Package ‘tulpa’

October 19, 2026

Title Templated Unified Library for Posterior Approximation in Bayesian Hierarchical Models

Version 0.7.5

Description A general-purpose engine for fitting Bayesian hierarchical models with spatial fields, temporal effects, spatially varying coefficients, and multiple inference backends. Scalable spatial structure includes Hilbert space approximate Gaussian processes (HSGP; Riutort-Mayol et al. 2023 <doi:10.1007/s11222-022-10167-2>), nearest-neighbor Gaussian processes (NNGP; Datta et al. 2016 <doi:10.1080/01621459.2015.1044091>), intrinsic conditional autoregressive models (ICAR; Besag, York, and Mollie 1991 <doi:10.1007/BF00116466>), the reparameterized Besag-York-Mollie model (BYM2; Riebler et al. 2016 <doi:10.1177/0962280216660421>), and stochastic partial differential equation fields (SPDE; Lindgren, Rue, and Lindstrom 2011 <doi:10.1111/j.1467-9868.2011.00777.x>). Temporal structure covers random walks, autoregressive processes, and Gaussian processes. Inference is tiered by correctness guarantee: exact Hamiltonian Monte Carlo with the No-U-Turn sampler, Laplace and nested Laplace approximations with hyperparameter integration (Rue, Martino, and Chopin 2009 <doi:10.1111/j.1467-9868.2008.00700.x>), and variational inference. Model-specific packages plug observation likelihoods into the engine through a templated C++ callback interface.

License MIT + file LICENSE

Encoding UTF-8

Language en-US

SystemRequirements C++17

Depends R (>= 4.1.0)

Imports Rcpp (>= 1.0.12), Matrix, tulpaMesh (>= 0.1.3), generics, stats, tools, utils, methods, graphics, grDevices

Suggests testthat (>= 3.0.0), bayesplot, coda, ggplot2, sf, terra, knitr, rmarkdown, posterior (>= 1.5.0), loo (>= 2.7.0), rstantools, lme4, nlme, lmtest, numDeriv, spdep, MASS, betareg, glmmTMB, pscl, tweedie, fmesher, patchwork, stars, statmod, withr

LinkingTo Rcpp, RcppEigen, Matrix

Config/testthat/edition 3

VignetteBuilder knitr

URL https://github.com/gcol33/tulpa, https://gillescolling.com/tulpa/

BugReports https://github.com/gcol33/tulpa/issues

Config/roxygen2/version 8.1.0

NeedsCompilation yes

Author Gilles Colling [aut, cre, cph] (ORCID: <https://orcid.org/0000-0003-3070-6066>),
Frances Y. Kuo [ctb, cph] (Sobol direction numbers in src/sobol_direction_numbers.h, BSD-3-clause),
Stephen Joe [ctb, cph] (Sobol direction numbers in src/sobol_direction_numbers.h, BSD-3-clause),
Arm Limited [cph] (exp and log routines in src/fastmath.h and src/fastmath_data.cpp, MIT),
The Walnutpie Developers [cph] (WALNUTS transition in inst/include/tulpa/walnuts.h, ported from walnutpie, MIT)

Maintainer Gilles Colling <gilles.colling051@gmail.com>

Repository CRAN

Date/Publication 2026-10-19 16:00:02 UTC

Contents

adjacency . . . 6
agq_fit . . . 9
AIC.tulpa_fit . . . 12
as_draws . . . 13
auto_grid . . . 14
auto_grid_place . . . 16
bayes_R2 . . . 17
bridge_sampling . . . 18
categorical_accessors . . . 19
check_adjacency . . . 21
check_diagnostics . . . 22
check_model . . . 23
coef.tulpa_fit . . . 24
compare_models . . . 24
confint.tulpa_fit . . . 25
criteria_doors . . . 26
diagnostics . . . 28
diagnostic_summary . . . 35
durbin_watson . . . 36
energy_bfmi . . . 37
family_names . . . 38

field_effects . . . 38
fitted.tulpa_fit . . . 40
fit_spde . . . 40
fit_st_nested . . . 44
fixef . . . 48
geweke_test . . . 49
glance.tulpa_fit . . . 50
imh_laplace . . . 51
inference_mode_info . . . 53
is_auto_grid . . . 53
latent . . . 54
latent_factor . . . 55
latent_factors . . . 57
logLik.tulpa_fit . . . 58
mala . . . 59
mcmc_draws . . . 61
model_average . . . 62
moran_i . . . 63
nobs.tulpa_fit . . . 64
node_index . . . 65
n_divergent . . . 66
pathfinder . . . 67
pit_residuals . . . 69
plot.tulpa_fit . . . 70
plot.tulpa_prior_predict . . . 70
plot.tulpa_st_summary . . . 71
plot.tulpa_svc_posterior . . . 72
plot.tulpa_temporal_posterior . . . 72
plot.tulpa_tvc_posterior . . . 73
plot_acf . . . 73
plot_divergences . . . 74
plot_energy . . . 75
plot_ess . . . 76
plot_map . . . 77
plot_map_panel . . . 79
plot_pairs . . . 81
plot_rhat . . . 82
posterior_predict . . . 83
posterior_sample . . . 85
post_hoc_lm . . . 86
pp_check . . . 87
predict.tulpa_fit . . . 88
print.tulpa_fit . . . 89
print.tulpa_nested_laplace . . . 92
print.tulpa_nested_laplace_joint . . . 101
print.tulpa_temporal_field_fit . . . 125
priors_default . . . 127
prior_beta . . . 128

prior_exponential . . . 129
prior_from_spec . . . 130
prior_gamma . . . 131
prior_half_cauchy . . . 131
prior_half_normal . . . 132
prior_normal . . . 132
prior_pc . . . 133
prior_predict . . . 134
ranef . . . 135
residuals.tulpa_fit . . . 136
re_cov_pc_lkj_prior . . . 137
rubins_pool . . . 138
sbc . . . 139
sbc_predictive . . . 143
select_main_params . . . 145
simulate.tulpa_fit . . . 145
smooth_effects . . . 146
spatial . . . 147
spatial_bym2 . . . 149
spatial_car . . . 150
spatial_car_proper . . . 151
spatial_gp . . . 152
spatial_multiscale . . . 154
spatial_range . . . 157
spatial_rsr . . . 158
spatial_spde . . . 161
spatial_spde_custom . . . 162
spatial_svc . . . 163
spatiotemporal . . . 165
spatiotemporal_effects . . . 167
spatiotemporal_gp . . . 169
svc . . . 170
temporal_ar . . . 172
temporal_ar1 . . . 173
temporal_ar2 . . . 175
temporal_corr . . . 176
temporal_gp . . . 177
temporal_multiscale . . . 179
temporal_rtr . . . 180
temporal_rw1 . . . 181
temporal_rw2 . . . 182
temporal_tvc . . . 183
test_dispersion . . . 185
test_outliers . . . 186
test_uniformity . . . 187
test_zero_inflation . . . 188
tgmrf . . . 188
tgmrf_cpp . . . 191

tidy.tulpa_fit . . . 192
tulpa . . . 193
tulpa_bar_field_replicate . . . 199
tulpa_bar_field_specs . . . 200
tulpa_batched_pareto_k . . . 202
tulpa_cache_clear . . . 203
tulpa_cache_dir . . . 204
tulpa_check_control . . . 205
tulpa_criteria . . . 205
tulpa_diagnostics . . . 207
tulpa_draws_array . . . 208
tulpa_eb . . . 209
tulpa_em_laplace . . . 213
tulpa_em_mc . . . 217
tulpa_ep . . . 219
tulpa_family . . . 221
tulpa_fit_summary . . . 222
tulpa_formula . . . 223
tulpa_gaussian . . . 223
tulpa_gibbs . . . 224
tulpa_grid_axis . . . 226
tulpa_grid_log_quad . . . 227
tulpa_hyper_check_copy_slab . . . 229
tulpa_hyper_copy_slab_density . . . 229
tulpa_hyper_draws . . . 230
tulpa_hyper_grid . . . 232
tulpa_hyper_grid_supports . . . 235
tulpa_hyper_slice_home . . . 236
tulpa_integrator . . . 237
tulpa_is_spatial_bar . . . 238
tulpa_joint_axis_specs_from_grid . . . 239
tulpa_joint_grid_batch . . . 240
tulpa_joint_inner_vcov_blocks . . . 241
tulpa_joint_inner_vcov_mixture . . . 243
tulpa_kfold . . . 245
tulpa_laplace . . . 246
tulpa_laplace_beta . . . 249
tulpa_latent . . . 251
tulpa_loglik . . . 251
tulpa_multinomial . . . 252
tulpa_normalise_weights_safe . . . 253
tulpa_nuts_beta . . . 254
tulpa_nuts_spde . . . 256
tulpa_ordinal . . . 258
tulpa_outer_grid . . . 260
tulpa_parse_formula . . . 261
tulpa_pit . . . 262
tulpa_posterior_draws . . . 263

tulpa_posterior_draws.tulpa_nested_laplace_joint . . . 265
tulpa_powerscale_sensitivity . . . 266
tulpa_priors . . . 268
tulpa_profile . . . 270
tulpa_psis . . . 271
tulpa_reloo . . . 273
tulpa_re_aghq . . . 274
tulpa_re_cov_gibbs . . . 278
tulpa_re_cov_nested . . . 281
tulpa_rpg . . . 286
tulpa_simulate . . . 287
tulpa_spatial . . . 288
tulpa_spde_log_hyperprior . . . 289
tulpa_spde_precision_Q . . . 290
tulpa_temporal . . . 291
tulpa_tgmrf . . . 291
tulpa_theta_matrix . . . 293
tulpa_validate . . . 294
tulpa_variogram . . . 294
tvc . . . 295
validate_gp . . . 296
validate_mode . . . 297
validate_temporal_multiscale . . . 298
VarCorr . . . 299
vcov.tulpa_fit . . . 300
wbic . . . 300
with_tulpa_integrator . . . 302

Index **303**

`adjacency` *Construct a spatial adjacency graph for areal models*

Description

Build the symmetric adjacency matrix that `spatial()` and `spatial_car()` consume, from a common spatial layout, instead of hand-coding it. The model still receives an explicit graph (`spatial(graph = g$adjacency, ...)`), so the graph stays inspectable before fitting – the engine never guesses connectivity from coordinates silently.

