Package ‘adlaplace’

October 26, 2026

Type Package

Title Laplace Approximations Using Automatic Differentiation

Version 0.7.5

Date 2026-10-14

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

Description Computes Laplace-approximate maximum likelihood for hierarchical and latent Gaussian models. A formula interface specifies observation densities (e.g. Gaussian, negative Binomial), fixed effects, and random terms (independent effects, smoothing terms). Customized response distributions and models for random effects can be specified through a low-level interface or by writing interoperable packages. Derivatives are computed using automatic differentiation with 'CppAD', and used in trust region optimization to find modes of random effects. Derivatives of the Laplace approximation of the log-likelihood are computed using third order contractions and is compatible with custom atomics. The approach follows Kristensen, Nielsen, Berg, Skaug, and Bell (2016) <doi:10.18637/jss.v070.i05>.

License GPL (>= 2)

Depends R (>= 3.5.0)

Imports Matrix (>= 1.2.18), Rcpp (>= 1.0.3), methods, stats, graphics, RCppAD (>= 1.20260000.0-2)

Suggests knitr, rmarkdown, trustOptim, testthat (>= 3.0.0), data.table, mvQuad, mgcv, MASS, nlme, sn, terra, GET, WoodburyMatrix, extraDistr, RSpectra, aghq, numDeriv

Config/testthat/edition 3

VignetteBuilder knitr

LinkingTo Rcpp, RcppEigen, trustOptim, Matrix, RCppAD (>= 1.20260000.0-2)

Encoding UTF-8

SystemRequirements OpenMP

NeedsCompilation yes

Collate 'RcppExports.R' 'classes.R' 'density_data.R' 'ad_pack_ptr.R' 'ad_pack.R' 'aghq_ff.R' 'obs_groups.R' 'model_data.R' 'log_lik.R' 'fit.R' 'grad_obs_units.R' 'amip.R' 'deriv_scan.R' 'dirichlet_multinom.R' 'effects.R' 'families.R' 'sim_fit.R' 'fit_methods.R' 'fixed_terms.R' 'format_parameters.R' 'germany-data.R' 'hessian_map.R' 'iid.R' 'iwp.R' 'london-data.R' 'mvn_utils.R' 'prepare_model_rows.R' 'prior.R' 'psd.R' 'replace_vars.R' 'rpoly.R' 'sim_iid.R' 'takahashi_davis.R' 'term_data_setup.R' 'zzz.R'

LazyData true

Config/roxygen2/version 8.1.0

Author Patrick Brown [aut, cre, cph]

Repository CRAN

Date/Publication 2026-10-26 19:30:10 UTC

Contents

.model_role_levels . . . 4
adlaplace . . . 4
adlaplace_cpp . . . 6
adlaplace_fit-methods . . . 8
ad_pack . . . 10
ad_pack-class . . . 12
ad_pack_drop . . . 12
ad_pack_ptr . . . 13
ad_pack_ptr-class . . . 14
ad_pack_set_X . . . 14
aghq_ff . . . 15
amip_set . . . 16
apply_theta_log . . . 16
binomial-class . . . 17
by_group . . . 18
by_group-class . . . 18
c.ad_pack_ptr . . . 19
clone_ad_pack_ptr . . . 19
collect_terms . . . 20
density_data-class . . . 20
deriv_scan_joint . . . 22
deriv_scan_laplace . . . 23
dirichlet_multinom . . . 25
dirichlet_multinom-class . . . 25
fit_outer . . . 27
format_parameters . . . 28
fpoly-class . . . 28
gamm . . . 30
gaussian-class . . . 30

germany . . . 32
grad_obs_units . . . 33
has_openmp . . . 34
hinv_trace_index . . . 34
iid-class . . . 35
influence_scores . . . 36
inner_opt . . . 37
intercept-class . . . 38
iwp . . . 40
iwp-class . . . 41
laplace_half_H_inv . . . 42
laplace_loglik . . . 43
linear-class . . . 44
log_lik_laplace . . . 45
london . . . 47
mean_mvquad . . . 48
model-generics . . . 49
model_data . . . 50
model_term-class . . . 51
nbinom-class . . . 52
n_groups . . . 53
n_obs_units . . . 54
obs_groups . . . 54
obs_groups_units . . . 55
outer_optim_wrappers . . . 56
phi_gamma . . . 57
poisson-class . . . 58
prior-class . . . 59
psd . . . 59
psd_prior . . . 60
replace_vars . . . 61
rmvnldl . . . 62
rpoly-class . . . 63
sim_fit . . . 64
sim_iid . . . 65
sim_random . . . 66
sizes . . . 67
takahashi_davis . . . 68
term_data_setup . . . 68

Index **70**

.model_role_levels *Valid model type factor levels*

Description

Character vector of allowed levels for the `model_role` slot on `model_term` objects.

Usage

```
.model_role_levels
```

Value

A character vector of length 3, with elements `"fixed"`, `"random"`, and `"response"` in that order. These are the allowed levels of the `model_role` factor on `model_term` objects, indicating a fixed effect, a random effect, or the response.

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 `obs_groups()`, build the AD handle with `ad_pack()`, 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(),
  control = list(),
  control_inner = list(maxit = 100L, report.level = 0, report.freq = 0),
  method = "L-BFGS-B",
  hessian = NULL,
  start = NULL,
  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, num_threads = 1L,num_shards = 100L)`. When `config$obs_groups` is missing it is filled by `obs_groups(A, num_shards = config$num_shards)`, or with `elgm_matrix` when the model data includes one (e.g. case-crossover). `config$num_threads` is the OpenMP thread count for the AD handle; `config$num_shards` is the target number of observation shards for `obs_groups()`.
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 `NULL` (default), set to `TRUE` if there are at most five outer parameters and `FALSE` otherwise. When `TRUE`, `optimHess` 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`.
start	Optional numeric vector of outer starting values `c(beta, theta)` on the *internal* (log-theta) scale, length `nrow(par_info)`. When `NULL` (default), term `init` values are used.
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_info Parameter table with `mle`, `se`, `mle_internal`, `se_internal`, bounds, and `log`.

gamma Named random-effect modes at the optimum.

nobs Number of observations.

details Laplace evaluation at the optimum, including `outer_opt`, `inner_opt`, `vcov`, `log_lik`, `gradient`, and `hessian`.

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

If `optim` throws (for example L-BFGS-B when `fn` is not finite), the same class is returned with `par_info` estimates set to `NA`, `details$outer_opt = NULL`, and `details$error` set to the condition message. `ad_pack` and `cache` are still present so the Laplace objective can be inspected or re-run without rebuilding tapes:

```
stats::optim(
  fit$cache$last_par_fn,
  fn = adlaplace::outer_fn, gr = adlaplace::outer_gr,
  method = fit$method, control = fit$control,
  lower = fit$par_info$lower, upper = fit$par_info$upper,
  config = fit$config, ad_pack = fit$ad_pack,
  cache = fit$cache, control_inner = fit$control_inner
)
```

See Also

summary.adlaplace_fit, predict.adlaplace_fit, log_lik_laplace, model_data, fit_outer

Examples

```
set.seed(1)
n <- 80
g <- factor(sample(4, n, replace = TRUE))
x <- rbinom(n, 1, 0.5)
eta <- 0.5 + 0.4 * x + rnorm(4, sd = 0.3)[g]
nb_sd <- 0.3
z <- rgamma(n, nb_sd^(-2), nb_sd^(-2))
dat <- data.frame(y = rpois(n, exp(eta) * z), x = x, g = g)
md <- model_data(
  nbinom(y, lower = 1e-9, init = 0.15) ~ x + iid(g, init = 0.3),
  data = dat
)
config <- list(
  transform_theta = TRUE,
  obs_groups = obs_groups(md$term_data$A, num_shards = 5),
  verbose = FALSE
)
ap <- ad_pack(md, config)
cache <- new.env(parent = emptyenv())
x0 <- md$term_data$info$parameters$init
outer_fn(x = x0, config = config, cache = cache, ad_pack = ap)
outer_gr(x = x0, config = config, cache = cache, ad_pack = ap)

fit <- adlaplace(md, config = list(num_shards = 5L, verbose = FALSE))
summary(fit)
confint(fit)
```

