Package: adlaplace (via r-universe)

July 17, 2026

Type Package

Title Laplace Approximations Using Automatic Differentiation

Version 0.5.2

Maintainer Patrick Brown <patrick.borgnine@gmail.com>

Description Computes Laplace approximations for hierarchical models using automatic differentiation (CppAD) and trust-region optimization.

License MPL-2.0

Depends R (>= 3.5.0)

Imports Matrix (>= 1.2.18), Rcpp (>= 1.0.3), methods, data.table, stats, graphics, RCppAD

Suggests numDeriv, abind, knitr, rmarkdown, trustOptim, testthat (>= 3.0.0), mvQuad, mgcv, MASS, nlme, sn, terra

Config/testthat/edition 3

VignetteBuilder knitr

Enhances RSpectra

LinkingTo Rcpp, RcppEigen, trustOptim, Matrix, RCppAD

Encoding UTF-8

SystemRequirements OpenMP

RoxygenNote 7.3.3

NeedsCompilation yes

Collate 'RcppExports.R' 'classes.R' 'ad_data.R' 'ad_fun.R' 'ad_fun_ptr.R' 'ad_shards.R' 'data_setup.R' 'effects.R' 'families.R' 'model_data.R' 'log_lik.R' 'fit.R' 'sim_fit.R' 'fit_methods.R' 'fixed_terms.R' 'format_parameters.R' 'germany-data.R' 'hessian_map.R' 'iid.R' 'rpoly.R' 'iwp.R' 'mvn_utils.R' 'prepare_model_rows.R' 'zzz.R'

LazyData true

Repository https://eborgnine.r-universe.dev

Date/Publication 2026-07-17 22:33:33 UTC

RemoteUrl https://github.com/eborgnine/adlaplace

RemoteRef main

RemoteSha 96d35f802cd1c076c717dc2284fafebab64dad52

RemoteSubdir adlaplace

Contents

.type_factor_levels . . . 3
ad_data . . . 3
ad_data-class . . . 5
ad_fun . . . 5
ad_fun-class . . . 6
ad_fun_ptr . . . 7
ad_fun_ptr-class . . . 8
ad_shards . . . 8
adlaplace . . . 9
adlaplace_cpp . . . 11
adlaplace_fit-methods . . . 12
apply_theta_log . . . 13
binomial-class . . . 14
by_group . . . 15
by_group-class . . . 15
c.ad_fun_ptr . . . 16
clone_ad_fun_ptr . . . 16
collect_terms . . . 17
format_parameters . . . 17
fpoly-class . . . 18
gamm . . . 19
gaussian-class . . . 20
germany . . . 21
has_openmp . . . 22
iid-class . . . 22
inner_opt . . . 23
intercept-class . . . 24
iwp . . . 26
iwp-class . . . 27
laplace_half_H_inv . . . 28
linear-class . . . 28
log_lik_laplace . . . 30
mean_mvquad . . . 31
model-class . . . 32
model-generics . . . 33
model_data . . . 34
nbinom-class . . . 35
outer_optim_wrappers . . . 36
rmvnldl . . . 37
rpoly-class . . . 38

sim_fit 39
sizes 40

Index **41**

`.type_factor_levels` *Valid model type factor levels*

Description

Character vector of allowed levels for the `type` slot on `model` objects.

Usage

```
.type_factor_levels
```

Format

An object of class `character` of length 3.

`ad_data` *Build design matrices and parameter metadata from a formula*

Description

Build design matrices and parameter metadata from a formula

Usage

```
ad_data(
  y = missing(),
  A = NULL,
  X = NULL,
  beta_map = NULL,
  gamma_map = NULL,
  theta_map = NULL,
  elgm_matrix = NULL,
  ad_fun = NA_character_,
  ad_kind = NA_character_,
  package = NA_character_,
  precision = NULL,
  weights = numeric(0)
)

data_setup(formula, data, verbose = FALSE)
```

Arguments

y	Response vector (default empty), or an existing ad_data object to recast as another shard.
A	Random-effects design matrix (nrow(A) = length(y); any Matrix class).
X	Fixed-effects design matrix (nrow(X) = length(y); any Matrix class).
beta_map	Beta parameter map (any Matrix; coerced to ngCMatrix), or a shorthand: length-1 integer n gives Matrix::Matrix(nrow = n, ncol = 0); length-2 c(row, n) gives one column with a structural 1 at row row (1-based); and list(indices, n) gives one column per entry in indices with structural 1s at those rows. nrow is the global beta block size; column j maps local X column j to a global beta row. When ncol(X) > 0, ncol(beta_map) must equal ncol(X) with exactly one nonzero per column. If NULL, defaults to Matrix::Diagonal(ncol(X)) when ncol(X) > 0, otherwise an empty matrix.
gamma_map	Gamma parameter map (any Matrix; coerced to ngCMatrix; at most one nonzero per row and column), or the same shorthand forms as beta_map. Column j maps local A column j into the global gamma block. When ncol(A) > 0, ncol(gamma_map) must equal ncol(A) with exactly one nonzero per column. If NULL, defaults to Matrix::Diagonal(ncol(A)) when ncol(A) > 0, otherwise an empty matrix.
theta_map	Theta parameter map (any Matrix; coerced to ngCMatrix), or the same shorthand forms as beta_map. Column j selects global theta row for component j. If NULL, defaults to an empty matrix.
elgm_matrix	Optional ELGM map (any Matrix; coerced to ngCMatrix). If NULL, defaults to Matrix::Matrix(nrow = length(y), ncol = 0).
ad_fun	Registered AD density name for this shard (optional).
ad_kind	Shard kind ("observations", "parameters", "random"; optional).
package	Package recording AD tapes for this shard (optional).
precision	Optional precision payload (any R object).
weights	Optional per-observation weights (e.g. binomial trial counts). Empty means all ones.
formula	Model formula or named list of term objects.
data	Data frame.
verbose	Print parsing messages.

Value

List with terms, y, A, X, data, info. The info$theta data frame includes a logical transform column: TRUE means the parameter is stored on the log scale when config$transform_theta is TRUE (default for all rows unless a model term sets otherwise). info$parameters stacks beta and theta rows with the same transform column (FALSE for fixed effects); init, lower, and upper are already on the optimization (log) scale where transform is TRUE.

Functions

- ad_data(): Construct an ad_data object

ad_data-class *Per-shard layout for AD density evaluation*

Description

Per-shard layout for AD density evaluation

Slots

`y` Response vector.

`ATp` Transpose of random-effects design matrix.

`XTp` Transpose of fixed-effects design matrix.

`beta_map` Beta parameter map (`ngCMatrix`). `nrow` is the global beta block size; column `j` has one structural nonzero at row `r` meaning local design column `j` of `X` uses global beta `r`. When `ncol(X) > 0`, require `ncol(beta_map) == ncol(X)`.

`gamma_map` Gamma parameter map (`ngCMatrix`); at most one structural nonzero per row and column. Same column convention as `beta_map` for local `A` columns into the global gamma block.

`theta_map` Theta parameter map (`ngCMatrix`). Column `j` selects global theta row `r` for this shard's theta component `j`.