`adjacency()` is a single front-door verb that dispatches on the kind of input:

a `data.frame` cell centroids on a regular grid – queen (edge or corner) or rook (edge only) contiguity over the lattice. This covers both plain coordinate grids and rasterised cells passed as a table of centres.

an `sf` **object** polygon contiguity from shared boundaries (queen = share a point or edge; rook = share an edge). Requires **sf**.

a `SpatRaster` **(terra) or** `stars` **object** raster cells – the lattice of (non-NA) cell centres, queen or rook. Requires **terra** or **stars** respectively.

The result is a `tulpa_adjacency` object holding the matrix plus the cell identifier for each node, so a model can pass `g$adjacency` while the observation data is remapped to 1-based node indices with `node_index()`.

Usage

```
adjacency(x, ...)

## Default S3 method:
adjacency(x, ...)

## S3 method for class 'data.frame'
adjacency(
  x,
  type = c("queen", "rook"),
  x_coord = "x",
  y_coord = "y",
  id = NULL,
  order = 1L,
  tolerance = 1.5,
  offsets = NULL,
  ...
)

## S3 method for class 'sf'
adjacency(x, type = c("queen", "rook", "touches"), id = NULL, ...)

## S3 method for class 'SpatRaster'
adjacency(
  x,
  type = c("queen", "rook"),
  order = 1L,
  tolerance = 1.5,
  offsets = NULL,
  na_rm = TRUE,
  ...
)

## S3 method for class 'stars'
adjacency(
  x,
  type = c("queen", "rook"),
  order = 1L,
  tolerance = 1.5,
  offsets = NULL,
  na_rm = TRUE,
```

```
  ...
)

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

Arguments

`x`	The spatial layout: a `data.frame` of centroids, an `sf` polygon layer, or a raster (`SpatRaster` / `stars`).
`...`	Passed to methods.
`type`	Contiguity rule: `"queen"` (default; neighbours share an edge or a corner) or `"rook"` (neighbours share an edge only). For polygons, `"touches"` is an alias of `"queen"`.
`x_coord, y_coord`	For the `data.frame` method, the names of the coordinate columns holding the cell centres (default `"x"` and `"y"`).
`id`	For the `data.frame` and `sf` methods, an optional column naming each cell's identifier. Node `i` of the graph corresponds to `id`'s value in row `i`; pass this column to `node_index()` to translate the observation data's cell column into node indices. Default `NULL` uses the row position (`1:n`) as the identifier.
`order`	For grid / raster layouts, the neighbourhood order (ring count): `order = 1` (default) is first-order contiguity (queen = 8 neighbours, rook = 4); `order = k` extends the stencil to the k-th ring, so queen keeps every cell within Chebyshev distance `k` (`(2k + 1)^2 - 1` neighbours: 8, 24, 48, ... for `k = 1, 2, 3`) and rook every cell within Manhattan distance `k` (`2k(k + 1)` neighbours: 4, 12, 24, ...). Any positive integer is allowed, so the neighbourhood is fully settable. Ignored by the `sf` (polygon) method, which is first-order contiguity only.
`tolerance`	For grid / raster layouts, the per-offset neighbour distance cut-off as a multiple of the inferred cell size: a candidate at a lattice offset is kept when its true centre distance is at most `tolerance` times that offset's expected distance. The default `1.5` admits a snapped neighbour while rejecting one much farther than its lattice slot implies; raise it only for irregularly spaced centroids. Neighbourhood extent is set by `order`, not by `tolerance`.
`offsets`	Advanced, grid / raster layouts only: a custom neighbour stencil as a two-column integer matrix or a list of length-2 `c(dx, dy)` lattice offsets (cell-step units, origin excluded). Supplying it overrides `type` and `order` and builds exactly that stencil, so any neighbourhood is expressible (anisotropic, ring-only, off-axis). An ICAR / CAR field is *undirected*, so the graph must be symmetric: an asymmetric stencil (e.g. `list(c(0, 1), c(-1, 0), c(1, 0))` = up / left / right but not down) is symmetrized to an undirected graph, with a message, since a directed neighbourhood cannot be represented by an undirected field. Default `NULL` uses the `type` / `order` stencil.
`na_rm`	For raster layouts, drop cells whose value is `NA` before building the graph (default `TRUE`), so the nodes are the cells that carry data.

Value

A `tulpa_adjacency` object: a list with

`adjacency` the `[n x n]` symmetric sparse adjacency matrix (`dgCMatrix`, 0/1, zero diagonal) to pass as `graph` / `adjacency`.

`ids` the cell identifier for each node, in node order (length `n`).

`n` the number of nodes.

`cellsize` the inferred cell size `c(x, y)` for grid / raster layouts, or `NA` for polygons.

`type` the contiguity rule used.

See Also

`node_index()` to map cell identifiers to node indices, `check_adjacency()` to validate a hand-built matrix, `spatial()` and `spatial_car()` which consume the graph.

Examples

```
# A 3 x 3 regular grid of cell centres
grid <- expand.grid(x = 1:3, y = 1:3)
grid$cell <- paste0("c", seq_len(nrow(grid)))

g <- adjacency(grid, x_coord = "x", y_coord = "y", id = "cell",
               type = "queen")
g
g$adjacency

# Second-order (24-neighbour) queen contiguity: any order is settable
g2 <- adjacency(grid, id = "cell", order = 2)

# Advanced: a custom stencil (symmetrized for the undirected field)
g3 <- adjacency(grid, id = "cell", offsets = list(c(1, 0), c(0, 1)))

# Use it in a model: graph stays explicit and inspectable
spatial(graph = g$adjacency, formula = ~ 1 || cell_idx)

# Remap observation data (original cell ids -> 1:n node indices) by key
obs <- data.frame(cell = c("c5", "c1", "c5", "c9"))
obs$cell_idx <- node_index(g, obs$cell)
obs
```

---

agq_fit — *Adaptive Gauss-Hermite quadrature for one-RE GLMMs*

Description

Marginal-likelihood approximation for a generalised linear mixed model with one cluster-level intercept random effect. Adaptive Gauss-Hermite quadrature (AGQ) generalises Laplace by replacing the single-point Gaussian integral around the cluster's posterior mode with `n_quad`-point quadrature. AGQ at `n_quad = 1` recovers the Laplace approximation; higher `n_quad` reduces approximation error, especially for clusters with few observations or non-Gaussian likelihoods.

This is an engine block, not a front door: its tuning knobs (`max_iter`, `tol`, `n_quad`) sit in the signature rather than in a `control` list.

Scoped to the `lme4::glmer(..., nAGQ = N)` use case: one intercept- only RE term, families `binomial`, `poisson`, or `gaussian`. For multi-RE or random-slope models, use `tulpa_laplace()` (Laplace at the joint mode) or HMC.

Usage

```
agq_fit(
  y,
  X,
  group,
  n_groups = max(group),
  family = c("binomial", "poisson", "gaussian"),
  n_trials = NULL,
  sigma_eps = 1,
  n_quad = 7L,
  offset = NULL,
  beta_init = NULL,
  sigma_init = 1,
  max_iter = 200L,
  tol = 1e-06,
  verbose = FALSE
)
```

Arguments

`y`	Response vector.
`X`	Fixed-effects design matrix (`n_obs x p`).
`group`	Integer cluster labels (1-based, length `n_obs`).
`n_groups`	Number of clusters (default `max(group)`).
`family`	One of `"binomial"`, `"poisson"`, `"gaussian"`.
`n_trials`	Trial sizes (binomial only; default `rep(1, n_obs)`).
`sigma_eps`	Residual SD (gaussian only; default `1`). Held **fixed** at this value – it is not profiled or estimated, so for a gaussian response set it to a known / pre-estimated residual SD rather than the default.
`n_quad`	Number of Gauss-Hermite quadrature nodes per cluster. `1` recovers Laplace; common choices are `5` or `7`. Default `7`.
`offset`	Optional fixed per-observation offset added to eta (default all zero, i.e. none).

beta_init	Initial fixed-effects (default zeros).
sigma_init	Initial RE SD (default 1).
max_iter	Optimiser iteration cap (default 200).
tol	Convergence tolerance (default 1e-6).
verbose	Print optimiser summary (default FALSE).

Value

A list with class tulpa_fit carrying:

- means: named vector c(beta, log_sigma_re).
- mode: same as means (Gaussian fit at the optimum).
- cov: covariance matrix of (beta, log_sigma_re) from inverse observed information.
- sigma_re: estimated RE SD.
- log_marginal: AGQ-approximated log marginal likelihood at the fitted values.
- n_quad: quadrature order used.
- n_iter, converged: optimiser diagnostics.
- inference_mode: "structured".
- inference_tier: 2L.
- backend: "agq".

An optimum at which the AGHQ objective is undefined (some group's solve fails there, so its value is the failure sentinel rather than a marginal likelihood) is an error naming the groups, not a returned fit.

Tier

Tier 2 (Structured). AGQ is exact in the limit n_quad -> infinity but at finite n_quad it is a controlled approximation – same epistemic class as Laplace.

References

Pinheiro, J. C., & Bates, D. M. (1995). Approximations to the log-likelihood function in the non-linear mixed-effects model. *Journal of Computational and Graphical Statistics*, 4(1), 12-35.

See Also

tulpa_laplace() for the joint-mode Laplace path (handles multi-RE and spatial structure).

Examples