---

adlaplace_cpp *C++ backend entry points*

Description

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

Usage

```
joint_log_dens(ad_pack, x, ad_shards = NULL, negative = TRUE)

grad(ad_pack, x, ad_shards = NULL, inner = FALSE, negative = TRUE)

hessian(
  ad_pack,
  x,
  ad_shards = NULL,
  inner = FALSE,
  verbose = FALSE,
  negative = TRUE
)

trace_hinv_t(ad_pack, x, LinvPt, LinvPtColumns, verbose = FALSE, H_inv = NULL)

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

Arguments

`ad_pack`	An `ad_pack` S4 object or `ad_pack_ptr` handle.
`x`	Numeric parameter vector of length `Nparams`.
`ad_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()`.
`H_inv`	Optional numeric values of the joint inverse Hessian, in the nonzero order of `chol_inner_list$H_inv`. A sparse matrix contributes its `@x` slot. `NULL` keeps every shard on the column walk.
`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`.

Thread affinity

These are **serial debug** APIs. They error if the handle was built with `ad_pack(..., num_threads > 1)`. Use a plain `ad_pack_ptr` (or `ad_pack(..., num_threads = 1)`), or `clone_ad_pack_ptr()` before assigning multi-thread affinity. Parallel evaluation uses `fun_obj_fdfh` / `inner_opt` / `log_lik_laplace` on the affined handle.

`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$log = 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.

Value

`coef` Named numeric vector of outer parameter estimates. With `transform = TRUE` (the default), parameters stored on the log scale (for example standard deviations) are exponentiated to the natural scale.

`vcov` Variance-covariance matrix of the outer parameters on the optimization scale. With `transform = TRUE`, rows and columns for log-scale parameters are mapped to the natural scale by the delta method.

`logLik` An object of class `"logLik"`: the Laplace-approximate marginal log-likelihood, with attributes `df` (number of outer parameters) and `nobs`.

`nobs` Integer number of observations.

`confint` Numeric matrix of Wald intervals, one row per outer parameter. With `transform = TRUE`, endpoints for log-scale parameters are exponentiated to the natural scale.

`fitted` Numeric vector of length `nobs`: the linear predictor at the joint mode of the fixed effects and random effects.

`predict` Without `newdata`, the same vector as `fitted`. With `newdata`, a matrix with `nrow(newdata)` rows and `n` columns of linear predictors drawn from the Laplace approximation to the random effects.

`summary` A list of class `"summary.adlaplace_fit"` with the coefficient table (natural-scale estimates, standard errors, and interval endpoints), the log-likelihood, degrees of freedom, the number of observations and random effects, the outer optimizer convergence code, and the maximum absolute profile gradient.

`print`, `print.summary`, `plot` No return value used by the caller. They print or draw and return the input object invisibly. `plot` draws each smooth term as curves simulated from the Laplace approximation, with the pointwise mean highlighted, or the random-effect modes when the model has no smooth term.

---

ad_pack *Build AD function with Hessian templates*

Description

Build AD function with Hessian templates

Usage

```
ad_pack(
  x,
  config = NULL,
  num_threads = 1L,
  reorder_shards = c("none", "gradient", "hessian", "third"),
  templates = NULL,
  ...
)

## S4 method for signature 'ad_pack_ptr'
ad_pack(
  x,
  config = NULL,
  num_threads = 1L,
  reorder_shards = c("none", "gradient", "hessian", "third"),
  templates = NULL,
  ...
)

## S4 method for signature 'density_data'
ad_pack(
  x,
  config = NULL,
  num_threads = 1L,
  reorder_shards = c("none", "gradient", "hessian", "third"),
```

```
  templates = NULL,
  ...
)

## S4 method for signature 'list'
ad_pack(
  x,
  config = NULL,
  num_threads = 1L,
  reorder_shards = c("none", "gradient", "hessian", "third"),
  templates = NULL,
  ...
)
```

Arguments

x	One of: an ad_pack_ptr, an density_data shard with ad_kind/ad_pack set, or a model_data() bundle (list with observations, random, parameters).
config	Model configuration list. Required for density_data and model_data; unused for ad_pack_ptr. For model_data, beta, gamma, and theta are filled from x$term_data$info when omitted (only shards, transform_theta, etc. are needed). When config$num_threads is set, it overrides the num_threads argument for density_data and model_data methods. When config$reorder_shards is set, it overrides reorder_shards. config$hessian_sparsity (default TRUE) controls whether observation / parameter / random tapes discover and calibrate Hessian sparsity. Set FALSE for grad-only packs (usable with grad, not with inner_opt LDL / Hessian paths); see grad_obs_units. config$obs_hessian is "analytic" (default) or "discover". Analytic builds the observation Hessian pattern from design co-occurrence (and ELGM strata for Dirichlet-multinomial) instead of CppAD for_hes_sparsity. Use "discover" as an escape hatch. When config$obs_groups is missing and config$obs_units is set (0-based unit indices), observation shards are built via obs_groups_units (see also ad_pack_drop).
num_threads	Positive integer; OpenMP thread count for inner_opt and parallel trace_hinv_t. Default 1L (serial).
reorder_shards	Character; how to assign OpenMP owner threads when num_threads > 1 and there are multiple shards: "none" (default) uses owner_thread = shard_index % num_threads; "gradient" times each shard's outer grad(..., inner = FALSE) (mean of 2; no warmup), then longest-processing-time (LPT) assignment; "hessian" times each shard's outer hessian(..., inner = FALSE) (1 warmup + mean of 2), then LPT; "third" times a Reverse3 sweep with dummy directions matching the symbolic half_H_inv / trace_columns sparsity (no numeric Cholesky), then LPT. Physical shard order is unchanged; only owner_thread is rewritten.
templates	Optional existing ad_pack whose Hessian templates (and group sparsity) are reused when the new pack has identical beta/gamma/theta sizes and the same number of shards. Useful when rebuilding packs for modular inference over exposure samples.
...	For ad_pack_ptr input, optional additional ad_pack_ptr shards to combine before attaching templates.

Value

Object of class ad_pack.

---

ad_pack-class *AD function with Hessian templates attached*

Description

AD function with Hessian templates attached

Slots

`ptr` Raw combined handle (`ad_pack_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`) and `hinv_q_index` (per-shard slots of Q into `H_inv`, when a shard defines `hinv_trace_index`).

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

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

---

ad_pack_drop *Retape an AD pack after dropping observation units*

Description

Builds a new `ad_pack` on `fit$model_data` with observation shards filtered to the remaining units (same coarse sharding as `fit$config$obs_groups` when present). Random and parameter shards are retaped; reuse of the merged fit pack is not attempted.

Usage

```
ad_pack_drop(fit, drop, num_threads = NULL, verbose = FALSE)
```

Arguments

fit	An "adlaplace_fit" from adlaplace.
drop	Integer vector of **1-based** unit indices to remove.
num_threads	OpenMP thread count; default from fit$config.
verbose	Logical; forwarded into the retape config.

Value

An ad_pack usable with inner_opt at the original outer MLE.

See Also

obs_groups_units, influence_scores, amip_set

---

ad_pack_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@density. 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). For random_mult it must be list(Q, log_det, rank) where Q is a square dsCMatrix with uplo = "U" (upper triangle only); use adlaplace:::as_dsC_upper(Q) to coerce a dgCMatrix. Missing precision means the shard is not built (e.g. diffuse rpoly with sd = Inf via model_data()); calling ad_pack_ptr() without it is an error.

Usage

```
ad_pack_ptr(data, config = NULL, info = NULL)
```

Arguments

data	An density_data object with ad_kind and ad_pack 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$num_threads does not assign OpenMP threads; use num_threads on ad_pack after merging shards. When info is supplied, missing beta / theta / gamma / transform_theta are filled from it; explicit config entries win. Missing gamma becomes zeros of length nrow(info$gamma).
info	Optional model_data()$term_data$info list. When provided, tape seeds use info$beta$init and info$theta$init (the latter on the log scale where info$theta$log is TRUE); gamma defaults to zeros (info$gamma has no init column).

Details

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

Value

External pointer of class `ad_pack_ptr`.

---

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

Description

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

See Also

`ad_pack_ptr`, `ad_pack`, `clone_ad_pack_ptr`

---

ad_pack_set_X *Rebuild an* `ad_pack` *for a new model-data design*

Description

Covariate values in observation `XTp` are baked into CppAD tapes at build time, so they cannot be mutated in place. This helper retapes a new `model_data` bundle and reuses Hessian templates from `ad_pack` whenever sizes and shard counts match (identical sparsity across exposure samples). Prefer building a list of packs once for modular inference rather than calling this inside every likelihood evaluation.

Usage