`elgm_matrix` Optional exposure-lag map (`ngCMatrix`; empty by default).

`ad_fun` Registered AD density name for this shard.

`ad_kind` Shard kind (`"observations"`, `"parameters"`, `"random"`).

`package` Package name whose shared library records tapes for this shard (defaults to `"adlaplace"` when missing).

`precision` Optional precision payload (any R object).

`weights` Optional per-observation weights (e.g. binomial trial counts). Empty means all ones.

ad_fun *Build AD function with Hessian templates*

Description

Build AD function with Hessian templates

Usage

```
ad_fun(x, config = NULL, num_threads = 1L, ...)

## S4 method for signature 'ad_fun_ptr'
ad_fun(x, config = NULL, num_threads = 1L, ...)

## S4 method for signature 'ad_data'
ad_fun(x, config = NULL, num_threads = 1L, ...)

## S4 method for signature 'list'
ad_fun(x, config = NULL, num_threads = 1L, ...)
```

Arguments

x	One of: an ad_fun_ptr, an ad_data shard with ad_kind/ad_fun set, or a model_data() bundle (list with observations, random, parameters).
config	Model configuration list. Required for ad_data and model_data; unused for ad_fun_ptr. For model_data, beta, gamma, and theta are filled from x$data$info when omitted (only shards, transform_theta, etc. are needed). When config$num_threads is set, it overrides the num_threads argument for ad_data and model_data methods.
num_threads	Positive integer; OpenMP thread count for inner_opt and parallel trace_hinv_t. Shards are assigned owner_thread = shard_index % num_threads at attach time. Default 1L (serial).
...	For ad_fun_ptr input, optional additional ad_fun_ptr shards to combine before attaching templates.

Value

Object of class ad_fun.

---

ad_fun-class *AD function with Hessian templates attached*

Description

AD function with Hessian templates attached

Slots

ptr Raw combined handle (ad_fun_ptr).

group_sparsity Per-group inner gradient sparsity patterns.

outer Outer Hessian template (dgCMatrix).

inner Inner-gamma Hessian template (dgCMatrix).

map_outer Outer Hessian shard map.

`map_inner` Inner Hessian shard map.

`parallel_map` Thread-affinity map (`ngCMatrix`); rows are shards, columns are threads, with one nonzero per shard row.

`chol_inner` Symbolic LDL factor or empty sparse matrix.

`chol_inner_list` Numeric LDL list for C++ (`L1`, `Linv`, `perm`, `perm_inv`, `half_H_inv`, `H_inv`), plus `trace_columns` (per-shard column indices for `trace_hinv_t`).

`sizes` Named numeric vector `beta`/`gamma`/`theta`.

`info` List of parameter metadata (`beta`, `gamma`, `theta`, `parameters`); populated from `model_data()$data$info` when the handle is built from a model-data bundle, otherwise empty.

`ad_fun_ptr` *Build raw AD handle for one density shard*

Description

Constructs CppAD tapes for a single shard. The density kind and name come from `data@ad_kind` and `data@ad_fun`. For `ad_kind = "random"`, `data@precision` is passed straight to the backend. For `random_diagonal` it must be a numeric vector of diagonal precision weights with `length == ncol(gamma_map)`. Missing precision means the shard is not built (e.g. diffuse `rpoly` with `sd = Inf` via `model_data()`); calling `ad_fun_ptr()` without it is an error.

Usage

```
ad_fun_ptr(data, config)
```

Arguments

`data`	An `ad_data` object with `ad_kind` and `ad_fun` slots set.
`config`	Model configuration list (`beta`, `theta`, etc.). `gamma` is optional; when missing, zeros of length `nrow(gamma_map)` are used as AD tape seeds. `config$num_threads` does not assign OpenMP threads; use `num_threads` on `ad_fun` after merging shards.

Details

Merge handles for multiple shards with `c()` before calling `ad_fun`.

Value

External pointer of class `ad_fun_ptr`.

ad_fun_ptr-class *Raw AD handle (external pointer)*

Description

S4 class for a C++ AD backend handle. Objects are created by ad_fun_ptr, combined with c.ad_fun_ptr, and typically passed to ad_fun to attach Hessian templates.

See Also

ad_fun_ptr, ad_fun, clone_ad_fun_ptr

ad_shards *Partition observations into AD shards by sparsity pattern*

Description

The function partitions columns into num_shards shards using quantiles of the first right singular vector, optionally using **RSpectra** for efficiency when available.

Arguments

A	Random-effects design matrix (nrow(A) = number of observations).
elgm_matrix	A numeric matrix (or matrix-like object) for extended latent gaussian models.
num_shards	Integer giving the maximum number of shards to construct. The actual number of shards may be smaller if fewer distinct loadings are present.
min_groups	Integer giving the minimum number of shards. When there are fewer distinct singular-vector loadings than min_groups, only the largest exact-loading group is split until min_groups is reached (or that group cannot be split further).

Details

The resulting grouping is returned as a sparse matrix whose columns correspond to shards and whose entries are the singular-vector loadings. Shards are ordered from most heterogeneous to most homogeneous.

If the **RSpectra** package is available, the leading singular vector is computed using RSpectra::svds; otherwise, a full singular value decomposition via svd is used.

Shard boundaries are defined by empirical quantiles of the loadings. Shards are subsequently re-ordered so that shards with larger within-shard variability appear first.

Value

A sparse matrix of class "dgCMatrix" (from **Matrix**), with one column per shard and one row per observation. Nonzero entries correspond to singular-vector loadings.

Examples

```
set.seed(1)
A <- matrix(rnorm(100), 20, 5)
G <- ad_shards(A, num_shards = 3)
G
```

`adlaplace` *Fit a hierarchical model by Laplace-approximate maximum likelihood*

Description

One-call interface to the **adlaplace** pipeline: parse the formula with `model_data()`, partition observations into AD shards with `ad_shards()`, build the AD handle with `ad_fun()`, maximize the Laplace-approximate marginal likelihood with `optim()` using the exact profile gradient (`outer_fn` / `outer_gr`), and return a fitted-model object with standard accessor methods.

Usage

```
adlaplace(
  formula,
  data,
  config = list(),
  num_threads = 1L,
  num_shards = 100L,
  control = list(),
  control_inner = list(maxit = 100L, report.level = 0, report.freq = 0),
  method = "L-BFGS-B",
  hessian = TRUE,
  verbose = FALSE,
  na_omit = TRUE
)
```

Arguments

`formula`	Model formula built from model terms, e.g. `nbinom(y) ~ x + iid(g)`, a list of term objects from `collect_terms`, or a precomputed `model_data` result (in which case `data` is ignored).
`data`	Data frame containing the variables referenced in `formula`. Not needed when `formula` is already a `model_data` result.
`config`	List of backend options merged over the defaults `list(transform_theta = TRUE, verbose = verbose)`. When `config$shards` is missing it is filled by `ad_shards(A, num_shards = num_shards)`.
`num_threads`	Positive integer, OpenMP thread count for the AD handle.
`num_shards`	Target number of observation shards for `ad_shards()`.
`control`	List of `optim` control options, merged over the defaults `list(maxit = 200L, parscale = <from terms>)`.

control_inner	List of control options for the inner (random effects) trust-region optimizer.
method	optim method; bounds are used for "L-BFGS-B" and "Brent" only.
hessian	Logical. When TRUE (default), optim numerically differentiates the exact AD profile gradient at the optimum, giving the observed information used by vcov.adlaplace_fit, summary.adlaplace_fit, and confint.adlaplace_fit.
verbose	Logical, print progress.
na_omit	Passed to model_data().