```
set.seed(1)
n_g <- 20; n_per <- 5; n <- n_g * n_per
group <- rep(seq_len(n_g), each = n_per)
x <- rnorm(n)
X <- cbind(1, x)
u <- rnorm(n_g, 0, 0.8)
```

```
eta <- 0.3 + 0.7 * x + u[group]
y <- rbinom(n, 1, plogis(eta))

# Compare Laplace (n_quad = 1) and AGQ-7.
fit_lap <- agq_fit(y, X, group, family = "binomial", n_quad = 1)
fit_agq <- agq_fit(y, X, group, family = "binomial", n_quad = 7)
c(fit_lap$log_marginal, fit_agq$log_marginal)
```

`AIC.tulpa_fit` *Information criteria on a tulpa fit*

Description

AIC and BIC penalise a maximised log-likelihood. `logLik.tulpa_fit()` reports one only for a fit that maximised over every parameter it reports (`quantity = "log_likelihood"`, as `agq_fit()` does); a mean log posterior, a log evidence and a conditional log marginal likelihood are not, and both criteria refuse them rather than return a number. Compare those fits by their evidence, or by `compare_models(criterion = "waic")` / `"loo"`.

Usage

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

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

Arguments

`object`	A `tulpa_fit` object.
`...`	Further fits.
`k`	Penalty per parameter.

Value

As `stats::AIC()` / `stats::BIC()` for maximised log-likelihoods; errors otherwise.

as_draws *Posterior draws in the posterior package's format*

Description

Convert a fit's posterior draws to a `posterior` draws object. `as_draws()` returns the `draws_array` shape; `as_draws_array()`, `as_draws_matrix()`, `as_draws_df()` and `as_draws_rvars()` return theirs. When the `posterior` package is installed these are also registered against its generics, so `posterior::as_draws(fit)` works.

Usage

```
as_draws(x, ...)

## S3 method for class 'tulpa_fit'
as_draws(x, n_draws = NULL, seed = NULL, ...)

as_draws_array(x, ...)

## S3 method for class 'tulpa_fit'
as_draws_array(x, n_draws = NULL, seed = NULL, ...)

as_draws_matrix(x, ...)

## S3 method for class 'tulpa_fit'
as_draws_matrix(x, n_draws = NULL, seed = NULL, ...)

as_draws_df(x, ...)

## S3 method for class 'tulpa_fit'
as_draws_df(x, n_draws = NULL, seed = NULL, ...)

as_draws_rvars(x, ...)

## S3 method for class 'tulpa_fit'
as_draws_rvars(x, n_draws = NULL, seed = NULL, ...)
```

Arguments

`x`	A `tulpa_fit` object.
`...`	Passed to the corresponding `posterior` converter.
`n_draws`	Number of draws to synthesize from the Gaussian approximation for a fit that carries none. `NULL` (default) errors on such a fit rather than silently approximating. Ignored, with a warning, when the fit already carries draws.
`seed`	Optional integer seed for the synthesis. The RNG state is restored afterwards.

Details

Fits differ in whether they carry draws at all. Sampler and nested-Laplace fits do, and convert directly. A Gaussian-approximation fit (`mode = "laplace"`, `mode = "eb"`) carries a mode and a precision instead, and converting it means *drawing from the approximation* – which is a modelling decision, not a format change, because every downstream `posterior` summary would then treat a normal approximation as a posterior sample. So it is opt-in: pass `n_draws` to synthesize that many draws from `N(coef(object), vcov(object))`, or get an error naming the alternative. Synthesized draws form a single chain and cover the fixed effects only.

Value

A `posterior` draws object of the requested shape.

See Also

`tulpa_draws_array()` for the base R array without the dependency, `posterior_sample()` for the raw matrix.

Examples

```
set.seed(1)
df <- data.frame(x = rnorm(80))
df$y <- rpois(80, exp(0.5 + 0.3 * df$x))
fit <- tulpa(y ~ x, data = df, family = "poisson")
if (requireNamespace("posterior", quietly = TRUE)) {
  posterior::summarise_draws(as_draws(fit))
}
```

---

auto_grid — *Mark an outer-grid setting as a default rather than a pin*

Description

Declares that a setting shaping the outer hyperparameter grid carries a *default* the caller computed, not a choice the user made. The auto-recenter pass (`outer_grid_placement`) leaves a user-pinned setting exactly as given, and re-centres (or, for a prior, engages its own regularizer over) a marked one when the fit rails against its ceiling.

Four kinds of setting take the mark:

- a grid axis on a nested-Laplace `prior` block (`sigma_grid`, `tau_grid`, ...) – a numeric vector of nodes, or the `[n_cells x k]` matrix of pre-paired coordinates the families whose axis is a matrix take (`mcar` / `miid`'s `logchol_grid`, `tgmrf`'s `theta_grid_built`);
- an entry of `tulpa_nested_laplace_joint()`'s `phi_grid` – one arm's dispersion axis;
- a scalar grid-construction knob in `control`, for a driver that builds its axes rather than taking them (`fit_st_nested()`'s `n_grid_spatial`, `tau_upper`, ...);

- a `prior_sigma` hyperprior specification – a list, e.g. `list("pc.prec", c(U = 3, alpha = 0.01))`.

Wrapper packages are the intended caller: one that builds a default of its own – because it derives a second axis from it, hands the same vector to several blocks, or exposes its own argument with a default – would otherwise be indistinguishable from a user who pinned that setting deliberately. Mark it and the rescue stays live. A setting whose value is exactly the engine's own default is recognised without a mark; anything else needs one – and a dispersion axis ALWAYS needs one, since the engine has no default of its own to recognise it against.

The mark is an attribute, so it is dropped by `sort()`, `[`, `c()` and `as.numeric()`: build the value first, mark it last.

Usage

```
auto_grid(x, place = TRUE)
```

Arguments

`x`	Numeric vector or matrix of grid nodes, a numeric scalar knob, or a prior-specification list.
`place`	May the auto-placement pass move this axis onto its own posterior? `TRUE` (default) declares a default the engine may re-place; `FALSE` declares one it must integrate as written.

Value

`x` carrying the marker attribute. Numeric input is coerced to double IN PLACE, so everything else it carries – `dim()` and `dimnames()` above all – survives the mark; a list is returned unchanged apart from the attribute.

Declaring a default the engine must integrate as written

`place = FALSE` separates the two questions the mark otherwise answers at once. Provenance – whose choice these nodes are – and placement policy – whether the pass may move them – are different questions, and a package that measured its own default as the one to integrate has an answer to the second that is not "the user pinned it". Such an axis is treated exactly as a pin everywhere the engine ACTS on it (the placement pass leaves it, refinement densifies within its span rather than following the posterior past the end nodes, and no curvature is computed for it), and differs only in what the fit REPORTS: `"default_axis_pinned"` rather than `"axis_pinned"`, so a reader is not told they pinned an axis they never wrote down. Leaving the mark off instead buys the same integration and says the wrong thing about it.

`place` is read wherever the engine would otherwise RE-PLACE the setting: a grid axis, and a scalar grid-construction knob. A `prior_sigma` specification is not re-placed but replaced, by the engine's own regularizer, and carries no `place`.

See Also

`is_auto_grid()`, `auto_grid_place()`, `tulpa_nested_laplace_joint()`, `fit_st_nested()`

Examples

```
prior <- list(type = "icar", sigma_grid = auto_grid(c(0.1, 0.5, 1, 2, 3)))
is_auto_grid(prior$sigma_grid)
is_auto_grid(auto_grid(list("pc.prec", c(U = 3, alpha = 0.01))))
auto_grid_place(auto_grid(c(0.5, 1, 2), place = FALSE))
```

---

auto_grid_place *May the placement pass move a marked outer-grid axis?*

Description

Reads back what auto_grid()'s place argument recorded, so a wrapper that rebuilds a value (as.numeric() drops every attribute) can re-apply both halves of the declaration rather than only the provenance half.

Usage

```
auto_grid_place(x)
```

Arguments

x	Any object.

Value

FALSE when x was marked auto_grid(place = FALSE), TRUE otherwise – including for a value carrying no mark at all, which the engine holds because it reads as a pin rather than because it asked to be held.

See Also

auto_grid(), is_auto_grid()

Examples

```
auto_grid_place(auto_grid(c(0.5, 1, 2)))
auto_grid_place(auto_grid(c(0.5, 1, 2), place = FALSE))
```

bayes_R2 *Bayesian R-squared*

Description

Per posterior draw `s`, `R2_s = Var_i(mu_si) / (Var_i(mu_si) + Var_res_s)`, where `mu_si` are the response-scale fitted means and `Var_res_s` is the family's residual variance averaged over observations (Gelman et al. 2019). The linear predictor is rebuilt per draw exactly as in `posterior_predict()` (fixed effects + random effects + offset at the training data).

Usage

```
bayes_R2(object, ...)