```
ad_pack_set_X(
  ad_pack,
  model_data,
  config = list(),
  num_threads = 1L,
  reorder_shards = c("none", "gradient", "hessian", "third")
)
```

Arguments

ad_pack	Existing ad_pack whose templates may be reused.
model_data	A model_data() bundle with the same random-effect structure and (typically) a different fixed-effect design matrix.
config	Configuration list passed to ad_pack.
num_threads, reorder_shards	Forwarded to ad_pack.

Value

A new ad_pack.

See Also

ad_pack, replace_vars

---

aghq_ff — *Build* **aghq** *objective list from an adlaplace fit*

Description

Returns list(fn, gr, he) suitable for aghq, closing over the fitted AD handle and cache. The wrappers return the *negated* Laplace-approximate log posterior (same convention as outer_fn / outer_gr), so call aghq::default_control(negate = TRUE).

Usage

```
aghq_ff(fit, control_inner = NULL)
```

Arguments

fit	An "adlaplace_fit" from adlaplace.
control_inner	Inner trust-region control list; defaults to fit$control_inner when present, otherwise a quiet short run.

Value

A list with fn, gr, and he. The Hessian is a numerical Jacobian of gr (no AD outer Hessian is available).

See Also

outer_fn, outer_gr, adlaplace

amip_set — *Approximate most influential removal set (AMIP)*

Description

Greedy rule: drop units with the largest scores of the same sign as `phi` until $\sum_{u \in S} \psi_u$ matches the sign of ϕ and $|\sum| \geq |\phi|$. Leaving out S changes ϕ by approximately $-\sum \psi_S$.

Usage

```
amip_set(psi, phi)
```

Arguments

`psi`	Numeric vector of per-unit scores (e.g. from `influence_scores`).
`phi`	Scalar quantity of interest at the full-data mode.

Value

A list with `idx` (1-based indices to drop), `alpha` (fraction of units), and `approx_change`. If the linear approximation cannot flip the sign, `idx` is empty and `alpha` is `NA`.

See Also

`influence_scores`, `ad_pack_drop`

apply_theta_log — *Apply log transform to selected theta columns*

Description

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

Usage

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

Arguments

`theta_info`	Data frame from `info$theta` (see `term_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.

Usage

```
binom(x, size = NULL)
```

Arguments

`x`	Outcome variable name (counts of successes; 0/1 when Bernoulli).
`size`	Optional column name for the number of trials per observation.

Details

Binomial observation model term

Value

A `binomial` object.

Slots (inherited from `model`)

`ad_pack` 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_pack_ptr *Combine* ad_pack_ptr *handles*

Description

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

Usage

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

Arguments

x	An ad_pack_ptr.
...	Additional ad_pack_ptr objects.

Value

A combined ad_pack_ptr.

See Also

clone_ad_pack_ptr, ad_pack

clone_ad_pack_ptr *Deep copy of an* ad_pack_ptr *handle*

Description

Returns a new ad_pack_ptr with independent C++ state (CppAD tapes and sparsity patterns). Hessian templates are not copied; call ad_pack on the clone to attach maps. The source handle is left unchanged, unlike c.ad_pack_ptr which moves shards and clears the inputs. OpenMP thread affinity from a prior ad_pack(..., num_threads > 1) is cleared on the clone so joint_log_dens / grad / hessian remain available as serial debug APIs.

Usage

```
clone_ad_pack_ptr(x)
```

Arguments

x	An ad_pack_ptr object.

Value

A new ad_pack_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 (`binom(y, size = N)`).
`verbose`	print extra information

Value

List of model term objects

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

Description

Per-shard layout for AD density evaluation

Usage

```
density_data(
  y = missing(),
  A = NULL,
  X = NULL,
  beta_map = NULL,
  gamma_map = NULL,
  theta_map = NULL,
  elgm_matrix = NULL,
  density = NA_character_,
```

```
  ad_kind = NA_character_,
  package = NA_character_,
  precision = NULL,
  weights = numeric(0)
)
```

Arguments

y	Response vector (default empty), or an existing density_data object to recast as another shard.
A	Random-effects design matrix (any Matrix class). When elgm_matrix is missing, nrow(A) = length(y). When the response is indexed by ELGM rows (length(y) = nrow(elgm_matrix)), nrow(A) equals length(y) or ncol(elgm_matrix). When the response is aggregated on ELGM columns (length(y) = ncol(elgm_matrix)), nrow(A) equals nrow(elgm_matrix) (one design row per cell).
X	Fixed-effects design matrix (any Matrix class). Row alignment matches A.
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). Pattern matrices are coerced to ngCMatrix. Numeric matrices are stored as dgCMatrix, so entry weights are kept. Either the response is indexed by rows (nrow(elgm_matrix) = length(y); columns are strata or design rows) or, for an aggregated likelihood, rows are design cells and columns are aggregated units (ncol(elgm_matrix) = length(y)). If NULL, defaults to Matrix::Matrix(nrow = length(y), ncol = 0).
density	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). For density = "random_mult", a list with Q as dsCMatrix (uplo = "U"); dgCMatrix is coerced via as_dsC_upper().
weights	Optional per-observation weights (e.g. binomial trial counts). Empty means all ones.

Value

An S4 object of class "density_data": one automatic-differentiation shard. Slots hold the response, the transposed fixed- and random-effect design blocks, the maps from local design columns into the global parameter vectors, the registered density name, and the shard kind ("observations", "parameters", or "random").

Functions

- density_data(): Construct a density_data object

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 (CsparseMatrix). A pattern matrix (ngCMatrix) marks membership. A numeric dgCMatrix stores a weight on each entry, such as population in a cell-region intersection. Rows index either observations or design cells; columns index strata or aggregated units. Empty by default.

density 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.

---

deriv_scan_joint *Scan joint log-density AD derivatives vs finite differences*

Description

Evaluates the joint log density and its AD gradient/Hessian while varying one coordinate of the full parameter vector (beta, gamma, theta). Returns plain matrices shaped for matplot()/matlines().

Usage

```
deriv_scan_joint(
  ad_pack,
  x,
  index,
  width = 0.1,
  n = 11L,
  ad_shards = NULL,
  negative = FALSE
)
```

Arguments

ad_pack	An ad_pack object (typically with num_threads = 1).
x	Numeric full parameter vector.
index	Integer (1-based) coordinate of x to vary.
width	Half-width of the scan interval around x[index].
n	Integer number of grid points (default 11).
ad_shards	Optional 0-based shard indices passed to joint_log_dens / grad / hessian.
negative	Logical; passed through to the joint-density evaluators.

Value

A list with:

grid	Coordinate values (length n).
mid	Midpoints for finite differences (length n - 1).
value	Joint density along the grid.
fd_value	Midpoint differences of value.
grad	n x length(x) matrix of AD gradients (rows = grid).
fd_grad	(n-1) x length(x) midpoint slopes of grad.
hess_row	n x length(x) AD Hessian rows for index.

---

deriv_scan_laplace *Scan Laplace profile AD derivatives vs finite differences*

Description

Evaluates log_lik_laplace(deriv = TRUE) while varying one coordinate of the outer parameter vector (beta, theta only; gammas are profiled out). Returns plain matrices shaped for matplot()/matlines().

Usage