Value

An object of class "adlaplace_fit": a list with components

call, formula The matched call and model formula.

par Named outer parameter estimates on the optimization scale (log scale where par_info$transform is TRUE).

par_info Parameter metadata table (info$parameters).

vcov Inverse of the numerically differentiated Hessian of the negative log likelihood (optimization scale), or NULL.

gamma Named random-effect modes at the optimum.

logLik, nobs Laplace-approximate marginal log likelihood and the number of observations.

optim Full optim result, including hessian when requested.

details Result of log_lik_laplace(deriv = TRUE) at the optimum (gradient, Cholesky factors, inner solution).

model_data, ad_fun, config, control_inner, cache Objects needed by the accessor methods and for warm restarts.

See Also

summary.adlaplace_fit, predict.adlaplace_fit, log_lik_laplace, model_data

Examples

```
## Not run:
fit <- adlaplace(
  nbinom(y, lower = 1e-9, init = 0.15) ~ x + iid(g, init = 0.1),
  data = dat, num_threads = 2L
)
summary(fit)
confint(fit)

## End(Not run)
```

adlaplace_cpp *C++ backend entry points*

Description

Low-level entry points exposed to R. Accept an ad_fun S4 object (via @ptr) or a raw ad_fun_ptr.

Usage

```
joint_log_dens(ad_fun, x, shards = NULL, negative = TRUE)

grad(ad_fun, x, shards = NULL, inner = FALSE, negative = TRUE)

hessian(
  ad_fun,
  x,
  shards = NULL,
  inner = FALSE,
  verbose = FALSE,
  negative = TRUE
)

trace_hinv_t(ad_fun, x, LinvPt, LinvPtColumns, verbose = FALSE)

fun_obj_fdfh(ad_fun, parameters, gamma, inner = TRUE, verbose = FALSE)
```

Arguments

ad_fun	An ad_fun S4 object or ad_fun_ptr handle.
x	Numeric parameter vector of length Nparams.
shards	Optional integer vector of 0-based shard indices; NULL or integer(0) evaluates all shards.
negative	Logical (default TRUE). If TRUE, return the **negative** log density $-\ell(x)$ and its derivatives (minimization / **trustOptim** sign, consistent with inner_opt()). If FALSE, return $\ell(x)$, $\nabla\ell$, and $\nabla^2\ell$.
inner	Logical; if TRUE, evaluate inner-γ derivatives; if FALSE, evaluate outer derivatives at the full parameter vector.
verbose	Logical; if TRUE, print threads, shards, and CppAD session phases.
LinvPt, LinvPtColumns	Sparse factors for trace_hinv_t().
parameters	Numeric vector of length Nbeta + Ntheta.
gamma	Numeric vector of length Ngamma.

Value

See individual functions.

Sign convention

With default `negative = TRUE`, `joint_log_dens()`, `grad()`, and `hessian()` match `inner_opt()` (negative log-density). Set `negative = FALSE` for the joint log density and its derivatives at the same `x`.

`adlaplace_fit-methods` *Methods for fitted adlaplace models*

Description

Standard accessor methods for objects of class `"adlaplace_fit"` returned by `adlaplace()`. Estimates are stored on the optimization scale (log scale for parameters with `par_info$transform = TRUE`, e.g. standard deviations); methods with a `transform` argument report the natural scale by exponentiating and, for standard errors, applying the delta method.

Usage

```
## S3 method for class 'adlaplace_fit'
coef(object, transform = TRUE, ...)

## S3 method for class 'adlaplace_fit'
vcov(object, transform = FALSE, ...)

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

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

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

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

## S3 method for class 'adlaplace_fit'
predict(object, newdata, n = 500L, ...)

## S3 method for class 'adlaplace_fit'
print(x, digits = max(3L, getOption("digits") - 3L), ...)

## S3 method for class 'adlaplace_fit'
summary(object, level = 0.95, ...)

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

```
## S3 method for class 'adlaplace_fit'
plot(x, k = 200L, draws = 100L, ...)
```

Arguments

object, x	An adlaplace_fit object.
transform	Logical; report transformed (natural-scale) values.
...	Passed to downstream methods.
parm	Optional subset of parameter labels or indices.
level	Confidence level.
newdata	Data frame of prediction covariates.
n	Number of random-effect draws for predict.
digits	Number of significant digits to print.
k	Number of evaluation points per smooth term in plot.
draws	Number of simulation draws per smooth term in plot.

Details

predict draws random effects from the Laplace (Gaussian) approximation at the mode via sim_fit and returns the linear predictor on newdata: a nrow(newdata) by n matrix of simulated values (plus fixed-effect contributions from terms whose variables appear in newdata). Without newdata it returns fitted(object).

plot draws each smooth term (terms with more than one knot) on a grid spanning its knots, showing simulated curves from the Laplace approximation with the pointwise mean in red. With no smooth terms it plots the random-effect modes.

---

apply_theta_log *Apply log transform to selected theta columns*

Description

When active is TRUE, logs values in cols for rows where theta_info$transform is TRUE. Missing or NA transform entries default to TRUE.

Usage

```
apply_theta_log(theta_info, cols = "init", active = TRUE)
```

Arguments

theta_info	Data frame from info$theta (see data_setup).
cols	Character vector of column names to transform (e.g. "init").
active	Logical; when FALSE, returns theta_info unchanged.

Value

A copy of `theta_info` with selected entries logged.

`binomial-class` *Binomial observation term*

Description

Model term for the observation-level binomial log density with a logit link, registered as `binomial_obs`. For each observation with linear predictor `eta = X beta + A gamma`, trial count `N`, and success count `y`, the log density contribution is `y * eta - N * log(1 + exp(eta))` (up to a data-only constant). When `size` is omitted, `N = 1` (Bernoulli). There are no observation-level hyperparameters.

Creates a response term wired to the `binomial_obs` AD shard. A Bernoulli likelihood (`size` omitted) has no observation-level hyperparameters. With `size`, trial counts are read from that column of `data` and stored as observation weights.

When called with no response variable this function falls back to `stats::binomial()`, so `glm(..., family = binomial)` and `MASS::glmmPQL(..., family = binomial)` keep working with **adlaplace** attached.

Usage

```
binomial(x, size = NULL, link = "logit")
```

Arguments

`x`	Outcome variable name (counts of successes; 0/1 when Bernoulli).
`size`	Optional column name for the number of trials per observation.
`link`	Link passed to `stats::binomial()` in the fallback case.
`term`	A model term object (S4 methods).
`data`	A data frame containing the variables (S4 methods).

Details

Binomial observation model term

Value

A `binomial` object (or a `stats::family` when called with no response variable).

Slots (inherited from `model`)

`ad_fun` Character scalar `"binomial_obs"`.

`ad_kind` Character scalar `"observations"`.

Slots

`size` Optional column name for per-observation trial counts.

by_group *By-Group Classes and Functions*

Description

Functions for creating and managing by_group objects for hierarchical model grouping information.

Usage