## S3 method for class 'tulpa_fit'
bayes_R2(
  object,
  ndraws = NULL,
  summary = TRUE,
  probs = c(0.025, 0.975),
  seed = NULL,
  ...
)
```

Arguments

`object`	A `tulpa_fit` object from `tulpa()`.
`...`	Passed to methods.
`ndraws`	Number of posterior draws to use. Defaults to all stored draws, or 400 on the draw-free Laplace tier.
`summary`	Summarize the per-draw values (default `TRUE`).
`probs`	Quantiles reported by the summary (default 2.5% / 97.5%).
`seed`	Optional integer seed (RNG state is restored on exit), used by the Gaussian fixed-effect sampling on draw-free fits.

Value

With `summary = TRUE` (default) a one-row data frame with `estimate` (posterior median), `std.error`, and the `probs` quantiles; with `summary = FALSE` the vector of per-draw R^2 values.

References

Gelman, Goodrich, Gabry & Vehtari (2019). R-squared for Bayesian regression models. *The American Statistician* 73(3):307-309.

Examples

```
set.seed(1)
d <- data.frame(x = rnorm(200))
d$y <- rnorm(200, 2 * d$x, 1)
fit <- tulpa(y ~ x, data = d, family = "gaussian", mode = "laplace", phi = 1)
bayes_R2(fit)
```

`bridge_sampling` *Bridge sampling for marginal likelihood*

Description

Estimate the log marginal likelihood $\log Z = \log p(y)$ from posterior draws via the Meng-Wong / Gronau bridge-sampling identity. Useful for Bayes factors, model comparison, and as a sanity check on Laplace-approximated marginal likelihoods.

The classical iterative scheme (Meng & Wong 1996; Gronau et al. 2017) is used with a multivariate-normal proposal fit to half of the posterior draws – the other half is used for the bridge ratio so the proposal is independent of the samples that score it. All computations are done in log-space via `logsumexp` for numerical stability.

Usage

```
bridge_sampling(
  draws,
  log_posterior,
  n_proposal = nrow(draws),
  max_iter = 1000L,
  tol = 1e-10,
  split = TRUE,
  verbose = FALSE
)
```

Arguments

`draws`	Numeric matrix of posterior draws, one parameter per column. Typically `fit$draws` from a tulpa Tier-1 fit.
`log_posterior`	Function `function(theta) -> numeric` returning the *unnormalized* log posterior density `log p(theta, y)` at a single parameter vector. Must accept a numeric vector of length `ncol(draws)` and return a finite scalar.
`n_proposal`	Number of proposal draws. Default `nrow(draws)`.
`max_iter`	Maximum bridge iterations (default 1000).
`tol`	Convergence tolerance on `|log r_{t+1} - log r_t|` (default 1e-10).
`split`	Logical: split the posterior draws in half so the proposal is fit on one half and scored on the other (default `TRUE`, recommended). Set `FALSE` only for diagnostic comparison.
`verbose`	Print iteration history (default `FALSE`).

Value

A list with:

- `log_marginal`: the bridge-sampling estimate of $\log Z$.
- `n_iter`: bridge iterations to convergence.
- `converged`: logical.
- `re_sd`: relative MSE estimate (Fruehwirth-Schnatter 2004); a rough quality check, smaller is better.
- `proposal`: the fitted proposal `list(mean, cov)`.

Tier

Bridge sampling is a *post-hoc* marginal-likelihood estimator, not a sampling backend, so it does not appear in the `INFERENCE_TIERS` registry. It operates on draws from any Tier-1 (Exact) backend.

References

Meng, X.-L., & Wong, W. H. (1996). Simulating ratios of normalizing constants via a simple identity: a theoretical exploration. *Statistica Sinica*, 6, 831-860.

Gronau, Q. F., Sarafoglou, A., Matzke, D., Ly, A., Boehm, U., Marsman, M., ... & Steingroever, H. (2017). A tutorial on bridge sampling. *Journal of Mathematical Psychology*, 81, 80-97.

Examples

```
# Toy: marginal likelihood of N(theta | 0, 1) under Y = theta + eps,
# eps ~ N(0, 1), prior theta ~ N(0, 10). Closed-form available.
y <- 1.5
log_post <- function(theta) {
  dnorm(y, theta, 1, log = TRUE) + dnorm(theta, 0, 10, log = TRUE)
}
draws <- matrix(rnorm(2000, mean = y * 100 / 101, sd = sqrt(100 / 101)),
                ncol = 1)
bs <- bridge_sampling(draws, log_post)
bs$log_marginal  # should match dnorm(y, 0, sqrt(101), log = TRUE)
```

`categorical_accessors` *Observation-level accessors for categorical fits*

Description

`stats::fitted()`, `stats::residuals()`, `stats::predict()` and `posterior_predict()` for a categorical response: a `tulpa_multinomial()` fit (`family = "multinomial"`) or a `tulpa_ordinal()` fit (`family = "ordinal"`). The response is one of K classes, so the response-scale quantities are N x K matrices with one column per class.

- `fitted()` returns the class probabilities at the posterior mode.

- residuals() returns, per class, the indicator of the observed class minus its fitted probability ("response"), or that difference divided by the indicator's standard deviation sqrt(p (1 - p)) ("pearson"). A categorical response has no single scalar residual; each class column is the Bernoulli residual of that class's indicator.
- predict() returns the link scale – the K - 1 baseline-category logits of the multinomial model (N x (K - 1)), or the ordinal location predictor x' beta (length N) – or the class probabilities (type = "response").
- posterior_predict() draws one class per observation for each posterior draw and returns the class indices, with the response levels attached; simulate.tulpa_fit() turns them into factor columns.

The pointwise log-likelihood behind cpo(), dic(), loo::waic() and loo::loo() is the log probability of each observed class at each stored posterior draw; dic() evaluates it at the posterior-mean parameters.

Usage

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

## S3 method for class 'tulpa_categorical'
residuals(object, type = c("pearson", "response"), ...)

## S3 method for class 'tulpa_categorical'
predict(
  object,
  newdata = NULL,
  type = c("link", "response"),
  se.fit = FALSE,
  level = 0.95,
  ...
)

## S3 method for class 'tulpa_categorical'
posterior_predict(object, newdata = NULL, ndraws = NULL, seed = NULL, ...)

## S3 method for class 'tulpa_categorical'
pp_check(object, ndraws = 50, ...)
```

Arguments

object	A tulpa_multinomial or tulpa_ordinal fit.
...	Ignored.
type	For residuals(), "pearson" (default) or "response". For predict(), "link" (default) or "response".
newdata	Optional data frame of covariates; NULL uses the training design.
se.fit	For predict(), also return standard errors and credible bounds. On the link scale the standard error is exact for the Gaussian (Laplace) posterior, sqrt(a' V a),

with Gaussian bounds; on the response scale the standard error and the bounds are the posterior SD and quantiles of each class probability over the fit's posterior draws.

`level` Credible-interval level (default 0.95).

`ndraws` Number of posterior draws for `posterior_predict()`; defaults to all stored draws.

`seed` Optional integer seed (RNG state is restored on exit).

Value

`fitted()`: an N x K probability matrix. `residuals()`: an N x K matrix. `predict()`: a matrix (or, for the ordinal link scale, a vector); with `se.fit = TRUE` a list of `fit`, `se.fit`, `lower`, `upper` of that shape. `posterior_predict()`: an `ndraws x N` integer matrix of class indices with attributes `levels` and `ordered`.

See Also

`tulpa_multinomial()`, `tulpa_ordinal()`

---

check_adjacency *Validate a spatial adjacency matrix*

Description

Check that a hand-built adjacency matrix is a well-formed spatial graph before passing it to `spatial()` / `spatial_car()`: square, finite, symmetric, zero on the diagonal, 0/1 valued, free of negative weights and of isolated nodes, and connected. `adjacency()` runs the same checks on the graphs it constructs. The spatial constructors refuse a graph with a missing, non-finite or negative entry.

Usage

```
check_adjacency(adjacency, ids = NULL)

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

Arguments

`adjacency` A matrix (dense or sparse `Matrix`).

`ids` Optional cell identifiers; if supplied, their length must match `nrow(adjacency)` and they must be unique.

`x` A `tulpa_adjacency_check` report.

`...` Ignored.

Value

Invisibly, a `tulpa_adjacency_check` list with the per-check results (`square`, `finite`, `symmetric`, `zero_diag`, `binary`, `nonneg`, isolated-node indices, `n_components` and `component_sizes`, edge count) and an overall `ok` flag. Issues are reported via `warning()` and printed; the function does not stop, so every problem surfaces in one pass.

See Also

`adjacency()` to construct a graph, `node_index()`.

Examples

```
adj <- matrix(0, 4, 4)
for (i in 1:3) adj[i, i + 1] <- adj[i + 1, i] <- 1
check_adjacency(adj)
```

---

check_diagnostics *Quick convergence check*

Description

Runs the core convergence diagnostics on a fit and reports whether Rhat, bulk-ESS, and divergence thresholds are all met. A terse companion to `diagnostic_summary()` for use in scripts and tests.

Usage

```
check_diagnostics(
  fit,
  rhat_threshold = 1.01,
  ess_threshold = 400,
  quiet = FALSE
)
```

Arguments

`fit`	A `tulpa_fit` object.
`rhat_threshold`	Maximum acceptable Rhat (default 1.01).
`ess_threshold`	Minimum acceptable bulk-ESS (default 400).
`quiet`	Logical; if TRUE, suppress messages (default FALSE).

Value

Invisibly, `TRUE` if all checks pass, `FALSE` if any fail, or `NA` for a non-chain (approximation) fit where Rhat/ESS do not apply.

See Also

diagnostic_summary(), diagnostics(), n_divergent()

Examples

```
set.seed(123)
df <- data.frame(x = rnorm(60))
df$y <- rpois(60, exp(0.5 + 0.3 * df$x))
fit <- tulpa(y ~ x, data = df, family = "poisson", mode = "hmc",
             control = list(n_iter = 500L, warmup = 250L, n_chains = 2L,
                            seed = 1L))
check_diagnostics(fit)
```

---

check_model *Diagnostic panel plot*

Description

Produces a 2x2 or 1x3 panel of diagnostic plots:

1. QQ plot of PIT residuals vs Uniform (with KS p-value)
2. Residuals vs fitted (with lowess smoother)
3. Dispersion histogram (observed variance vs simulated)
4. Spatial correlogram via Moran's I (if coords provided)

Usage

```
check_model(object, ...)