```
deriv_scan_laplace(
  ad_pack,
  x,
  index,
  width = 0.5,
  n = 7L,
  config = list(),
  gamma = NULL,
  control = list(maxit = 100L, report.level = 0, report.freq = 0)
)
```

Arguments

ad_pack	An ad_pack object.
x	Numeric outer parameter vector (length beta + theta).
index	Integer (1-based) coordinate of x to vary.
width	Half-width of the scan interval around x[index].
n	Integer number of grid points (default 7).
config	Config list passed to log_lik_laplace.
gamma	Optional gamma start for the inner optimization.
control	Inner-optimization control list.

Value

A list with:

grid	Coordinate values (length n).
mid	Midpoints for finite differences.
value	Negative log likelihood along the grid.
fd_value	Midpoint differences of value.
grad	n x length(x) AD profile gradients.
fd_grad	Midpoint slopes of grad.
half_log_det	Half log-determinant along the grid.
d_det	AD derivative of the half log-determinant wrt outer params.
fd_det	Midpoint differences of half_log_det.
u_hat	n x n_gamma inner modes.
dU_index	n x n_gamma AD dU slices for index.
fd_u	Midpoint slopes of u_hat.
mid_result	Full log_lik_laplace result at the middle grid point.

dirichlet_multinom *Dirichlet-multinomial case-crossover response term*

Description

Dirichlet-multinomial case-crossover response term

Usage

```
dirichlet_multinom(
  x,
  by,
  init = 0.001,
  lower = 0,
  upper = Inf,
  parscale = 1,
  log = TRUE
)
```

Arguments

x	Outcome variable name.
by	Character vector of stratification variables (e.g. c("year", "month", "dow")).
init	Initial value for overdispersion sd.
lower	Lower bound for sd.
upper	Upper bound for sd.
parscale	Parameter scale for optimization.
log	Whether theta is optimized on the log scale.

Value

A dirichlet_multinom object.

dirichlet_multinom-class *Dirichlet-multinomial response term*

Description

Model term for the case-crossover Dirichlet-multinomial log density registered as dirichlet_multinomial.

Usage

```
## S4 method for signature 'dirichlet_multinom'
elgm_matrix(term, data)

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

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

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

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

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

Arguments

`term`	A `dirichlet_multinom` term object.
`data`	Analysis-ready data frame.

Details

Dirichlet-multinomial case-crossover response term

Value

Sparse `ngCMatrix` mapping observations to strata columns.

Functions

- `elgm_matrix(dirichlet_multinom)`: Case-crossover stratum map for AD shards.
- `design(dirichlet_multinom)`: Design method (not used).
- `precision(dirichlet_multinom)`: Precision method (not used).
- `theta_info(dirichlet_multinom)`: Theta info for overdispersion `sd`.
- `beta_info(dirichlet_multinom)`: Beta info method (not used).
- `random_info(dirichlet_multinom)`: Random info method (not used).

Slots

`by` Character vector of stratification variables for the case-crossover design.

Slots (inherited from `model`)

`ad_pack` Character scalar `"dirichlet_multinomial"`.

`ad_kind` Character scalar `"observations"`.

`package` Character scalar `"adlaplace"`.

`fit_outer` *Outer optimization scaffolding for Laplace profile likelihoods*

Description

Runs `optim` on a user-supplied outer objective / gradient (typically `outer_fn` / `outer_gr`, or a modular average of several packs) and returns the standard `par_info` / `vcov` / `opt` structure used by `adlaplace`.

Usage

```
fit_outer(
  fn,
  gr,
  par_meta,
  method = "L-BFGS-B",
  control = list(),
  hessian = FALSE,
  start = NULL,
  ...
)
```

Arguments

`fn`, `gr`	Objective and gradient functions with signature `function(par, ...)`.
`par_meta`	Data frame of outer-parameter metadata with columns `label`, `init`, `lower`, `upper`, `parscale`, `log` (as produced by `model_data()$term_data$info$parameters`).
`method`	`optim` method.
`control`	Control list for `optim`.
`hessian`	Logical; compute a numerical outer Hessian via `optimHess` when `TRUE`.
`start`	Optional starting values on the internal scale; defaults to `par_meta$init`.
`...`	Extra arguments forwarded to `fn` / `gr` / `optimHess` (e.g. `config`, `ad_pack`, `cache`, `control_inner`).

Value

A list with components `opt`, `par_info`, `vcov`, and `error` (character message when `optim` fails, otherwise `NULL`).

See Also

`adlaplace`, `outer_fn`

format_parameters *Format Laplace estimates using parameter metadata*

Description

Maps flat Laplace parameter vectors onto labeled tables from `info` (as stored on `ad_pack@info` or `model_data()$term_data$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,
  basis_scale = 1
)

## 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.
basis_scale	Positive scale for the covariate. Column k of the raw polynomial is divided by basis_scale^k. iwp passes the mean knot spacing when the polynomial is the fixed part of an IWP.
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 `normal(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).

Usage

```
normal(
  x,
  init = 1,
  lower = .my_theta_lower,
  upper = .my_theta_upper,
  parscale = .my_theta_parscale,
  log = TRUE
)

## 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.
`log`	Whether theta is optimized on the log scale.
`term`	A model term object (S4 methods).

Details

Gaussian observation model term

Value

A `gaussian` object.

Functions

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

Slots (inherited from `model`)

`ad_pack` 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 $y_i \sim \mathrm{Poisson}(E_i e^{\eta_i})$ 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 (`dsCMatrix`, `uplo = "U"`).

prec List passed to `density = "random_mult"`: `Q` (`dsCMatrix`, `uplo = "U"`), `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")` in ETRS89 / LCC Germany (`EPSG:4839`) 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.

grad_obs_units *Per-unit observation gradients (streaming, grad-only tapes)*

Description

Computes sparse gradients of each observation **unit** contribution at a fixed full parameter vector x, without re-running inner_opt. Each unit is taped temporarily with config$hessian_sparsity = FALSE (sparse Jac only), reverse-mode differentiated, then discarded.

Usage

```
grad_obs_units(
  data,
  x,
  config = list(),
  units = NULL,
  batch_size = 32L,
  inner = TRUE,
  negative = TRUE
)
```

Arguments

data	An observation density_data (e.g. fit$model_data$observations[[1]]), or a model_data() bundle (uses the first observation shard).
x	Numeric full parameter vector c(beta, gamma, theta) at the evaluation point (typically inner_opt()$full_parameters).
config	Config list passed to the observation tape builder. Always forced to hessian_sparsity = FALSE; compact_tape defaults to TRUE and transform_theta defaults to TRUE when omitted (same as adlaplace / ad_pack). Supply beta/gamma/theta seeds if needed for recording (taken from x when omitted). Prefer passing fit$config (or at least its transform_theta) when evaluating at an MLE from adlaplace.
units	Optional integer vector of **0-based** unit indices; default all units.
batch_size	Positive integer; how many one-unit tapes to hold at once.
inner	Logical; if TRUE (default), return only the γ block (rows = n_gamma).
negative	Logical; if TRUE (default), return the negative log density gradient (same sign as grad / inner_opt).

Details

Units are ELGM stratum columns when data@elgm_matrix has columns; otherwise units are rows of the observation vector y.

Value

A dgCMatrix of size n_gamma x length(units) when inner = TRUE, or n_full x length(units) when inner = FALSE. Column j is the gradient for units[j].

Config note

hessian_sparsity = FALSE yields packs usable for grad(..., inner = TRUE) but not for Hessian / inner_opt LDL paths.

See Also

grad, inner_opt, ad_pack

---

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.

---

hinv_trace_index — *Positions of a shard's quadratic form inside the joint inverse Hessian*

Description

Called from ad_pack() after the symbolic H_inv pattern exists. The default method returns NULL, which leaves that shard on the trace_hinv_t column walk. A backend method may return an integer vector of 0-based slots into H_inv@x, one per stored entry of the shard's precision, with -1L where the pair is absent from the pattern.

Usage

```
hinv_trace_index(shard, H_inv)

## S4 method for signature 'density_data'
hinv_trace_index(shard, H_inv)
```

Arguments

shard	A density_data shard.
H_inv	Symbolic upper triangle of the joint inverse Hessian.

Value

An integer vector, or NULL.

---

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,
  log = TRUE
)

## 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.

log	Whether theta is optimized on the log scale.
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

---

influence_scores *Per-unit influence scores for an inner linear functional*

Description

Freezes outer parameters at the MLE, re-optimizes γ, forms $v = H^{-1}a$ from the fit-pack inner Hessian, and streams one-unit observation gradients via grad_obs_units (temporary tapes). Scores are $\psi_u = -v^\top g_u$.

Usage