```
by_group(term, data = NULL, levels = NULL)
```

Arguments

term	Character, the grouping variable name
data	A data frame. If provided and levels is missing, unique values are extracted from data[[term]]
levels	Character vector of unique levels for the grouping variable. If missing and data is provided, unique values are extracted from data[[term]]

Value

A by_group object

Examples

```
# Create with explicit levels
bg <- by_group(term = "group", levels = c("A", "B", "C"))

# Create from data
dat <- data.frame(group = c("A", "B", "A", "C"))
bg <- by_group(term = "group", data = dat)
```

by_group-class *By-Group Class*

Description

A class for grouping information in hierarchical model terms. Contains the grouping variable name and its levels/labels.

Slots

`term` Character vector of length 1, the grouping variable name

`levels` Character vector of unique levels for the grouping variable

`labels` Character vector of formatted labels for the levels

c.ad_fun_ptr *Combine* ad_fun_ptr *handles*

Description

Moves AD shards from one or more ad_fun_ptr objects into a single handle. Inputs are cleared after the merge (ownership moves).

Usage

```
## S3 method for class 'ad_fun_ptr'
c(x, ...)
```

Arguments

x	An ad_fun_ptr.
...	Additional ad_fun_ptr objects.

Value

A combined ad_fun_ptr.

See Also

clone_ad_fun_ptr, ad_fun

clone_ad_fun_ptr *Deep copy of an* ad_fun_ptr *handle*

Description

Returns a new ad_fun_ptr with independent C++ state (CppAD tapes and sparsity patterns). Hessian templates are not copied; call ad_fun on the clone to attach maps. The source handle is left unchanged, unlike c.ad_fun_ptr which moves shards and clears the inputs.

Usage

```
clone_ad_fun_ptr(x)
```

Arguments

x	An ad_fun_ptr object.

Value

A new ad_fun_ptr.

collect_terms *Parse Model Terms from Formula*

Description

Parses a formula and creates model terms by evaluating constructor calls in the formula environment (so `pkg::term(...)` and terms from attached packages on the search path work). Falls back to the **adlaplace** namespace for built-in constructors.

Usage

```
collect_terms(formula, verbose = FALSE)
```

Arguments

`formula`	Model formula with constructor terms (e.g. `iwp(x, ...)`, `iid(fac)`, `nbinom(y, ...)` on the LHS). Bare symbols such as `x` are coerced to `linear(x)`. Unnamed first positional symbols are treated as column names in `data`; named arguments (e.g. `x = sites` for spatial terms) are evaluated as objects in the formula environment. The named argument `size` is also treated as a column name when passed as a symbol (`binomial(y, size = N)`).
`verbose`	print extra information

Value

List of model term objects

format_parameters *Format Laplace estimates using parameter metadata*

Description

Maps flat Laplace parameter vectors onto labeled tables from `info` (as stored on `ad_fun@info` or `model_data()$data$info`).

Usage

```
format_parameters(
  info,
  full_parameters = NULL,
  parameters = NULL,
  gamma = NULL
)
```

Arguments

info	List with beta, gamma, theta, and parameters data frames.
full_parameters	Numeric vector c(beta, gamma, theta). When supplied, parameters and gamma are ignored.
parameters	Numeric outer vector c(beta, theta); used when full_parameters is NULL.
gamma	Numeric inner vector; used when full_parameters is NULL.

Value

A list with data frames parameters and gamma, or list() when info is missing or incomplete.

---

fpoly-class *Fixed Polynomial Model Term*

Description

Creates and manages fixed polynomial model terms.

Usage

```
fpoly(
  x,
  p = 2,
  ref_value = 0,
  init = .my_beta_init,
  lower = .my_beta_lower,
  upper = .my_beta_upper,
  parscale = .my_beta_parscale
)

## S4 method for signature 'fpoly'
design(term, data)

## S4 method for signature 'fpoly'
beta_info(term, data)
```

Arguments

x	Variable name.
p	Polynomial degree (default: 2).
ref_value	Reference value for the polynomial.
init	Initial values for beta parameters.
lower	Lower bounds for beta parameters.

upper	Upper bounds for beta parameters.
parscale	Parameter scales for optimization.
term	A ‘fpoly‘ term object.
data	A data frame containing the variables used in the term.

Value

A ‘fpoly‘ term object.

A design matrix for the fixed polynomial term, or NULL if p.order is 0.

A data frame containing beta parameter information for the fixed polynomial term.

Functions

- `design(fpoly)`: Design method for fpoly objects
- `beta_info(fpoly)`: Beta info method for fpoly objects

Methods

The following methods are available for ‘fpoly‘ objects:

`design(term, data)` Creates design matrix for fpoly term

`precision(term, data)` Creates precision matrix for fpoly term

`theta_info(term)` Extracts theta parameter information

`beta_info(term, data)` Extracts beta parameter information

`random_info(term, data)` Extracts random effects information

Examples

```
# Create a fixed polynomial term
fpoly_term <- fpoly(x = "temp", p = 3, ref_value = 15)
```

---

gamm *Example GAMM simulation data*

Description

Simulation data for the GAMM vignette example.

Format

A data frame `dat` with 500 rows and columns from `mgcv::gamSim(6, n = 500)`, plus simulated negative-binomial responses used in the GAMM vignette (`y`, `f2`, `mu`, `Z`, etc.).

Source

Generated with `mgcv::gamSim()` and a custom negative-binomial response; see the regeneration comments in `vignettes/gamm.Rmd`.

Examples

```
data("gamm", package = "adlaplace")
exists("dat")
nrow(dat)
```

`gaussian-class` *Gaussian observation term*

Description

Model term for the observation-level Gaussian log density registered as `gaussian_obs`. This is the default observation term: a bare response symbol on the left-hand side of a formula (e.g. `y ~ x`) is coerced to `gaussian(y)` by `collect_terms`.

Creates a response term wired to the `gaussian_obs` AD shard, with a single residual standard deviation parameter (log scale during optimization).

When called with no arguments this function falls back to `stats::gaussian()`, so `glm(..., family = gaussian)` keeps working with **adlaplace** attached.

Usage

```
gaussian(
  x,
  init = 1,
  lower = .my_theta_lower,
  upper = .my_theta_upper,
  parscale = .my_theta_parscale
)

## S4 method for signature 'gaussian'
theta_info(term)
```

Arguments

`x`	Outcome variable name.
`init`	Initial value for the residual standard deviation.
`lower`	Lower bound for the residual standard deviation.
`upper`	Upper bound for the residual standard deviation.
`parscale`	Parameter scale for optimization.
`term`	A model term object (S4 methods).
`data`	A data frame containing the variables (S4 methods).

Details

Gaussian observation model term

Value

A `gaussian` object (or a `stats::family` when called with no arguments).

Functions

- `theta_info(gaussian)`: Theta info for the residual standard deviation.

Slots (inherited from `model`)

`ad_fun` Character scalar `"gaussian_obs"`.

`ad_kind` Character scalar `"observations"`.

`germany` *Germany oral cavity cancer (Besag-York-Mollie example)*

Description

Disease-mapping data distributed with **INLA** (Besag, York, and Mollie 1991). Counts y_i in 544 districts follow $y_i \sim \text{Poisson}(E_i e^{\eta_i})$ with a BYM spatial random effect. The bundled ICAR precision `Q_scaled` is the INLA `scale.model` version with sum-to-zero constraint; `prec` is the `random_mult` precision payload (`Q`, `log_det`, `rank`).

Usage

