = NULL,
  coef_fn = get_fixef,
  vcov_fn = get_vcov,
  show_progress = TRUE,
  plot.it = TRUE,
  ...
)
```

Arguments

model	A fitted model object that will be used to simulate responses.
nsim	The number of simulations to perform.
responses	An optional list of values to be used as response variables to refit the model.
test_coefficients	Numeric vector. A vector with values to be used to compute the test statistic. It should be the coefficients that was used to compute the fitted values of the response. If NULL defaults to coef_fn(model).
refit_fn	Function to refit the model with new responses. If NULL, defaults to get_refit(model, y, ...).
coef_fn	Function that retrieves the coefficients of the model.
vcov_fn	Function that computes the variance-covariance matrix for the models adjusted in the simulations.
show_progress	Display a progress bar for the simulation iteration.
plot.it	Logical. Generate ecdf plot for joint Wald test.
...	Additional arguments to be passed to refit_fn.

Details

If `responses` is provided, the function refits the model with these new response vectors. Otherwise, it generates new responses with `stats::simulate()`.

For each new response calls `get_refit()` to generate a new model with the new response. It gets the fixed effects and the variance and covariance matrix with `get_fixef()` and `get_vcov()`.

Each simulated model is refitted using the specified `refit_fn` (or the default refit function) and the fixed effects coefficients and variance-covariance matrix are extracted using `coef_fn` and `vcov_fn`, respectively. The univariate Wald test is computed from the Wald statistic for each coefficient, while the joint Wald test uses the inverse variance-covariance matrix to compute a Wald statistic for the test_coefficients. P-values are calculated from a chi-squared distribution with appropriate degrees of freedom.

Value

An object of class `AD_pvalues`, which contains the following components:

test_coefficients Vector of coefficients being tested.

pvalues_matrix Matrix of p-values where each column corresponds to a simulation and each row corresponds to a coefficient.

pvalues_joint Vector containing the joint p-values obtained from each simulation.

simulation_fixef List of fixed effect coefficient estimates from each simulation.

simulation_vcov List of covariance matrices estimated from each simulation.

simulation_warning Vector of boolean indicating if a simulation threw a warning.

converged Logical vector indicating whether model refitting converged for each simulation.

responses Simulated responses used for refitting the model.

See Also

`plot.AD_pvalues()` for plotting.

Examples

```
# from help("glm")
counts <- c(18, 17, 15, 20, 10, 20, 25, 13, 12)
outcome <- gl(3, 1, 9)
treatment <- gl(3, 3)
model <- glm(counts ~ outcome + treatment, family = poisson())
new_responses <- replicate(100, MASS::rnegbin(fitted.values(model), theta = 4.5), simplify = FALSE)
simulate_wald_pvalues(model, responses = new_responses, nsim = 100)

## Using custom refit_fn
if (require("survival")) {
  fit <- survreg(Surv(futime, fustat) ~ ecog.ps + rx, ovarian,
    dist = "exponential"
  )
  fitted_rate <- 1 / fitted(fit)
 new_responses <- replicate(100, rexp(length(fitted_rate), fitted_rate), simplify = FALSE)
```

```
  refit_surv_ovarian <- function(.y) {
    survreg(Surv(.y, fustat) ~ ecog.ps + rx, ovarian, dist = "exponential")
  }
  simulate_wald_pvalues(fit, responses = new_responses, refit_fn = refit_surv_ovarian)
}
```

Index

concat_pvalues, 2
cooks.distance, *14*
cooks.distance(), *14*

envelope, 3
envelope(), *11*
envelope_residual, 5
envelope_residual(), *3*, *11*

fitted(), *15, 16*

get_converged, 6
get_fixef, 6
get_fixef(), *20*
get_model_response, 7
get_refit, *4*, 8
get_refit(), *3, 4*, *20*
get_vcov, 8
get_vcov(), *20*
glm(), *16*
graphics::plot, *11*, *13*, *15*
graphics::plot(), *14*

lme4::refit(), *8*

model.frame(), *7*

parametric_bootstrap, 9
parametric_bootstrap(), *4*
plot, *14, 15*
plot(), *15*
plot.AD_envelope, *4*, 11
plot.AD_pvalues, 12
plot.AD_pvalues(), *20*
plot_cook, 13
plot_ecdf_pvalue, 14
plot_res_vs_linear_predictor, 15
predict(), *15, 16*

refit_model, 16
rstudent, *4*
rstudent(), *5*

select_covariates, 17
simulate, *4*
simulate_wald_pvalues, 19
simulate_wald_pvalues(), *12*
stats::coef(), *7*, *17*
stats::ks.test(), *13*
stats::model.frame(), *16*
stats::residuals(), *5*
stats::simulate(), *4*, *20*
stats::update(), *8*, *16–18*
stats::vcov(), *9*