```
influence_scores(
  fit,
  a,
  batch_size = 32L,
  control = list(report.level = 0, report.freq = 0),
  verbose = FALSE
)
```

Arguments

fit	An "adlaplace_fit" from adlaplace.
a	Named or positional numeric vector of length n_gamma (e.g. from phi_gamma).
batch_size	Passed to grad_obs_units.
control	Control list for inner_opt.
verbose	Logical; forwarded to inner_opt.

Value

A list with phi, gamma_hat, H, v, G, psi, x_full, and n_units.

See Also

phi_gamma, amip_set, grad_obs_units, ad_pack_drop

---

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 prebuilt ad_pack handle.

Usage

```
inner_opt(
  parameters,
  gamma,
  ad_pack,
  control = NULL,
  deriv = FALSE,
  verbose = FALSE,
  return_hessians = TRUE
)
```

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_pack	ad_pack S4 object from ad_pack.
control	List of trust-region control parameters (see **trustOptim**).
deriv	Logical; if TRUE, also return outer gradient/Hessian pieces, Cholesky-based quantities, and the profile gradient at the inner mode.
verbose	Logical; if TRUE, print thread/shard diagnostics.
return_hessians	Logical; if FALSE and deriv=TRUE, skip converting sparse Hessians / inverses to R Matrix objects and return only the scalar objective, inner_opt$solution, and deriv$d_neg_log_lik (optimizer hot path).

Details

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

Value

A list with `log_lik`, `neg_log_lik`, `parameters`, `full_parameters`, `inner_opt`, `gradient` (list `inner/outer`), and `hessian` (list `inner/outer/chol_inner/half_log_det`; with `deriv=TRUE` and `return_hessians=TRUE` also `half_H_inv`, `H_inv`, `trace3`). With `deriv=TRUE`, also `deriv` (profile pieces including `d_neg_log_lik`) and `dU`. Objective and derivatives use the negative log-density convention.

`intercept-class` *intercept Model Term*

Description

Creates and manages intercept model terms for fixed effects. With a finite `sd`, the intercept is treated as a random effect with a known-SD Gaussian prior (`N(0, sd^2)`), using `random_diagonal` without a theta parameter.

Usage

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

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

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

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

## S4 method for signature 'intercept'
random_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)

`sd`	Prior standard deviation. `Inf` (default) keeps a fixed effect; a finite positive value places a known-SD Gaussian prior and moves the intercept into the random-effect block.
`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
- `precision(intercept)`: Precision for known-SD intercept prior
- `random_info(intercept)`: Random info for known-SD intercept prior

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. The basis is divided by the mean knot spacing to the power p, and the penalty weights are the knot spacings divided by that mean spacing. Design entries and the standard-deviation parameter are then unchanged if x and the knots are rescaled by the same factor.

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,
  log = TRUE,
  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.
log	Whether theta is optimized on the log scale.
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.

Predictive SD

The standard-deviation hyperparameter is on knot-spacing units. Use [psd()] for the predictive SD $\sigma\sqrt{(h/s)^{2p-1}/((2p-1)((p-1)!)^2)}$ with mean spacing s, and [psd_prior()] for a PC prior on that scale.

Examples

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

---

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

Value

`design` Numeric design matrix for the integrated Wiener process basis, with one column per basis function and one row per observation. Entries are the basis evaluated at `data`.

`precision` Sparse precision matrix for those basis coefficients. The penalty is the integrated Wiener process prior.

`theta_info` One-row data frame describing the standard-deviation parameter: label, initial value, bounds, parameter scale, and whether optimization is on the log scale.

`random_info` Data frame with one row per random coefficient, including `gamma_label`, the name of that coefficient in a fitted model.

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 `hessian` layout).

Value

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

See Also

`log_lik_laplace`, `rmvnldl`

laplace_loglik — *Laplace-approximate log-likelihood (positive convention)*

Description

Thin wrapper around `log_lik_laplace` that returns the *positive* profile log-likelihood and its gradient with respect to the outer parameters `c(beta, theta)`. Prefer this over reading `inner_opt(...)$deriv$d_neg_log_lik` when building custom outer objectives (e.g. modular / multi-pack likelihoods).

Usage

```
laplace_loglik(
  ad_pack,
  par,
  gamma,
  config = list(),
  control_inner = list(),
  deriv = TRUE
)
```

Arguments

`ad_pack`	An `ad_pack` object.
`par`	Numeric outer parameter vector `c(beta, theta)` on the internal (log-theta) scale.
`gamma`	Numeric vector of random-effect modes used as the inner start.
`config`	Configuration list (may be empty).
`control_inner`	Control list for the inner trust-region solve.
`deriv`	Logical; if `TRUE` (default), return the profile gradient.

Value

A list with components

log_lik Scalar Laplace-approximate marginal log-likelihood.

gradient Numeric vector $\nabla_\psi \ell$, or `NULL` when `deriv = FALSE`.

gamma Inner modes at the optimum.

See Also

`log_lik_laplace`, `inner_opt`, `outer_fn`

linear-class *Linear Model Term*

Description

Creates and manages linear model terms for fixed effects. With a finite `sd`, coefficients are random effects with a known-SD Gaussian prior (`N(0, sd^2)`).

Usage

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

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

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

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

## S4 method for signature 'linear'
random_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)
`sd`	Prior standard deviation. `Inf` (default) keeps a fixed effect; a finite positive value places a known-SD Gaussian prior and moves the coefficients into the random-effect block.
`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
- `precision(linear)`: Precision for known-SD linear prior
- `random_info(linear)`: Random info for known-SD linear prior

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_pack,
  data,
  package = c(config[["package"]], "adlaplace")[1L],
  deriv = FALSE,
  return_hessians = TRUE
)
```

Arguments

x	Numeric vector of outer parameters (beta, then theta). Length must equal num_beta + num_theta from ad_pack@sizes.
config	A list of backend options (verbose, package, etc.). When ad_pack is missing, must also include beta, gamma, and theta so ad_pack() 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_pack@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_pack	Optional ad_pack object from ad_pack. 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_pack when ad_pack is not supplied.
package	Character scalar naming the backend package to use for ad_pack() and inner_opt(). Defaults to "adlaplace". Other backends must export a compatible ad_pack() builder and inner_opt().
deriv	Logical scalar. If TRUE, include derivative quantities in the output (gradient, intermediate derivatives).
return_hessians	Logical. Forwarded to inner_opt. When FALSE with deriv=TRUE, skip Matrix Hessian objects (optimizer path). Default TRUE for diagnostic / post-fit evaluations.

Details

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

When deriv=FALSE, the return value mirrors inner_opt() with inner_opt, top-level gradient, and hessian. When deriv=TRUE, the profile gradient is assembled in C++ inside inner_opt; the same list is augmented with deriv and profile-derivative pieces in gradient$outer.

Value

A list. With `deriv=FALSE`, same structure as `inner_opt` (without `parameters`). With `deriv=TRUE`, additionally:

deriv Data frame of profile derivative pieces.

gradient$outer$dU, gradient$outer$trace3 Intermediate profile objects.

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_pack`, `inner_opt`

`london` *London daily mortality and PM10 (case-crossover example)*

Description

Daily all-cause mortality and air-quality covariates for London, 2002–2006, from the case-crossover example of Tobias et al. (2024).

Usage

```
london
```

Format

A data frame with 1826 rows and columns:

city City label (`"London"`).

date Calendar date.

year, month, day Integer calendar components.

dow Day of week (integer encoding from the source file).

all Daily all-cause death count.

tmean Mean temperature (degrees Celsius).

rh Relative humidity (percent).

pm10 PM10 concentration.

Source

https://github.com/aureliotobias/casecrossover

References

Tobias, A., et al. (2024). Case-crossover designs for air pollution epidemiology (example data).

Examples

```
data("london", package = "adlaplace")
head(london)
```

`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_pack,
  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. `laplace$inner_opt$solution`).
`cov`	H^{-1} for the inner block (e.g. `laplace$hessian$H_inv`).
`ad_pack`	`ad_pack` 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-generics *Model Term Generics*

Description

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

Default reads data[[term@name]]. Aggregated terms override this when the counts are not one row of data (district totals on cell-level designs). NULL means the column is absent.