```
germany
```

Format

A list with:

Y Observed case counts (length 544).

E Expected counts (length 544).

region District index (length 544).

adj District adjacency matrix (`dgCMatrix`, 544 x 544).

Q_scaled Scaled ICAR precision (`dgCMatrix`).

prec List passed to `ad_fun = "random_mult"`: `Q`, `log_det`, and `rank`.

Details

District boundaries are not stored in this object (to avoid a hard dependency on **terra**). They ship as `system.file("extdata", "germany_map.gpkg", package = "adlaplace")` and can be loaded with `terra::vect(...)` when **terra** is installed.

Source

INLA `Germany` data and `germany.graph`.

References

Besag, J., York, J., and Mollie, A. (1991). Bayesian image restoration, with two applications in spatial statistics. *Annals of the Institute of Statistical Mathematics*, 43(1), 1–20.

Rue, H., Martino, S., and Chopin, N. (2009). Approximate Bayesian inference for latent Gaussian models by using integrated nested Laplace approximations. *Journal of the Royal Statistical Society: Series B*, 71(2), 319–392.

---

has_openmp *Whether this build was compiled with OpenMP support.*

Description

Whether this build was compiled with OpenMP support.

Usage

```
has_openmp()
```

Value

`TRUE` if OpenMP was enabled at compile time, otherwise `FALSE`.

---

iid-class *IID Random Effects Term*

Description

Creates and manages IID (independent and identically distributed) random effects terms.

Usage

```
iid(
  x,
  init = .my_theta_init,
  lower = .my_theta_lower,
  upper = .my_theta_upper,
  parscale = .my_theta_parscale
)

## S4 method for signature 'iid'
design(term, data)
```

```
## S4 method for signature 'iid'
precision(term, data)

## S4 method for signature 'iid'
random_info(term, data)

## S4 method for signature 'iid'
theta_info(term)
```

Arguments

x	Variable name (typically a factor).
init	Initial value for theta parameter.
lower	Lower bound for theta parameter.
upper	Upper bound for theta parameter.
parscale	Parameter scale for optimization.
term	An iid term object
data	A data frame containing the term variable

Value

An 'iid' term object (in a named list).

Functions

- `design(iid)`: Creates design matrix for iid term
- `precision(iid)`: Creates precision matrix for iid term
- `random_info(iid)`: Extracts random effects information for iid term
- `theta_info(iid)`: Extracts theta parameter information for iid term

---

inner_opt *Inner optimization over gamma using trust-region CG (sparse)*

Description

Runs the inner optimization over γ using the **trustOptim** sparse trust-region CG solver on a pre-built `ad_fun` handle.

Usage

```
inner_opt(
  parameters,
  gamma,
  ad_fun,
  control = NULL,
  deriv = FALSE,
  verbose = FALSE
)
```

Arguments

parameters	Numeric vector of fixed outer parameters (beta, theta; length Nbeta+Ntheta).
gamma	Numeric vector of starting values for inner parameters (gamma; length Ngamma).
ad_fun	ad_fun S4 object from ad_fun.
control	List of trust-region control parameters (see **trustOptim**).
deriv	Logical; if TRUE, also return outer gradient/Hessian pieces and Cholesky-based quantities at the inner mode.
verbose	Logical; if TRUE, print thread/shard diagnostics.

Details

Negates tape log-density values in C++ for minimization via **trustOptim**.

Value

A list with log_lik, neg_log_lik, parameters, full_parameters, opt, gradient (list inner/outer), and hessian (list inner/outer/chol_inner/half_log_det; with deriv=TRUE also half_H_inv, H_inv, trace3). Objective and derivatives use the negative log-density convention.

---

intercept-class *intercept Model Term*

Description

Creates and manages intercept model terms for fixed effects.

Usage

```
intercept(
  init = .my_beta_init,
  lower = .my_beta_lower,
  upper = .my_beta_upper,
  parscale = .my_beta_parscale
)
```

```
## S4 method for signature 'intercept'
design(term, data)

## S4 method for signature 'intercept'
beta_info(term, data)
```

Arguments

`init`	Initial value for beta parameter (default: 0)
`lower`	Lower bound for beta parameter (default: -Inf)
`upper`	Upper bound for beta parameter (default: Inf)
`parscale`	Parameter scale for optimization (default: 1)
`term`	A intercept term object
`data`	A data frame containing the term variable

Details

intercept Model Term

Value

A intercept term object

Functions

- `design(intercept)`: Creates design matrix for intercept term
- `beta_info(intercept)`: Extracts beta parameter information for intercept term

Methods

The following methods are available for ‘intercept‘ objects:

`design(term, data)` Creates design matrix for intercept term

`precision(term, data)` Creates precision matrix for intercept term

`theta_info(term)` Extracts theta parameter information

`beta_info(term, data)` Extracts beta parameter information

`random_info(term, data)` Extracts random effects information

Examples

```
# Create a intercept term
intercept_term <- intercept(init=2)
```

iwp *Integrated Wiener Process Term Constructor*

Description

Creates an integrated Wiener process (IWP) model term.

Usage

```
iwp(
  x,
  p = 2,
  ref_value = 0,
  knots,
  range = NULL,
  init = .my_theta_init,
  lower = .my_theta_lower,
  upper = .my_theta_upper,
  parscale = .my_theta_parscale,
  boundary_is_random = TRUE,
  include_poly = TRUE
)
```

Arguments

x	Variable name.
p	Order of the integrated Wiener process (default: 2).
ref_value	Reference value for the basis.
knots	Vector of knot locations.
range	Range of the data (optional).
init	Initial values for theta parameters.
lower	Lower bounds for theta parameters.
upper	Upper bounds for theta parameters.
parscale	Parameter scales for optimization.
boundary_is_random	Whether boundary should be treated as random.
include_poly	Whether to include polynomial terms.

Value

A list containing the ‘iwp‘ term object and optionally polynomial terms.

Examples

```
# Example usage:
# knots <- seq(0, 1, length.out = 5)
# iwp_term <- iwp(x = "age", knots = knots)
```

`iwp-class` *Integrated Wiener Process Term*

Description

Creates and manages integrated Wiener process (IWP) model terms.

Usage

```
## S4 method for signature 'iwp'
design(term, data)