## Default S3 method:
check_model(object, coords = NULL, nsim = 250L, seed = 123L, ...)
```

Arguments

object	A fitted model with simulate(), fitted(), residuals()
...	Passed to methods.
coords	Optional N x 2 coordinate matrix for spatial panel
nsim	Number of simulations (default 250)
seed	Random seed (default 123)

Value

Invisible list with ks_p, disp_ratio, moran (if spatial)

Examples

```
set.seed(1)
d <- data.frame(x = rnorm(80))
d$y <- rpois(80, exp(0.3 + 0.5 * d$x))
fit <- tulpa(y ~ x, data = d, family = "poisson", mode = "laplace")
check_model(fit, nsim = 50)
```

---

coef.tulpa_fit *Fixed-effect coefficients*

Description

Fixed-effect coefficients

Usage

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

Arguments

`object`	A `tulpa_fit` object.
`...`	Ignored.

Value

Named numeric vector of fixed-effect posterior means (the Laplace mode for the Laplace tier). Random effects come from `ranef()`.

---

compare_models *Compare models by information criteria*

Description

Rank fitted models best-first by an information criterion. `"waic"` and `"loo"` use the native pointwise log-likelihood layer (`tulpa_criteria()` / `tulpa_psis()`) – WAIC and PSIS-LOO respectively, computed from each fit's `[n_draws x n_obs]` pointwise log-likelihood (`fit$draws$log_lik`), with no `loo` package dependency. `"loglik"` returns the (integrated) joint log-likelihood with the parameter count. A fit carrying no pointwise log-likelihood (a deterministic / point approximation) yields `NA` criterion columns rather than an error, so the table always has one row per model.

`"wbic"` ranks by `wbic()`, an estimate of each fit's negative log evidence, and needs every fit to carry its tempered draws (`mode = "smc"`, `control = list(wbic = TRUE)`); a fit without them gets an `NA` row.

Usage

```
compare_models(..., criterion = c("waic", "loo", "wbic", "loglik"))
```

Arguments

`...`	Named `tulpa_fit` objects.
`criterion`	`"waic"` (default), `"loo"`, `"wbic"`, or `"loglik"`.

Value

A data frame. For `"wbic"` (ranked best-first): `model`, `wbic`, `delta` (WBIC gap to the best model, >= 0), and `weight` (`exp(-delta)` normalized: the posterior model probability under equal prior model weights, with each log evidence read off its WBIC). For `"loglik"`: `model`, `n_params`, `logLik`, `quantity` and `conditioned_on` (which log-scale quantity `logLik()` reports, and the hyperparameters it holds fixed; fits that disagree on either are refused rather than ranked). For `"waic"` / `"loo"` (ranked best-first): `model`, `elpd`, `se_elpd`, `p_eff`, `ic` (`-2 * elpd`), `delta` (elpd gap to the best model), `se_diff` (SE of that pointwise elpd difference), and `weight` (the Akaike-style weight on the criterion).

See Also

`model_average()` for model-averaged predictions, `tulpa_criteria()` and `tulpa_psis()` for the native criteria layer.

Examples

```
set.seed(1)
df <- data.frame(x = rnorm(120))
df$y <- rpois(120, exp(0.4 + 0.5 * df$x))
f1 <- tulpa(y ~ x, data = df, family = "poisson")
f2 <- tulpa(y ~ 1, data = df, family = "poisson")
compare_models(full = f1, null = f2, criterion = "waic")
```

`confint.tulpa_fit` *Credible intervals for the fixed effects*

Description

Credible intervals for the fixed effects

Usage

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

Arguments

`object`	A `tulpa_fit` object.
`parm`	Parameter names or indices (default: all fixed effects). Indices address the fixed effects. A name that is not a fixed effect is read off the fit's posterior draws when they carry a column of that name (for example a sampler's `log_sigma_re`), as their equal-tailed quantile interval; any other name is an error.
`level`	Interval level, a single number in `(0, 1)` (default 0.95).
`...`	Ignored.

Value

Matrix with lower and upper columns, labelled as `stats::confint.default()` labels them (`"2.5 %"` and `"97.5 %"` at the default level). A nested-Laplace fit carries `interval_source` / `interval_declined` (which read produced the bounds – by default the grid's Gaussian-mixture CDF), `retained_mass` (the share of the grid weight the bounds are conditional on), and `skew_applied`, one logical per reported coefficient saying whether its bounds are the inner-Laplace skew-corrected quantiles or not. See `summary.tulpa_fit()`.

`criteria_doors` *DIC, CPO, WAIC and PSIS-LOO on a fit*

Description

Generic front doors onto the two criteria `tulpa_criteria()` computes that the **loo** package owns no generic for. WAIC and PSIS-LOO have theirs (`loo::waic()`, `loo::loo()`), so a model package registers methods on those rather than on new names that would mask them.

Usage

```
pointwise_loglik(object, ...)

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

## S3 method for class 'tulpa_fit'
pointwise_loglik(object, ndraws = NULL, ...)

dic(object, ...)

## Default S3 method:
dic(object, loglik_at_mean = NULL, ...)

## S3 method for class 'tulpa_fit'
dic(object, ...)

cpo(object, ...)
```

```
## Default S3 method:
cpo(object, ...)

## S3 method for class 'tulpa_fit'
cpo(object, ...)

## S3 method for class 'tulpa_fit'
waic(x, ...)

## S3 method for class 'tulpa_fit'
loo(x, ...)
```

Arguments

`object`, `x`	A pointwise log-likelihood matrix (draws x observations), or a fitted model object a method is registered for, such as a `tulpa_fit`.
`...`	For `dic()` and `cpo()`, passed to `tulpa_criteria()` (e.g. `group`, `chunk_size`). For `waic()` and `loo()`, passed to **loo**'s matrix methods (e.g. `cores`, `save_psis`).
`ndraws`	Number of posterior draws the matrix is evaluated at. Defaults to all stored draws, or 400 on the draw-free Laplace tier; a smaller number subsamples the stored ones. Read by the `tulpa_fit` method of `pointwise_loglik()`, which is the door the criteria below reach the matrix through.
`loglik_at_mean`	Length-`n_obs` vector of pointwise log-likelihoods at the posterior mean of the parameters. Required for DIC's plug-in deviance; without it the DIC fields are `NA`.

Details

`pointwise_loglik()` is the one door onto the `[n_draws x n_obs]` matrix itself, the input every criterion above is computed from. `compare_models()` and `model_average()` call it (through an internal wrapper) rather than assuming the engine's own fit layout, so a model package that registers `pointwise_loglik.<its fit class>()` – alongside its own `waic()` / `loo()` / `dic()` / `cpo()` methods – reaches model comparison and averaging too.

The default methods take a draws x observations pointwise log-likelihood matrix, the same input `tulpa_criteria()` takes. A model package registers a method taking its own fit object, builds the matrix from the posterior, and delegates here.

The `tulpa_fit` methods build that matrix from the fit itself, from the same source `compare_models(criterion = "waic")` reads: a `log_lik` the backend stored with its draws, else the family density evaluated at posterior draws of the in-sample linear predictor (sampler draws, the outer-grid mixture of a nested-Laplace fit, or Gaussian draws at the fixed-effect mode and covariance). The draws are pinned to a fixed internal seed, so repeated calls return the same numbers and leave the session RNG untouched. `dic()` plugs in the posterior mean of the linear predictor, so its `p_dic` counts effective parameters in the linear predictor's parameterization; where the fit stored its `log_lik` and carries no linear predictor draws, the DIC fields are `NA` and `dbar` is still reported. A fit with no pointwise log-likelihood (no stored response, a family that is not one built-in family name, or a linear predictor the fit cannot reproduce) is refused with an error naming the fit's class and the reason.

`loo::waic()` and `loo::loo()` dispatch to the `tulpa_fit` methods once **loo** is loaded, and return **loo**'s own `waic` / `psis_loo` objects, so `loo::loo_compare()` reads them. On an MCMC chain fit `loo()` passes relative effective sample sizes computed over the fit's chains; on an i.i.d. or approximation fit the draws are independent and `r_eff` is 1.

Value

`dic()` and `cpo()` return a `tulpa_criteria` object. `waic()` returns a **loo** `waic` object and `loo()` a **loo** `psis_loo` object.

See Also

`tulpa_criteria()` for every criterion at once and for what the LOO unit means; `compare_models()` to rank several fits.

Examples

```
set.seed(1)
y  <- rnorm(40)
mu <- matrix(rnorm(200 * 40, sd = 0.2), 200, 40)
ll <- dnorm(matrix(y, 200, 40, byrow = TRUE), mean = mu, log = TRUE)
cpo(ll)

d <- data.frame(x = rnorm(120))
d$y <- rpois(120, exp(0.4 + 0.6 * d$x))
fit <- tulpa(y ~ x, data = d, family = "poisson", mode = "laplace")
dic(fit)
cpo(fit)$lpml
if (requireNamespace("loo", quietly = TRUE)) loo::waic(fit)
```

`diagnostics` *Posterior diagnostics for a fitted model*

Description

The diagnostic front door for any tulpa fit. What a fit's posterior draws can be asked depends on how they were produced, so this reads the draws provenance and returns the diagnostic that applies:

MCMC chain draws improved Rhat (the maximum of rank-normalized split-Rhat and folded split-Rhat), bulk / tail / mean / sd / quantile effective sample size, and Monte Carlo standard errors, following Vehtari et al. (2021). Split-Rhat is defined for a single chain, so a result is produced for any number of chains.

i.i.d. approximation draws the approximation-reliability table – the PSIS tail-shape `pareto_k` scored against the exact inner-Laplace marginal and the outer-grid quadrature effective sample size (the OUTER hyperparameter-grid integration layer), the inner-Laplace skewness diagnostic `gamma_3` when computed (the INNER Gaussian approximation to the latent field, a separate layer `pareto_k` does not cover), a combined whole-fit verdict naming which layer degrades when one does, and a per-parameter posterior summary. See the "Approximation reliability" sections below for the full description of this table and its attributes.

point summaries no sample to diagnose; returns NULL with a message naming the backend.

Provenance is read from $draws_kind (stamped by tulpa_dispatch), falling back to the backend registry's emits property and then to an inner $joint_fit. A fit that predates the tag is treated as a chain, so an older fit is never silently refused.

Usage

```
diagnostics(fit, ...)