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)

observation_response(term, data)

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

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

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

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

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

## S4 method for signature 'model_term'
observation_response(term, data)

extra_density(term)

## S4 method for signature 'model_term'
extra_density(term)
```

Arguments

`term`	A model term object.
`data`	A data frame.

Value

Method-specific return values.

A numeric vector, or `NULL`.

Character density name, or `NULL`.

`model_data` *Assemble model terms and per-shard* `density_data` *objects*

Description

Parses a formula, builds design matrices, and returns formula terms plus `density_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 `NA`s 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 `term_data_setup`, including `elgm_matrix` when an observation term defines an `elgm_matrix` method.

`observations` Named list of observation `density_data` shards.

`parameters` Named list of parameter `density_data` shards.

`random` Named list of random-effect `density_data` shards.

Examples

```
set.seed(1)
dat <- data.frame(
  y = rpois(30, 2),
  x1 = rnorm(30),
  x2 = runif(30)
)
md <- model_data(
  y ~ x1 + iwp(x2, p = 2, knots = seq(0, 1, length.out = 5)),
  data = dat
)
names(md$terms)
```

`model_term-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

`name` 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

`log` Logical; optimize/tape each theta on the log scale (recycled to the number of thetas for multi-parameter terms).

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

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

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

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

`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,
  log = TRUE
)

## 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.
`log`	Whether theta is optimized on the log scale.
`term`	A model term object (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_pack` Character scalar `"nbinom_obs"`.

`ad_kind` Character scalar `"observations"`.

`n_groups` *Number of AD shards in an* `ad_pack_ptr` *handle*

Description

Number of AD shards in an `ad_pack_ptr` handle

Usage

```
n_groups(handle)
```

Arguments

`handle`	External pointer of class `ad_pack_ptr`.

Value

Integer count of groups (shards).

n_obs_units — *Number of removable observation units*

Description

Units are ELGM stratum columns when `elgm_matrix` has columns; otherwise rows of the observation vector `y`. Same domain as `grad_obs_units`.

Usage

```
n_obs_units(data)
```

Arguments

`data`	An observation `density_data`, a `model_data()` bundle, or an `"adlaplace_fit"`.

Value

Integer length-1 count of units.

obs_groups — *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.

Usage

```
obs_groups(A, elgm_matrix, num_shards, min_shards = 0)
```

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_shards`	Integer giving the minimum number of shards. When there are fewer distinct singular-vector loadings than `min_shards`, only the largest exact-loading group is split until `min_shards` is reached (or that group cannot be split further).

Details

The resulting grouping is returned as a pattern sparse matrix whose columns correspond to shards and whose nonzeros mark membership. Loadings are used only to form the partition. 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 `"ngCMatrix"` (from **Matrix**), with one column per shard and one row per observation (or ELGM stratum). Nonzero entries mark shard membership; there is no `@x` slot.

Examples

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

`obs_groups_units` *Build* `obs_groups` *for a subset of observation units*

Description

Restricts which observation units are recorded on observation tapes. Domain indices are **0-based** (ELGM stratum columns, or rows of `y`).

Usage

```
obs_groups_units(
  n_domain,
  units = NULL,
  grouping = c("identity", "together", "filter"),
  obs_groups = NULL
)
```

Arguments

`n_domain`	Number of units in the full domain.
`units`	Integer vector of 0-based unit indices to keep. Default all.
`grouping`	One of: `"identity"` One shard column per selected unit (scores). `"together"` One shard containing all selected units. `"filter"` Start from `obs_groups`, drop unselected rows, drop empty columns (keeps coarse fit sharding).
`obs_groups`	Existing shard map; required when `grouping = "filter"`.

Value

A ngCMatrix suitable for config$obs_groups.

See Also

n_obs_units, grad_obs_units, ad_pack_drop

---

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_pack, control_inner = list(), ...)

outer_gr(x, config, cache, ad_pack, 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_pack@sizes["gamma"]. Both functions update cache$gamma after each evaluation, and share cache$fg_x / cache$neg_log_lik / cache$d_neg_log_lik for the latest combined objective/gradient evaluation. For multi-pack / modular objectives, use one cache environment per ad_pack; cache$gamma is that pack's inner warm start.
ad_pack	ad_pack object from ad_pack.
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

L-BFGS-B requests fn then gr at the same parameter vector. Both wrappers share one inner_opt(..., deriv = TRUE, return_hessians = FALSE) evaluation at that point: the objective is free once the profile gradient has been computed. Cached numerics are stored in cache$fg_x, cache$neg_log_lik, and cache$d_neg_log_lik and reused until x changes.

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

```
set.seed(1)
n <- 80
g <- factor(sample(4, n, replace = TRUE))
x <- rbinom(n, 1, 0.5)
eta <- 0.5 + 0.4 * x + rnorm(4, sd = 0.3)[g]
nb_sd <- 0.3
z <- rgamma(n, nb_sd^(-2), nb_sd^(-2))
dat <- data.frame(y = rpois(n, exp(eta) * z), x = x, g = g)
md <- model_data(
  nbinom(y, lower = 1e-9, init = 0.15) ~ x + iid(g, init = 0.3),
  data = dat
)
config <- list(
  transform_theta = TRUE,
  obs_groups = obs_groups(md$term_data$A, num_shards = 5),
  verbose = FALSE
)
ap <- ad_pack(md, config)
cache <- new.env(parent = emptyenv())
x0 <- md$term_data$info$parameters$init
outer_fn(x = x0, config = config, cache = cache, ad_pack = ap)
outer_gr(x = x0, config = config, cache = cache, ad_pack = ap)
```

---

phi_gamma *Linear functional of the inner field γ*

Description

Builds a named numeric vector a aligned to fit$model_data$term_data$info$gamma$gamma_label so that $\phi(\gamma) = a^\top \gamma$.

Usage

```
phi_gamma(fit, term = NULL, new_x = NULL, coef = NULL)
```

Arguments

fit	An "adlaplace_fit" from adlaplace.
term	Optional character name of a covariate whose **random** terms contribute via design (e.g. "pm10").
new_x	Data frame of prediction points for term (required when term is set). Typically one row, e.g. data.frame(pm10 = 80).
coef	Optional gamma label (or labels) to set to 1 (unit / contrast vectors), e.g. a treatment coefficient name.

Value

Named numeric vector of length n_gamma.

See Also

influence_scores, amip_set, design, sim_random

---

poisson-class *Poisson observation term*

Description

Model term for the observation-level Poisson log density with a log link, registered as poisson_obs. For each observation with linear predictor eta = X beta + A gamma and optional offset o (via config$offset), the mean is exp(eta + o) and the log density contribution is y * eta - exp(eta + o) + y * o - lgamma(y + 1). There are no observation-level hyperparameters.

Creates a response term wired to the poisson_obs AD shard. There are no observation-level hyperparameters. An exposure or expected-count offset is supplied through config$offset when building AD shards (for example log(E) in disease mapping).

Usage

```
pois(x)
```

Arguments

x	Outcome variable name (non-negative counts).

Details

Poisson observation model term

Value

A poisson object.

Slots (inherited from `model`)

`ad_pack` Character scalar `"poisson_obs"`.

`ad_kind` Character scalar `"observations"`.

`prior-class` *Hyperparameter prior term*

Description

Places a prior on an existing theta (SD) parameter. Does not create a new theta; it emits an `ad_kind = "parameters"` shard that contributes to the joint log density.

Usage

```
prior(theta = 0L, dist = "exp", median = 1, rate = NULL)
```

Arguments

`theta`	Zero-based index into `info$theta$id`, or a character theta label (e.g. `"ID_iid"`).
`dist`	Prior family. Currently only `"exp"` (exponential on the SD scale) is supported.
`median`	Prior median on the SD scale. For `dist = "exp"`, the rate is `log(2) / median`.
`rate`	Optional exponential rate (overrides `median` when supplied).

Details

Hyperparameter prior term

Value

A `prior` term object.

`psd` *Predictive standard deviation of a random term*

Description

Generic for the unit-scale predictive SD $\sigma_x(h) = \mathrm{SD}\big[f(x+h) \mid f(x), f'(x), \ldots, f^{(p-1)}(x)\big]$ of a random model term. Returns ‘sd‘ times the unit PSD; the default ‘sd = 1‘ gives the prior-conversion factor used by [psd_prior()].

Usage

```
psd(term, h, x = NULL, sd = 1, ...)