## S4 method for signature 'iwp'
precision(term, data)

## S4 method for signature 'iwp'
theta_info(term)

## S4 method for signature 'iwp'
random_info(term, data)
```

Arguments

`term`	An iwp term object
`data`	A data frame containing the term variable

Functions

- `design(iwp)`: Creates design matrix for IWP term
- `precision(iwp)`: Creates precision matrix for IWP term
- `theta_info(iwp)`: Extracts theta parameter information for IWP term
- `random_info(iwp)`: Extracts random effects information for IWP term

Methods

The following methods are available for ‘iwp‘ objects:

`design(term, data)` Creates design matrix for IWP term

`precision(term, data)` Creates precision matrix for IWP term

`theta_info(term)` Extracts theta parameter information

`beta_info(term, data)` Extracts beta parameter information

`random_info(term, data)` Extracts random effects information

laplace_half_H_inv *Extract inner inverse Cholesky factor from Laplace output*

Description

Returns $H^{-1/2}$ for the inner (random-effect) Hessian from the output of log_lik_laplace. Accepts the flat half_H_inv matrix when deriv = TRUE, or reconstructs it from hessian$chol_inner.

Usage

```
laplace_half_H_inv(laplace)
```

Arguments

laplace	Output of log_lik_laplace(..., deriv = TRUE) (or inner_opt(..., deriv = TRUE) with the same extra$hessian layout).

Value

A sparse or dense matrix $H^{-1/2}$ with nrow = Ngamma.

See Also

log_lik_laplace, rmvnldl

linear-class *Linear Model Term*

Description

Creates and manages linear model terms for fixed effects.

Usage

```
linear(
  x,
  init = .my_beta_init,
  lower = .my_beta_lower,
  upper = .my_beta_upper,
  parscale = .my_beta_parscale
)

## S4 method for signature 'linear'
design(term, data)

## S4 method for signature 'linear'
beta_info(term, data)
```

Arguments

x	Variable name (character) or symbol in a formula (e.g. linear(x) in model_data()).
init	Initial value for beta parameter (default: 0)
lower	Lower bound for beta parameter (default: -Inf)
upper	Upper bound for beta parameter (default: Inf)
parscale	Parameter scale for optimization (default: 1)
term	A linear term object
data	A data frame containing the term variable

Details

Linear Model Term

Value

A linear term object

Functions

- design(linear): Creates design matrix for linear term
- beta_info(linear): Extracts beta parameter information for linear term

Methods

The following methods are available for ‘linear‘ objects:

design(term, data) Creates design matrix for linear term

precision(term, data) Creates precision matrix for linear term

theta_info(term) Extracts theta parameter information

beta_info(term, data) Extracts beta parameter information

random_info(term, data) Extracts random effects information

Examples

```
# Create a linear term
linear_term <- linear(x = "temperature")
```

log_lik_laplace *Log-likelihood with inner Laplace optimization*

Description

Evaluates the (profiled) log-likelihood for a hierarchical model by solving an inner optimization problem for the latent vector gamma (e.g. random effects) given outer parameters x (typically beta and theta). The function delegates the inner optimization to inner_opt() from the selected backend package (by default **adlaplace**).

Usage

```
log_lik_laplace(
  x,
  config = list(),
  gamma,
  control = list(report.level = 4, report.freq = 1),
  ad_fun,
  data,
  package = c(config[["package"]], "adlaplace")[1L],
  deriv = FALSE
)
```

Arguments

x	Numeric vector of outer parameters (beta, then theta). Length must equal num_beta + num_theta from ad_fun@sizes.
config	A list of backend options (verbose, package, etc.). When ad_fun is missing, must also include beta, gamma, and theta so ad_fun() can be built from data.
gamma	Optional numeric vector of starting values for the inner parameter gamma. If missing, uses config$gamma when present, otherwise a zero vector of length ad_fun@sizes["gamma"].
control	List of control parameters passed *as-is* to the backend inner optimizer (e.g., report.level, report.freq). See backend documentation (e.g., **trustOptim**) for supported options.
ad_fun	Optional ad_fun object from ad_fun. This is a single backend handle (no separate inner/outer handles). If missing, it will be constructed automatically using data.
data	Optional data list used to build ad_fun when ad_fun is not supplied.
package	Character scalar naming the backend package to use for ad_fun() and inner_opt(). Defaults to "adlaplace". Other backends must export a compatible ad_fun() builder and inner_opt().
deriv	Logical scalar. If TRUE, include derivative quantities in the output (gradient, intermediate derivatives).

Details

The default **adlaplace** backend uses a single AD handle. This function passes that handle to `inner_opt(..., ad_fun = ad_fun)`. Parameter block sizes are read from `ad_fun@sizes`; `config` is not required to contain `beta`, `gamma`, or `theta` when `ad_fun` is supplied.

When `deriv=FALSE`, the return value is the `inner_opt()` list (profile likelihood, nested `gradient` and `hessian`, and `opt`). When `deriv=TRUE`, the same list is augmented with `grad`, `deriv`, and `extra` from the internal profile-derivative helper.

Value

A list. With `deriv=FALSE`, same as `inner_opt`. With `deriv=TRUE`, additionally:

grad Gradient of `neg_log_lik` w.r.t. outer `x`.

deriv Data frame of profile derivative pieces.

extra Intermediate objects (`dU`, `trace3`, Cholesky factors).

Note

The Laplace approximation assumes the inner Hessian is positive definite at the optimum. If `inner_opt` fails to converge or returns a non-invertible Hessian, `log_lik_laplace()` issues a warning or error.

See Also

`ad_fun`, `inner_opt`

---

mean_mvquad *Posterior mean of random effects via multivariate quadrature*

Description

Approximates $E[\gamma \mid y]$ by importance-weighted quadrature over γ, using a Gaussian proposal $N(\hat{\gamma}, H^{-1})$ (Laplace covariance) and correcting with `joint_log_dens`.

Usage

```
mean_mvquad(
  parameters,
  mode,
  cov,
  ad_fun,
  n = 3L,
  type = "nHN",
  ndConstruction = "sparse",
  dec.type = 2L
)
```

Arguments

parameters	Outer (β, θ) at the Laplace fit.
mode	Inner mode $\hat{\gamma}$ (e.g. laplaceoptsolution).
cov	H^{-1} for the inner block (e.g. laplace$extra$hessian$H_inv).
ad_fun	ad_fun object for joint_log_dens.
n	Quadrature level passed to **mvQuad** createNIGrid.
type	mvQuad rule (default "nHN").
ndConstruction	mvQuad grid construction (default "sparse").
dec.type	Cholesky decomposition flag for mvQuad::rescale.

Details

Builds a sparse nHN grid with **mvQuad**, rescales to the Laplace proposal, then forms $\sum_i w_i \gamma_i \exp(\ell(\gamma_i) - \log q(\gamma_i)) / \sum_i w_i \exp(\ell(\gamma_i) - \log q(\gamma_i))$ with log-sum-exp stabilization.

Value

Numeric vector of length n_gamma.

See Also

log_lik_laplace, inner_opt, joint_log_dens, rmvnldl

---

model-class *Base Model Class*

Description

The base S4 class that all specific model term classes inherit from. Subclasses that need hierarchical grouping should include a by slot (either character or by_group type) and optionally by_levels and by_labels slots.

Slots

term Character vector of length 1, the term name

label Character scalar term label reused across metadata builders.

formula Formula object for the term

knots Numeric vector of knot locations (for spline terms)

ref_value Numeric reference value for centering

p.order Integer polynomial order

init Numeric vector of initial values

lower Numeric vector of lower bounds

upper Numeric vector of upper bounds

`parscale` Numeric vector of parameter scales for optimization

`type` Factor indicating term type ("fixed", "random", or "response")

`ad_fun` Registered AD density name for `ad_fun_ptr()`, or `NA` when the term does not map to a density shard.

`ad_kind` Shard kind passed to `ad_fun_ptr()` (`"observations"`, `"parameters"`, `"random"`, or `NA`).

`package` Package whose `.so` records AD tapes for this term (default `"adlaplace"` for built-in densities).

`model-generics` *Model Term Generics*

Description

Generic functions for extracting model matrices and parameter information from model terms.

When non-`NULL`, `model_data` emits an extra `ad_kind = "parameters"` shard (e.g. FEM log-determinant) alongside the random shard. Default is `NULL` (no companion).

Usage