## Default S3 method:
diagnostics(
  fit,
  pars = NULL,
  measures = c("rhat", "ess_bulk", "ess_tail"),
  probs = c(0.05, 0.95),
  sbc = NULL,
  ...
)

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

## S3 method for class 'sbc'
diagnostics(fit, ...)
```

Arguments

fit	A tulpa_fit (or subclass) carrying posterior $draws. Multiple chains are recognised from a 3D [iter, chain, param] draws array, a $chain_id row map, or an $n_chains count over chain-major rows.
...	Passed to the method.
pars	Optional character vector of parameter names to restrict to.
measures	Character vector selecting which diagnostics to compute, in output-column order. Available: "rhat", "rhat_bulk", "rhat_fold", "ess_bulk", "ess_tail", "ess_mean", "ess_sd", "mcse_mean", "mcse_sd", "ess_quantile", "mcse_quantile". Defaults to the core set c("rhat", "ess_bulk", "ess_tail"). Applies to chain fits; the approximation-reliability table has a fixed set of columns.
probs	Numeric probabilities for the quantile-based measures ("ess_quantile", "mcse_quantile"); each expands to one column named e.g. ess_q5, ess_q95. Default c(0.05, 0.95). Chain fits only.
sbc	Optional sbc() result for the same model, whose calibration verdict is attached to the returned table.
x	A laplace_diagnostics table returned by diagnostics().

Value

For a chain fit, a data frame with a `parameter` column followed by one column per requested measure; entries are `NA` for parameters that are constant or have too few draws. For an i.i.d. approximation fit, the `laplace_diagnostics` table (see the "Approximation-reliability table columns" section below for its attributes). For a point fit, `NULL`.

Extending

`diagnostics()` is an S3 generic so a model package can answer for its own fit class (`diagnostics.ratiod_fit()`, say) rather than shadowing this export with a same-named function. The default method does the provenance routing described above and is what a method should delegate to once it has assembled a draws array.

Calibration alongside reliability

The tables above score ONE fit's own internal reliability. Whether the backend's posterior is CALIBRATED is a different question, answered over many simulated data sets by `sbc()`, and the two disagree in both directions – a fit whose reliability band is clean on both layers can still fail calibration, and one whose outer k-hat is well past the escalation threshold can pass. So the band is a screen, not a verdict. Pass an `sbc` result as `sbc =` and the calibration verdict is attached to the table and printed underneath it; `diagnostics()` called on the `sbc` result itself returns its report table.

Approximation reliability (i.i.d. draws)

Per-parameter reliability diagnostics for a fit whose posterior draws are i.i.d. samples from a deterministic approximation (the nested-Laplace grid-mixture posterior `sum_k w_k N(mode_k, V_k)`), where the between-chain Gelman-Rubin Rhat reported for a chain fit does not apply. This is the table that plays Rhat's role for the deterministic engine: it answers "did the approximation work", not "did the chains mix".

The headline is a Pareto-smoothed importance-sampling (PSIS) reliability diagnostic for the OUTER hyperparameter-grid integration. The nested integrator scores its hyperparameter grid against the exact inner-Laplace marginal posterior with a generalized-Pareto fit to the upper tail of the importance ratios `log p_target(theta) - log q_proposal(theta)` (see `tulpa_psis()`); the resulting tail-shape `pareto_k` is the "did the outer integration work" number – `k-hat < 0.5` good, `0.5-0.7` usable, `>= 0.7` unreliable (Vehtari et al. 2024; Yao et al. 2018). It is computed at fit time and read back here; a fit that did not run the diagnostic, or whose grid proposal degenerated, reports it as `NA` and is assessed on the grid quadrature reliability instead.

`pareto_k` scores the outer integration only. A high `pareto_k` on a fit whose grid quadrature is healthy (`ess_grid` well above 1, largest cell weight modest) flags outer-integration (CI-width) calibration in the right-skewed hyperparameter tail and does not by itself invalidate the point estimates, which the grid quadrature governs.

`outer_regime` qualifies what a high `pareto_k` means, and is the reason a bare threshold on `pareto_k` is not a reliability verdict. A sharp hyperparameter posterior collapses the grid onto ~1 cell (`ess_grid` near 1); the outer integration has then degenerated to a point evaluation at the modal hyperparameter, so `pareto_k` is scoring how well a Gaussian at that mode stands in for the hyperparameter marginal, not how well a grid integrated it. Where the dominant cell is INTERIOR to the grid the collapse is benign – the grid bracketed the mode, the estimate is empirical Bayes there, and only integrated hyperparameter uncertainty is missing. Where it sits at a grid BOUNDARY the grid

may simply be too narrow: `grid_edge_axes` / `grid_edge_sides` name the axes to widen. On a fit whose `pareto_k` cleared the good band, the outer diagnostic also fits a skew-normal proposal and reports the marginal's estimated skewness as `outer_skew_max`, so an inflated k-hat that was purely the symmetric proposal's mismatch with a skewed variance-component marginal is both corrected and explained. A skew-normal has Gaussian tails, so this can never mask a genuinely heavy-tailed target.

The grid quadrature reliability – the effective sample size `ess_grid = 1 / sum(w_k^2)` of the outer integration weights and the largest single cell weight – is always computed from the stored grid: a grid that collapses onto one cell (`ess_grid` near 1) integrates no hyperparameter uncertainty, while a spread grid does.

A SEPARATE layer – the inner Gaussian Laplace approximation to the latent-field conditional posterior `pi(x | theta, y)`, which `pareto_k` does not cover – is scored by `inner_skew`: the leading-order Edgeworth skewness estimate `gamma_3` (Rue, Martino & Chopin 2009 Sec 3.2.3) at the fitted MAP grid cell, computed when `control$diagnose_skew = TRUE` (the default) on the fitting call. Reading a high `pareto_k` alone as "the fit is broken" conflates the two layers: an occu_cover batch flagged 42/78 species "unreliable" on outer k-hat alone when their point estimates, governed by the healthy inner layer, were fine – the `reliability` attribute is the combined verdict that names which layer degrades, if either does.

Each parameter row also carries the rank-normalized split-Rhat and bulk / tail effective sample size of the draws (Vehtari et al. 2021). On i.i.d. draws these sit at `~1.00` and `~n_draws` by construction; they are reported, clearly as i.i.d.-draw Monte-Carlo diagnostics and not chain mixing, to document that the reported posterior summaries are not Monte-Carlo-limited.

A posterior sample is what those per-parameter rows are computed from, and nothing else here needs one: every reliability quantity above is read off the fit. So a fit that carries no draws – a default single-block nested-Laplace fit, whose posterior is the retained outer-grid mixture rather than a sample – reports the full band with an empty per-parameter body and `n_draws = NA`, and records why in the `param_table_declined` attribute. `tulpa_posterior_draws()` samples that mixture where the rows are wanted. The one case that still returns `NULL` is a fit with neither draws nor any reliability quantity, such as a plain Laplace fit with no outer grid to score.

Approximation-reliability scope

The PSIS `pareto_k` diagnoses the OUTER (hyperparameter) integration: whether the Gaussian-proposal-over-grid approximation of the marginal hyperparameter posterior `p(theta | data)` can be importance-corrected to the exact inner marginal. This is the dominant approximation in nested Laplace and the one with an exactly evaluable target. A full latent-space PSIS against the exact joint posterior `pi(x)` is not computed: the latent prior marginal `p(x) = integral p(x | theta) p(theta) dtheta` has no closed form, and for the marginalized-occupancy / cover-hurdle likelihoods the exact joint density is evaluable only inside the C++ kernel, so a stored fit cannot reconstruct it. The grid quadrature reliability is the complementary stored-fit number.

The inner layer carries a SECOND score, `inner_pareto_k`, which needs no likelihood derivative at all and therefore answers where `inner_skew` declines. The inner Gaussian at the fitted hyperparameter is an importance proposal for the exact conditional posterior, and the joint density is the target, so PSIS on that ratio scores the inner approximation directly. It is computed on the same probed indices along the same conditional-mean curve, one dimension per index, since importance sampling degrades with dimension and a k-hat over the whole latent field would report `n_x` rather than the approximation. A Pareto shape index is scale-free, so it is banded only on indices whose realized