## S4 method for signature 'model_term'
psd(term, h, x = NULL, sd = 1, ...)

## S4 method for signature 'list'
psd(term, h, x = NULL, sd = 1, ...)

## S4 method for signature 'iwp'
psd(term, h, x = NULL, sd = 1, ...)
```

Arguments

`term`	A [model_term-class] (or a list returned by a constructor such as [iwp()] / ‘mgp()‘).
`h`	Numeric step size(s). Recycled against ‘x‘ when both are vectors.
`x`	Optional evaluation location(s). Ignored by stationary-increment models such as [iwp()].
`sd`	Positive scale for the term’s SD hyperparameter (default 1).
`...`	Passed to methods.

Value

Numeric vector of predictive SDs, same length as the recycled ‘(h, x)‘ grid.

See Also

[psd_prior()], [iwp()]

---

psd_prior *PC prior on the predictive SD of a random term*

Description

Builds an exponential PC prior $P\big(\sigma_x(h) > u\big) = \alpha$ on the predictive SD of ‘term‘, returning a [prior()] term with rate $-\log(\alpha)\,\mathrm{psd}(\mathrm{term}, h, x, \mathrm{sd} = 1)/u$.

Usage

```
psd_prior(term, u, alpha = 0.5, h, x = NULL)
```

Arguments

`term`	A term with a [psd()] method, or a list returned by [iwp()] / ‘mgp()‘ (the element that carries a theta is selected).
`u`	Positive threshold on the predictive-SD scale.
`alpha`	Tail probability (default 0.5).
`h`	Positive step size for the predictive SD.
`x`	Optional location passed to [psd()] (required by some nonstationary terms such as ‘mgp‘).

Value

A [prior()] term targeting the SD theta of ‘term‘.

See Also

[psd()], [prior()]

`replace_vars` *Replace data-column names in model terms*

Description

Rewrites the data-column references on model terms (and formulas / term lists) according to a named character map. Used by modular-inference workflows that share one model structure across exposure sample columns (e.g. `sqrt_pm` -> `sqrt_pm_s3`).

Usage

```
replace_vars(x, map)

## S4 method for signature 'model_term'
replace_vars(x, map)

## S4 method for signature 'dirichlet_multinom'
replace_vars(x, map)

## S4 method for signature 'formula'
replace_vars(x, map)

## S4 method for signature 'list'
replace_vars(x, map)
```

Arguments

`x`	A `model_term`, `dirichlet_multinom`, formula, or list of such objects.
`map`	Named character vector: names are current column names, values are replacement column names. Names absent from `map` are left unchanged.

Value

An object of the same type as x, with column references updated and labels / formulas regenerated where applicable.

Examples

```
terms <- collect_terms(nbinom(y) ~ fpoly(sqrt_pm, p = 1))
replace_vars(terms, c(sqrt_pm = "sqrt_pm_s1"))
```

---

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, fit)
```

Arguments

n	Number of samples.
mean	Mean vector μ (length length(D)). When mean is named, those names are attached to the draws. Ignored when fit is supplied.
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. Ignored when fit is supplied.
fit	Optional "adlaplace_fit" object. When supplied, overrides mean and chol_prec with fit$gamma and fit$details$hessian$chol_inner.

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. When n = 1, a length-p vector. Column / element names come from names(mean) when present and of length p.

See Also

Cholesky, log_lik_laplace, adlaplace

rpoly-class *Random Polynomial Model Term*

Description

Creates and manages random polynomial model terms.

Usage

```
rpoly(x, p = 2, ref_value = 0, sd = Inf, basis_scale = 1)

## 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.
`basis_scale`	Positive scale for the covariate. Column `k` of the raw polynomial is divided by `basis_scale^k`. The default `1` leaves the raw polynomial unchanged. `iwp` passes the mean knot spacing so the polynomial matches the normalized IWP basis.
`term`	A ‘rpoly‘ term object.
`data`	A data frame containing the variables used in the term.

Details

Random Polynomial Model 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).

sim_iid *Simulate IID random effects with pointwise and global intervals*

Description

Identifies IID components in an adlaplace fit, forms their marginal Laplace covariance (Woodbury identity when mixed with other random effects), draws samples, and summarizes pointwise quantiles. When **GET** is available, also returns a global central region across levels.

Usage

```
sim_iid(fit, n = 500L, probs = c(0.1, 0.5, 0.9), coverage = 0.8, ...)
```

Arguments

fit	An "adlaplace_fit" from adlaplace.
n	Number of draws.
probs	Quantile levels for pointwise intervals.
coverage	Coverage for central_region global envelopes (used only when **GET** is installed).
...	Currently unused.

Details

IID rows are those with model == "iid" in fit$model_data$term_data$info$gamma. The marginal covariance uses **WoodburyMatrix** when installed; otherwise the IID block of fit$details$hessian$H_inv (if present) or a dense partitioned solve.

Value

A named list with one element per IID term. Each element is a list with x (basis / factor levels), sim (levels-by-draws matrix), pointwise (quantile matrix with columns named by probs), and global (matrix with columns lo, central, hi, or NULL if **GET** is unavailable). Attributes var_iid, iid_index, and iid_labels store the joint marginal covariance and index metadata.

See Also

sim_random, rmvnldl, iid

sim_random *Simulate a random-effect contribution for one covariate*

Description

Builds a prediction grid for variable x, forms the design matrix for all random terms on that variable, and multiplies by Laplace draws of the random effects. Intended for plotting smooths (iwp / rpoly) and similar random contributions from an adlaplace fit.

Usage

```
sim_random(
  x,
  fit,
  new_x,
  gamma_sims,
  num_grid = 101L,
  num_sim = 100L,
  probs = c(0.1, 0.5, 0.9),
  coverage = 0.8,
  transform = identity
)
```

Arguments

x	Character scalar: name of the covariate / grouping variable.
fit	An "adlaplace_fit" from adlaplace.
new_x	Optional one-column data frame of prediction points for x. When missing, a grid is built from term knots via pretty (or from unique basis levels in fit$model_data$term_data$info$gamma when there are no knots, e.g. IID).
gamma_sims	Optional matrix of random-effect draws (rows = draws, columns named by gamma labels). When missing, drawn with rmvnldl(n = num_sim, fit = fit).
num_grid	Target length for the default pretty grid.
num_sim	Number of draws when gamma_sims is missing.
probs	Quantile levels for pointwise intervals.
coverage	Coverage for central_region global envelopes (used only when **GET** is installed).
transform	Function applied to each simulated linear predictor before summarizing intervals (and returned as sim); default identity. Use exp for relative-risk scale.

Details

Only terms with model_role == "random" whose @name equals x are included (so companion rpoly bases for an iwp are included, while fixed fpoly bases are not). Column names of the design must match columns of gamma_sims.

IID prediction grids use factor levels stored in the gamma info table; levels not present at fit time are not invented here.

Value

A list with `x` (numeric or character grid), `sim` (an `length(x)` by draws matrix after `transform`), `pointwise` (quantile matrix), and `global` (GET envelope with columns `lo`, `central`, `hi`, or `NULL` if **GET** is unavailable).

See Also

`rmvnldl`, `sim_fit`, `sim_iid`, `design`

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

Description

Parameter-block row counts for an `density_data`

Usage

```
sizes(x, ...)

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

Arguments

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

Value

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

takahashi_davis *Sparse inverse subset of a symmetric positive-definite matrix*

Description

Entries of Q^{-1} on the sparsity pattern of the Cholesky factor, in the original variable order. Positions the factor does not store are absent, not zeros of the inverse.

Usage

```
takahashi_davis(Q, chol = NULL)
```

Arguments

`Q`	Symmetric sparse positive-definite matrix.
`chol`	Optional list with ‘perm‘ (0-based) and ‘L1‘ (unit-lower factor). When ‘NULL‘, the symbolic factor is computed from ‘Q‘.