```
design(term, data)

precision(term, data)

theta_info(term)

beta_info(term, data)

random_info(term, data)

elgm_matrix(term, data)

## S4 method for signature 'model'
design(term, data)

## S4 method for signature 'model'
precision(term, data)

## S4 method for signature 'model'
theta_info(term)

## S4 method for signature 'model'
beta_info(term, data)

## S4 method for signature 'model'
```

```
random_info(term, data)

extra_ad_fun(term)

## S4 method for signature 'model'
extra_ad_fun(term)
```

Arguments

term	A model term object.
data	A data frame containing the variables.

Value

Method-specific return values.

Character density name, or NULL.

---

model_data *Assemble model terms and per-shard* ad_data *objects*

Description

Parses a formula, builds design matrices, and returns formula terms plus ad_data shards for the observation density, parameter densities, and each random-effect term.

Usage

```
model_data(formula, data, verbose = FALSE, na_omit = TRUE)
```

Arguments

formula	Model formula with constructor terms (e.g. iwp(), iid()).
data	Data frame containing variables referenced in formula.
verbose	Print extra information while parsing terms.
na_omit	When TRUE (default), drop rows with NA in covariates, outcome, stratification (by), or random-slope mult variables; impute remaining outcome NAs as zero; and sort by observation-term by columns when present.

Value

A list with:

terms Named list of term objects from collect_terms.

data Output of data_setup, including elgm_matrix when an observation term defines an elgm_matrix method.

observations Named list of observation ad_data shards.

parameters Named list of parameter ad_data shards.

random Named list of random-effect ad_data shards.

Examples

```
## Not run:
md <- model_data(
  y ~ x1 + iwp(x2, p = 2, knots = seq(0, 1, len = 11)),
  data = dat
)

## End(Not run)
```

`nbinom-class` *Negative binomial observation term*

Description

Model term for the observation-level negative binomial log density registered as `nbinom_obs`.

Creates a response term wired to the `nbinom_obs` AD shard.

Usage

```
nbinom(
  x,
  init = .my_theta_init,
  lower = .my_theta_lower,
  upper = .my_theta_upper,
  parscale = .my_theta_parscale
)

## S4 method for signature 'nbinom'
theta_info(term)
```

Arguments

`x`	Outcome variable name.
`init`	Initial value for the overdispersion parameter (log-SD scale).
`lower`	Lower bound for the overdispersion parameter.
`upper`	Upper bound for the overdispersion parameter.
`parscale`	Parameter scale for optimization.
`term`	A model term object (S4 methods).
`data`	A data frame containing the variables (S4 methods).

Details

Negative binomial observation model term

Value

A nbinom object.

Functions

- theta_info(nbinom): Theta info for the overdispersion parameter.

Slots (inherited from model)

ad_fun Character scalar "nbinom_obs".

ad_kind Character scalar "observations".

---

outer_optim_wrappers *Outer objective and gradient wrappers*

Description

Convenience wrappers around log_lik_laplace for use with outer optimizers that expect fn / gr callbacks. Both functions solve (or warm-start) the inner problem over gamma via log_lik_laplace and update a mutable cache environment with the latest inner solution.

Usage

```
outer_fn(x, config, cache, ad_fun, control_inner = list(), ...)

outer_gr(x, config, cache, ad_fun, control_inner = list(), ...)
```

Arguments

x	Numeric outer parameter vector c(beta, theta).
config	Configuration list passed to log_lik_laplace.
cache	An environment used to warm-start and store the inner gamma solution. If cache$gamma is missing or the wrong length, it is initialized from config$gamma (if present) or zeros of length ad_fun@sizes["gamma"]. Both functions update cache$gamma after each evaluation.
ad_fun	ad_fun object from ad_fun.
control_inner	A list of control options forwarded to the control argument of log_lik_laplace for the inner optimization.
...	Additional arguments forwarded to log_lik_laplace (for example data or package).

Details

outer_fn() Returns the scalar objective negative log likelihood.

outer_gr() Returns the gradient.

Value

- outer_fn: a numeric scalar neg_log_lik.
- outer_gr: gradient of neg_log_lik (same sign as outer_fn).

See Also

log_lik_laplace

Examples

```
## Not run:
cache <- new.env(parent = emptyenv())
cache$gamma <- rep(0, nrow(data$ATp))
ad_fun <- ad_fun(data, config)

val <- outer_fn(x = x0, data = data, config = config, cache = cache, ad_fun = ad_fun)
gr <- outer_gr(x = x0, data = data, config = config, cache = cache, ad_fun = ad_fun)

## End(Not run)
```

---

rmvnldl *Simulate from a multivariate normal using LDL of the precision matrix*

Description

Draws n samples from $N(\mu, H^{-1})$ where H is the precision matrix, given its LDL factorization (unit lower L1, diagonal D, and permutation perm).

Usage

```
rmvnldl(n, mean = 0, chol_prec)
```

Arguments

n	Number of samples.
mean	Mean vector μ (length length(D)).
chol_prec	Either a CHMfactor, Cholesky, or dCHMsimpl from Matrix::Cholesky(..., LDL = TRUE), or a list like res$hessian$chol_inner with components L1, D, and perm.

Details

Uses $H^{-1/2} = P^\top L^{-1} D^{-1/2}$ (same as reformat_chol() in log_lik_deriv), and rnorm only: $x = \mu + H^{-1/2} z$, $z \sim N(0, I)$.

Value

An n-by-p matrix; each row is one draw.

See Also

Cholesky, log_lik_laplace

`rpoly-class` *Random Polynomial Model Term*

Description

Creates and manages random polynomial model terms.

Usage

```
rpoly(x, p = 2, ref_value = 0, sd = Inf)

## S4 method for signature 'rpoly'
design(term, data)

## S4 method for signature 'rpoly'
precision(term, data)

## S4 method for signature 'rpoly'
random_info(term, data)
```

Arguments

`x`	Variable name.
`p`	Polynomial degree (default: 2).
`ref_value`	Reference value for the polynomial.
`sd`	Standard deviation for random effects.
`term`	A ‘rpoly‘ term object.
`data`	A data frame containing the variables used in the term.

Value

A ‘rpoly‘ term object.

A design matrix for the random polynomial term.

A precision matrix for the random polynomial term.

A data frame containing random effects information for the random polynomial term.

Functions

- `design(rpoly)`: Design method for rpoly objects
- `precision(rpoly)`: Precision method for rpoly objects
- `random_info(rpoly)`: Random info method for rpoly objects

Methods

The following methods are available for ‘rpoly‘ objects:

`design(term, data)` Creates design matrix for rpoly term

`precision(term, data)` Creates precision matrix for rpoly term

`theta_info(term)` Extracts theta parameter information

`beta_info(term, data)` Extracts beta parameter information

`random_info(term, data)` Extracts random effects information

`sim_fit` *Simulate linear predictors on a new covariate grid*

Description

Draws random effects from the Laplace approximation and evaluates the linear predictor on `x` by summing fixed (β) and random (γ) contributions from each model term.

Usage