importance efficiency shows a correction worth describing; `inner_pareto_k_uniform` records that none did, which is what a well-approximated inner layer looks like.

`inner_skew` diagnoses the INNER (latent-field) Laplace: whether the Gaussian approximation to `pi(x_i | theta, y)` is itself a good fit, at each scored latent index `i`. `gamma_3` is exact for a gaussian-family coefficient (the log-likelihood is exactly quadratic in eta) and declines to `NA` – never a silently-wrong `0` ("perfectly Gaussian") – whenever no per-observation third-derivative oracle is available: a coupled multi-process likelihood (e.g. tulpaObs's `occu_cover`, whose arms combine non-separably through a `CellCouplingSpec`) or a family with no registered third derivative. Only the requested latent indices are scored (every arm's fixed-effects coefficients by default – see `control$skew_idx`), since each index costs one extra linear solve; the full latent field is not scored by default on a large spatial field.

Approximation-reliability table columns

The table has one row per parameter – `parameter`, `mean`, `sd`, `n_draws`, `mcse_mean`, `moment_source` – carrying attributes. There is deliberately no `rhat` / `ess_*` column: those are what the draws-provenance gate withholds on a non-chain fit, and this table is where it dispatches instead. Where the draws were sampled from an outer-grid mixture the fit carries (`attr(draws, "grid_mixture")`, stamped by `tulpa_posterior_draws()` on a single-block fit, by `tulpa_re_cov_nested()` and by model packages), `mean` / `sd` are that mixture's exact moments, `mcse_mean` is 0 and `moment_source` is `"mixture_moments"`; otherwise they are the draw moments, `mcse_mean` is `sd / sqrt(n_draws)` and `moment_source` is `"draws"`.

`pareto_k` the outer PSIS reliability k-hat (`NA` if not computed).

`pareto_k_band` `"good"` / `"ok"` / `"unreliable"` / `NA`.

`pareto_k_declined`, `pareto_k_declined_note` when `pareto_k` is `NA`, WHY: `"not_requested"`, `"not_applicable"`, `"unguessable_axis"` (naming the axis – a permanent limitation of that family, so read `ess_grid` instead), `"draws_too_few"`, `"grid_too_small"`, `"no_varying_axis"`, `"degenerate_proposal"`, or `"internal_inconsistency"` (an engine bug worth reporting), plus a one-line reading of it.

`pareto_k_is_ess` importance-sampling ESS on the smoothed weights.

`pareto_k_ci_low`, `pareto_k_ci_high`, `pareto_k_se_boot`, `pareto_k_band_confident`, `diagnose_draws` the bootstrap interval of the outer k-hat, its standard error, whether the interval sits inside one band, and the draw count it used; present when the fit ran the bootstrap.

`k_quality_requested`, `k_quality_reached`, `k_quality_best`, `k_quality_reason`, `k_quality_rounds` the outcome of a `control$k_quality` request; present when one was made.

`inner_skew`, `inner_skew_idx` the per-index inner-Laplace skewness `gamma_3` and the latent indices it was probed at; present when the probe ran.

`ess_grid`, `n_grid`, `rel_ess_grid`, `max_weight` grid quadrature reliability. `n_grid` counts the cells SOLVED, so a posterior sharp enough to underflow all but one of them reads `ess_grid = 1.00 of 124 cells` rather than of one, and `rel_ess_grid` is the share of the solved grid the quadrature actually uses.

`outer_regime` `"spread"` / `"collapsed_interior"` / `"collapsed_edge"` – whether the outer grid integrated hyperparameter uncertainty at all, and if not whether its dominant cell is interior (empirical Bayes at the mode: point estimates sound, hyperparameter uncertainty not integrated) or against a grid boundary (widen it).

`grid_edge_axes`, `grid_edge_sides` for an edge collapse, the axes the dominant cell sits against and on which side.

`grid_edge_mass_axes` axes holding material weight on one of their own boundary nodes, as `axis:side` – that axis truncates its own marginal there whether or not its mode sits on the node, which is the weaker and more common statement `grid_railed_axes` cannot make. Read in every regime, a spread grid included.

`outer_skew_max` largest estimated |skewness| of the hyperparameter marginal, computed only when the k-hat triggered the skew-normal proposal rescue (`NA` means the Gaussian proposal already fit, not "symmetric and unchecked").

`outer_regime_note` a one-line reading of a collapsed regime or of boundary mass on an axis, absent on a spread grid carrying neither.

`grid_railed_axes` outer axes whose OWN marginal is maximal at one of their own endpoints, as `axis:side` – the span does not contain that axis's posterior mode, so its marginal is a truncated tail at any spacing. Reported whether or not the engine was allowed to, able to, or built to move the axis.

`grid_placement`, `grid_recentred_axes`, `grid_placement_declined`, `grid_placement_note` whether the outer grid was re-centred, on which axes, and – when it was not – why.

`scope` the outer diagnostic's scope string.

`scope_backend`, `scope_title`, `scope_layers`, `scope_text` what the table describes: the backend it was read for, the header title, the reliability layers that backend has (`c("outer", "inner")` for a nested-Laplace fit, empty for a backend with neither), and the scope sentence – for a nested fit what its outer layer integrates over, for any other backend what its draws are and which reliability quantity it computes. A backend without the two layers has no whole-fit verdict, and its `reliability` is `NA`.

`inner_skew_max` the largest `|gamma_3|` among the scored latent indices (`NA` if `control$diagnose_skew = FALSE` or nothing scored).

`inner_skew_band` `"good"` / `"ok"` / `"unreliable"` / `NA`, banded on `inner_skew_max` by the general skewness-magnitude convention (Bulmer 1979) – not a Rue-Martino-Chopin-specific cutoff.

`inner_skew_scored`, `inner_skew_probed` how many of the probed latent indices returned a finite `gamma_3` vs how many were probed.

`inner_skew_declined`, `inner_skew_arms_declined`, `inner_skew_declined_note` when nothing was scored, WHY: `"coupled_arm"` (STRUCTURAL – the coupled arms have neither a per-observation sum nor a cell third-derivative tensor to read, so the outer k-hat is the only reliability number this fit has), `"curvature3_unavailable"`, `"no_finite_contribution"`, `"no_probe_indices"`, `"not_requested"`, `"backend_unsupported"`, `"solve_failed"`, or `"not_converged"` (the probe re-solve stopped short of a mode, so neither inner score has a point to read); the arms (1-based) a joint fit had no oracle for, which is also set on a PARTIALLY scored fit; and a one-line reading.

`inner_pareto_k`, `inner_pareto_k_band` the inner-Laplace importance k-hat over the probed subspace, and its band on the same convention as the outer k-hat. Available wherever a mode was found, including a fit `gamma_3` cannot score.

`inner_pareto_k_rel_ess`, `inner_pareto_k_is_ess` the smallest realized importance efficiency and effective sample size across the probed indices – how much correcting the inner Gaussian actually needs, which is what makes the scale-free shape above readable.

`inner_pareto_k_uniform` `TRUE` when no probed index carried a material correction, i.e. the inner Gaussian reproduces the conditional posterior over the sampled region.

`inner_pareto_k_scored`, `inner_pareto_k_probed` how many probed indices returned a finite k-hat vs how many were probed.

`inner_pareto_k_declined`, `inner_pareto_k_declined_note` when it is `NA`, WHY, from the same closed vocabulary the outer k-hat uses.

`reliability` the combined whole-fit verdict: `"reliable"` only when both layers are good; otherwise names which layer is scoped or flags both as unreliable. The inner layer enters through the worse of its two scores, so a fit whose cubic term declined is still assessed.

and a trailing `summary` attribute (a one-row data frame of the headline numbers) for printing.

References

Vehtari, Gelman, Simpson, Carpenter & Burkner (2021). Rank-normalization, folding, and localization: an improved Rhat for assessing convergence of MCMC. *Bayesian Analysis* 16(2):667-718.

Vehtari, Simpson, Gelman, Yao & Gabry (2024). Pareto smoothed importance sampling. *JMLR* 25(72):1-58.

Yao, Vehtari, Simpson & Gelman (2018). Yes, but did it work?: Evaluating variational inference. *ICML*, PMLR 80:5581-5590.

Rue, Martino & Chopin (2009). Approximate Bayesian inference for latent Gaussian models by using integrated nested Laplace approximations. *JRSS-B* 71(2):319-392.

See Also

`sbc()` for calibration, the approximation-reliability sections above for the i.i.d.-fit table in full, `tulpa_draws_array()`, `tulpa_posterior_draws()`, `tulpa_psis()`, `plot_rhat()`, `plot_ess()`, `diagnostic_summary()`, `check_diagnostics()`

Examples