Details

C++ callers ‘LinkingTo: adlaplace‘ and include ‘adlaplace/takahashi_impl.hpp‘. That function is ‘adlaplace::chol::takahashi_davis‘ and takes the unit-lower factor L and the LDL diagonal D. A caller that has only Q factors it with ‘chol_update_csc‘ first.

Value

A ‘dgCMatrix‘ of the selected inverse.

term_data_setup *Build design matrices and parameter metadata from a formula*

Description

Build design matrices and parameter metadata from a formula

Usage

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

Arguments

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

Index

∗ **datasets**
gamm, 30
germany, 32
london, 47
.model_role_levels, 4

ad_pack, *4*, 10, *12–15*, *19*, *33*, *34*, *37*, *46*, *47*, *56*
ad_pack,ad_pack_ptr-method *(ad_pack)*, 10
ad_pack,density_data-method *(ad_pack)*, 10
ad_pack,list-method *(ad_pack)*, 10
ad_pack-class, 12
ad_pack_drop, *11*, 12, *16*, *37*, *56*
ad_pack_ptr, 13, *14*
ad_pack_ptr-class, 14
ad_pack_set_X, 14
adlaplace, 4, *8*, *13*, *15*, *27*, *33*, *36*, *58*, *62*, *65*, *66*
adlaplace_cpp, 6
adlaplace_fit-methods, 8
aghq, *15*
aghq_ff, 15
amip_set, *13*, 16, *37*, *58*
apply_theta_log, 16

beta_info *(model-generics)*, 49
beta_info,dirichlet_multinom-method *(dirichlet_multinom-class)*, 25
beta_info,fpoly-method *(fpoly-class)*, 28
beta_info,intercept-method *(intercept-class)*, 38
beta_info,linear-method *(linear-class)*, 44
beta_info,model_term-method *(model-generics)*, 49
binom *(binomial-class)*, 17
binomial-class, 17
by_group, 18
by_group-class, 18

c.ad_pack_ptr, *14*, *19*, 19
central_region, *65*, *66*
Cholesky, *62*
clone_ad_pack_ptr, *8*, *14*, *19*, 19
coef.adlaplace_fit *(adlaplace_fit-methods)*, 8
collect_terms, *5*, 20, *30*, *51*
confint.adlaplace_fit, *5*
confint.adlaplace_fit *(adlaplace_fit-methods)*, 8

dat *(gamm)*, 30
density_data *(density_data-class)*, 20
density_data-class, 20
deriv_scan_joint, 22
deriv_scan_laplace, 23
design, *58*, *67*
design *(model-generics)*, 49
design,dirichlet_multinom-method *(dirichlet_multinom-class)*, 25
design,fpoly-method *(fpoly-class)*, 28
design,iid-method *(iid-class)*, 35
design,intercept-method *(intercept-class)*, 38
design,iwp-method *(iwp-class)*, 41
design,linear-method *(linear-class)*, 44
design,model_term-method *(model-generics)*, 49
design,rpoly-method *(rpoly-class)*, 63
dirichlet_multinom, 25
dirichlet_multinom-class, 25

elgm_matrix *(model-generics)*, 49
elgm_matrix,dirichlet_multinom-method *(dirichlet_multinom-class)*, 25
environment, *56*
extra_density *(model-generics)*, 49
extra_density,model_term-method *(model-generics)*, 49

fit_outer, *6*, 27
fitted.adlaplace_fit
(adlaplace_fit-methods), 8
format_parameters, 28
fpoly (fpoly-class), 28
fpoly-class, 28
fun_obj_fdfh, *8*
fun_obj_fdfh (adlaplace_cpp), 6

gamm, 30
gaussian-class, 30
germany, 32
grad, *11*, *19*, *23*, *33*, *34*
grad (adlaplace_cpp), 6
grad_obs_units, *11*, 33, *36*, *37*, *54*, *56*

has_openmp, 34
hessian, *11*, *19*, *23*
hessian (adlaplace_cpp), 6
hinv_trace_index, *12*, 34
hinv_trace_index,density_data-method
(hinv_trace_index), 34

iid, *65*
iid (iid-class), 35
iid-class, 35
influence_scores, *13*, *16*, 36, *58*
inner_opt, *8*, *11*, *13*, *33*, *34*, *36*, 37, *43*, *46*, *47*, *49*
intercept (intercept-class), 38
intercept-class, 38
iwp, *29*, 40, *63*
iwp-class, 41

joint_log_dens, *19*, *23*, *48*, *49*
joint_log_dens (adlaplace_cpp), 6

laplace_half_H_inv, 42
laplace_loglik, 43
linear (linear-class), 44
linear-class, 44
log_lik_laplace, *6*, *8*, *23*, *42*, *43*, 45, *49*, *56*, *57*, *62*, *64*
logLik.adlaplace_fit
(adlaplace_fit-methods), 8
london, 47

mean_mvquad, 48
model-generics, 49
model_data, *4*–*6*, *49*, 50, *64*
model_term-class, 51

n_groups, 53
n_obs_units, 54, *56*
nbinom (nbinom-class), 52
nbinom-class, 52
nobs.adlaplace_fit
(adlaplace_fit-methods), 8
normal (gaussian-class), 30

obs_groups, *4*, *5*, 54
obs_groups_units, *11*, *13*, 55
observation_response (model-generics), 49
observation_response,model_term-method
(model-generics), 49
optim, *4*, *5*, *27*
optimHess, *5*, *27*
outer_fn, *4*, *15*, *27*, *43*
outer_fn (outer_optim_wrappers), 56
outer_gr, *4*, *15*, *27*
outer_gr (outer_optim_wrappers), 56
outer_optim_wrappers, 56

phi_gamma, *36*, *37*, 57
plot.adlaplace_fit
(adlaplace_fit-methods), 8
pois (poisson-class), 58
poisson-class, 58
precision (model-generics), 49
precision,dirichlet_multinom-method
(dirichlet_multinom-class), 25
precision,iid-method (iid-class), 35
precision,intercept-method
(intercept-class), 38
precision,iwp-method (iwp-class), 41
precision,linear-method (linear-class), 44
precision,model_term-method
(model-generics), 49
precision,rpoly-method (rpoly-class), 63
predict.adlaplace_fit, *6*
predict.adlaplace_fit
(adlaplace_fit-methods), 8
pretty, *66*
print.adlaplace_fit
(adlaplace_fit-methods), 8
print.summary.adlaplace_fit
(adlaplace_fit-methods), 8

prior (prior-class), 59
prior-class, 59
psd, 59
psd,iwp-method (psd), 59
psd,list-method (psd), 59
psd,model_term-method (psd), 59
psd_prior, 60

random_info (model-generics), 49
random_info,dirichlet_multinom-method (dirichlet_multinom-class), 25
random_info,iid-method (iid-class), 35
random_info,intercept-method (intercept-class), 38
random_info,iwp-method (iwp-class), 41
random_info,linear-method (linear-class), 44
random_info,model_term-method (model-generics), 49
random_info,rpoly-method (rpoly-class), 63
replace_vars, *15*, 61
replace_vars,dirichlet_multinom-method (replace_vars), 61
replace_vars,formula-method (replace_vars), 61
replace_vars,list-method (replace_vars), 61
replace_vars,model_term-method (replace_vars), 61
rmvnldl, *42*, *49*, 62, *65–67*
rpoly (rpoly-class), 63
rpoly-class, 63

sim_fit, *9*, 64, *67*
sim_iid, 65, *67*
sim_random, *58*, *65*, 66
sizes, 67
sizes,density_data-method (sizes), 67
summary.adlaplace_fit, *5, 6*
summary.adlaplace_fit (adlaplace_fit-methods), 8
svd, *55*

takahashi_davis, 68
term_data_setup, *16*, *51*, 68
theta_info (model-generics), 49
theta_info,dirichlet_multinom-method (dirichlet_multinom-class), 25
theta_info,gaussian-method (gaussian-class), 30
theta_info,iid-method (iid-class), 35
theta_info,iwp-method (iwp-class), 41
theta_info,model_term-method (model-generics), 49
theta_info,nbinom-method (nbinom-class), 52
trace_hinv_t (adlaplace_cpp), 6

vcov.adlaplace_fit, *5*
vcov.adlaplace_fit (adlaplace_fit-methods), 8