```
sim_fit(x, data, fit, n = 500L)
```

Arguments

`x`	Data frame of prediction covariates (same variables as the fitted model). Only terms whose variables appear in `x` contribute.
`data`	Object returned by `model_data`.
`fit`	Result from `log_lik_laplace` at the fitted outer parameters.
`n`	Number of draws for random-effect simulation.

Value

The combined linear predictor: a `nrow(x)` by `n` matrix of simulated values (fixed-effect contributions are constant across columns).

`sizes` *Parameter-block row counts for an* `ad_data`

Description

Parameter-block row counts for an `ad_data`

Usage

```
sizes(x, ...)

## S4 method for signature 'ad_data'
sizes(x)
```

Arguments

`x`	An `ad_data` object.
`...`	Unused; for S4 generic compatibility.

Value

Named list with `n_beta`, `n_gamma`, and `n_theta`.

Index

∗ **datasets**
    .type_factor_levels, 3
    gamm, 19
    germany, 21
.type_factor_levels, 3

ad_data, 3
ad_data-class, 5
ad_fun, 5, *7–9*, *16*, *23*, *24*, *30*, *31*, *36*
ad_fun,ad_data-method *(*ad_fun*)*, 5
ad_fun,ad_fun_ptr-method *(*ad_fun*)*, 5
ad_fun,list-method *(*ad_fun*)*, 5
ad_fun-class, 6
ad_fun_ptr, 7, *8*
ad_fun_ptr-class, 8
ad_shards, 8, *9*
adlaplace, 9, *12*
adlaplace_cpp, 11
adlaplace_fit-methods, 12
apply_theta_log, 13

beta_info *(*model-generics*)*, 33
beta_info,fpoly-method *(*fpoly-class*)*, 18
beta_info,intercept-method
        *(*intercept-class*)*, 24
beta_info,linear-method *(*linear-class*)*,
        28
beta_info,model-method
        *(*model-generics*)*, 33
binomial *(*binomial-class*)*, 14
binomial-class, 14
by_group, 15
by_group-class, 15

c.ad_fun_ptr, *8*, 16, *16*
Cholesky, *38*
clone_ad_fun_ptr, *8*, 16, *16*
coef.adlaplace_fit
        *(*adlaplace_fit-methods*)*, 12
collect_terms, *9*, 17, *20*, *34*
confint.adlaplace_fit, *10*
confint.adlaplace_fit
        *(*adlaplace_fit-methods*)*, 12

dat *(*gamm*)*, 19
data_setup, *13*, *34*
data_setup *(*ad_data*)*, 3
design *(*model-generics*)*, 33
design,fpoly-method *(*fpoly-class*)*, 18
design,iid-method *(*iid-class*)*, 22
design,intercept-method
        *(*intercept-class*)*, 24
design,iwp-method *(*iwp-class*)*, 27
design,linear-method *(*linear-class*)*, 28
design,model-method *(*model-generics*)*, 33
design,rpoly-method *(*rpoly-class*)*, 38

elgm_matrix *(*model-generics*)*, 33
environment, *36*
extra_ad_fun *(*model-generics*)*, 33
extra_ad_fun,model-method
        *(*model-generics*)*, 33

fitted.adlaplace_fit
        *(*adlaplace_fit-methods*)*, 12
format_parameters, 17
fpoly *(*fpoly-class*)*, 18
fpoly-class, 18
fun_obj_fdfh *(*adlaplace_cpp*)*, 11

gamm, 19
gaussian *(*gaussian-class*)*, 20
gaussian-class, 20
germany, 21
grad *(*adlaplace_cpp*)*, 11

has_openmp, 22
hessian *(*adlaplace_cpp*)*, 11

iid *(*iid-class*)*, 22
iid-class, 22

inner_opt, 23, *31, 32*
intercept *(*intercept-class*)*, 24
intercept-class, 24
iwp, 26
iwp-class, 27

joint_log_dens, *31, 32*
joint_log_dens *(*adlaplace_cpp*)*, 11

laplace_half_H_inv, 28
linear *(*linear-class*)*, 28
linear-class, 28
log_lik_laplace, *10*, *28*, 30, *32*, *36–39*
logLik.adlaplace_fit
*(*adlaplace_fit-methods*)*, 12

mean_mvquad, 31
model-class, 32
model-generics, 33
model_data, *9, 10*, *33*, 34, *39*

nbinom *(*nbinom-class*)*, 35
nbinom-class, 35
nobs.adlaplace_fit
*(*adlaplace_fit-methods*)*, 12

optim, *9, 10*
outer_fn, *9*
outer_fn *(*outer_optim_wrappers*)*, 36
outer_gr, *9*
outer_gr *(*outer_optim_wrappers*)*, 36
outer_optim_wrappers, 36

plot.adlaplace_fit
*(*adlaplace_fit-methods*)*, 12
precision *(*model-generics*)*, 33
precision,iid-method *(*iid-class*)*, 22
precision,iwp-method *(*iwp-class*)*, 27
precision,model-method
*(*model-generics*)*, 33
precision,rpoly-method *(*rpoly-class*)*, 38
predict.adlaplace_fit, *10*
predict.adlaplace_fit
*(*adlaplace_fit-methods*)*, 12
print.adlaplace_fit
*(*adlaplace_fit-methods*)*, 12
print.summary.adlaplace_fit
*(*adlaplace_fit-methods*)*, 12

random_info *(*model-generics*)*, 33
random_info,iid-method *(*iid-class*)*, 22
random_info,iwp-method *(*iwp-class*)*, 27
random_info,model-method
*(*model-generics*)*, 33
random_info,rpoly-method *(*rpoly-class*)*,
38
rmvnldl, *28*, *32*, 37
rpoly *(*rpoly-class*)*, 38
rpoly-class, 38

sim_fit, *13*, 39
sizes, 40
sizes,ad_data-method *(*sizes*)*, 40
summary.adlaplace_fit, *10*
summary.adlaplace_fit
*(*adlaplace_fit-methods*)*, 12
svd, *8*

theta_info *(*model-generics*)*, 33
theta_info,gaussian-method
*(*gaussian-class*)*, 20
theta_info,iid-method *(*iid-class*)*, 22
theta_info,iwp-method *(*iwp-class*)*, 27
theta_info,model-method
*(*model-generics*)*, 33
theta_info,nbinom-method
*(*nbinom-class*)*, 35
trace_hinv_t *(*adlaplace_cpp*)*, 11

vcov.adlaplace_fit, *10*
vcov.adlaplace_fit
*(*adlaplace_fit-methods*)*, 12