```
set.seed(1)
df <- data.frame(x = rnorm(60))
df$y <- rpois(60, exp(0.5 + 0.3 * df$x))

# chain fit -> Rhat / ESS
hmc <- tulpa(y ~ x, data = df, family = "poisson", mode = "hmc",
             control = list(n_iter = 500L, warmup = 250L, n_chains = 2L,
                            seed = 1L))
diagnostics(hmc)

# deterministic fit -> PSIS approximation reliability
smc <- tulpa(y ~ x, data = df, family = "poisson", mode = "smc")
diagnostics(smc)
```

diagnostic_summary *Comprehensive diagnostic summary*

Description

Provides a comprehensive diagnostic report for a tulpa model, combining convergence metrics, divergence information, and actionable recommendations.

Usage

```
diagnostic_summary(fit, quiet = FALSE)

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

Arguments

`fit`	A `tulpa_fit` object.
`quiet`	Logical; if TRUE, suppress printed output (default: FALSE).
`x`	A `tulpa_diagnostic_summary` object.
`...`	Ignored.

Value

A list with class `tulpa_diagnostic_summary` containing:

status Overall status: "PASS", "WARN", or "FAIL"

n_divergent Number of divergent transitions

divergent_pct Percentage of divergent transitions

worst_rhat Data frame of parameters with worst Rhat

worst_ess Data frame of parameters with worst ESS

e_bfmi E-BFMI value (HMC only)

pareto_k, quad_ess approximation fits only: the outer PSIS k-hat, or the grid quadrature ESS when no k-hat was produced

hyper_share_min, hyper_share_max approximation fits only: the range over coefficients of the share of the fixed-effect marginal variance the hyperparameter integration contributed, `between / (within + between)` under the law of total variance over the outer grid. Near zero means the reported interval is what conditioning on a single hyperparameter point would have given. Reported without a threshold and never scored: a small share is not by itself a miscalibrated interval. `NA` when the fit retains no per-cell pieces to decompose

pareto_k_declined approximation fits only, and only when there is no k-hat: WHY – `"not_requested"` and `"unguessable_axis: <axis>"` are benign or permanent, `"degenerate_proposal"` and `"grid_too_small"` are signals about the fit, and `"internal_inconsistency"` is an engine bug and raises the status to `"WARN"`

inner_skew_max, inner_skew_declined approximation fits only: the largest scored inner-Laplace `|gamma_3|`, or why nothing was scored (`"coupled_arm"` marks arms the inner layer could score neither per observation nor through the cell tensor)

axis_fields_dropped data frame of grid axes the fit's own resolved path could not read and dropped as engine defaults: one row per field, with the block, family, path and the axis that path integrated instead. Absent whenever every supplied axis was used

recommendations Character vector of recommendations

See Also

`check_diagnostics()`, `diagnostics()`, `plot_diagnostics()`

Examples

```
set.seed(123)
df <- data.frame(x = rnorm(60))
df$y <- rpois(60, exp(0.5 + 0.3 * df$x))
fit <- tulpa(y ~ x, data = df, family = "poisson", mode = "hmc",
             control = list(n_iter = 500L, warmup = 250L, n_chains = 2L,
                            seed = 1L))
ds <- diagnostic_summary(fit)
print(ds)
```

`durbin_watson` *Durbin-Watson test for temporal autocorrelation*

Description

Tests first-order autocorrelation in temporally-ordered residuals.

Usage

```
durbin_watson(
  object,
  alternative = c("two.sided", "greater", "less"),
  time = NULL,
  resid_type = "pearson"
)
```

Arguments

`object`	A fitted model, or a numeric vector of temporally-ordered residuals. A fit's residuals are in its data's row order, so pass `time` unless the rows already are in time order.
`alternative`	`"two.sided"`, `"greater"` (positive autocorr), or `"less"`

time	Optional time index, one per residual; the residuals are put in increasing time order before the statistic is formed. NULL (default) takes them in the order given.
resid_type	Residual type if extracting from a model (default "pearson"), as for moran_i().

Value

An htest object with DW statistic, lag-1 r, and p-value

Examples

```
set.seed(1)
e <- as.numeric(arima.sim(list(ar = 0.6), n = 100))
durbin_watson(e)
# Residuals recorded out of time order:
tt <- sample(100)
durbin_watson(e[tt], time = tt)
```

---

energy_bfmi *Energy Bayesian Fraction of Missing Information (E-BFMI)*

Description

Computes E-BFMI (Betancourt 2016) from the per-iteration Hamiltonian energies an HMC or NUTS sampler records: the variance of the energy transition between successive iterations over the variance of the marginal energy. Each chain is diffed separately, since a transition across a chain boundary compares unrelated trajectories, and the lowest chain value is reported: one badly exploring chain is enough to distrust a fit. Values below 0.3 indicate that momentum resampling moves the sampler too little between energy level sets.

Usage

```
energy_bfmi(energy, chain = NULL)
```

Arguments

energy	Numeric vector of per-iteration energies.
chain	Chain index of each entry of energy. NULL (default), or a vector of the wrong length, treats energy as one chain.

Value

A list with value (the lowest per-chain E-BFMI, NA when no chain has two finite, non-constant energies), per_chain, energy_centred (energies centred within each chain) and energy_diff (within-chain transitions), the last two pooled across chains for plotting.

Examples

```
set.seed(1)
energy <- c(cumsum(rnorm(200, sd = 0.3)), rnorm(200, sd = 1))
energy_bfmi(energy, chain = rep(1:2, each = 200))$per_chain
```

`family_names` *Supported R-level family names*

Description

The response families `family =` accepts on the engine's front doors, in their canonical (default-link) spelling. A family's `*_<link>` variants (e.g. `"gamma_inverse"`) are accepted as well and not listed here.

Usage

```
family_names()
```

Value

A character vector of family names.

Examples

```
family_names()
```

`field_effects` *Spatial field effects per unit*

Description

The posterior of a fit's spatial field, one row per spatial unit: an areal region (a row of the adjacency matrix) or an SPDE mesh node. The value reported is the field's contribution to the linear predictor at that unit, the quantity a covariate is read against.

Usage

```
field_effects(object)
```

Arguments

`object`	A `tulpa_fit` carrying a spatial field.

Details

Where the posterior comes from follows what the fit computes:

- A ModelData sampler fit (`mode = "hmc"` and its siblings) with an ICAR, BYM2 or proper-CAR field: every draw's field contribution is evaluated through the engine's own linear-predictor assembly, so an intrinsic field is centred as its density centres it and a BYM2 field is the combined `sigma * (sqrt(rho) * s * phi + sqrt(1 - rho) * theta)` at that draw's `sigma` and `rho` (`s` the scaling factor). A unit that no observation falls in contributes nothing to the linear predictor and is reported as `NA`.
- A Polya-Gamma Gibbs fit (`mode = "gibbs"`) stores the field it samples (`phi_spatial[k]`); for BYM2 each draw's two unit-scale components are combined into the same `sigma * (sqrt(rho) * s * phi + sqrt(1 - rho) * theta)` the sampler path reports. The rows summarize those draws.
- A nested-Laplace fit with an ICAR or proper-CAR field: each outer cell contributes its conditional mode, and its marginal variance where the fit retained one, weighted by the cell's posterior weight. With the variances the summary is the exact mean, SD and quantiles of that Gaussian mixture (`source = "mixture"`); without them the weighted mean of the cell modes, with no spread (`source = "mode"`). A BYM2 field is reported where the fit stores its grid-weighted combined field (`fit_st_nested()`); a `tulpa()` nested BYM2 fit carries its two components with cell-dependent mixing and errors.
- A conditional-Laplace areal fit (`mode = "laplace"` / `"eb"`) reports the field at its mode, with no spread. For BYM2 this is the combined field at the `sigma = 1`, `rho = 0.5` the conditional kernel fixes.
- An SPDE fit (`fit_spde()`) reports the mesh-node field: its draws on the sampler path, its mode (the posterior-mean hyperparameters' fit on the nested path) otherwise.

A fit whose field cannot be located this way (a continuous GP / NNGP / HSGP field, a restricted field, an inline `spatial()` varying-coefficient field) errors naming the field and the backend, as does a fit with no spatial field. A temporal field is read by `temporal()`.

Value

A data frame with one row per spatial unit: `unit` (the node index), `label` (the adjacency row name where it has one, else the index), `estimate`, `sd`, the 2.5% / 97.5% bounds `conf.low` / `conf.high`, and `source`: `"sampled"` for a posterior draw summary, `"mixture"` for a nested-Laplace Gaussian mixture, `"mode"` for a point at the mode (`sd` and the bounds `NA`). The field type is attached as attribute `"field"`.

See Also

`smooth_effects()` for an `s(x)` term, `ranef()` for random effects, `temporal()` for a temporal field.

Examples

```
set.seed(1)
K <- 12
W <- matrix(0, K, K)
```

```
for (i in 1:K) { j <- if (i < K) i + 1 else 1; W[i, j] <- W[j, i] <- 1 }
region <- sample(1:K, 200, replace = TRUE)
d <- data.frame(region = region, x = rnorm(200))
d$y <- rpois(200, exp(0.3 + 0.5 * d$x + sin(2 * pi * region / K)))
fit <- tulpa(y ~ x + spatial(region), data = d, family = "poisson",
             spatial = list(type = "icar", adjacency = W), mode = "laplace")
fe <- field_effects(fit)
plot(fe$unit, fe$estimate, type = "b")
```

`fitted.tulpa_fit` *Fitted values (population level)*

Description

In-sample mean response from the fixed effects and the observation offset (`E[y] = g^{-1}(X beta + offset)`, trial-scaled for binomial). Random effects are held at their prior mean of zero; group-level effects are in `ranef()`. `y - fitted(object)` equals `residuals(object, type = "response")`. On a zero-inflated fit (`ziformula`) the mean is the mixture's, `(1 - pi) E[y | eta]` with `pi = plogis(X_zi beta_zi)`; over a zero-truncated family that is the hurdle mean.

Usage

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

Arguments

`object`	A `tulpa_fit` object (must carry `$model_matrix`).
`...`	Ignored.

Value

Numeric vector of fitted mean responses, length `nobs`.

`fit_spde` *Fit a spatial model using SPDE Laplace approximation*

Description

Fits a GLM with a Matern spatial field via the SPDE approach. Uses CHOLMOD sparse solver with optional nested Laplace for hyperparameter integration.

Usage

```
fit_spde(
  y,
  X,
  spatial,
  family = "binomial",
  n_trials = NULL,
  range = NULL,
  sigma = NULL,
  nested_laplace = is.null(range) || is.null(sigma),
  phi = 1,
  offset = NULL,
  re_idx = NULL,
  n_re_groups = 0L,
  sigma_re = 1,
  mode = c("laplace", "nuts"),
  hyperprior = c("proper", "flat"),
  control = list()
)
```

Arguments

`y`	Integer response vector.
`X`	Design matrix.
`spatial`	A `tulpa_spatial` object from `spatial_spde()` or `spatial_spde_custom()`.
`family`	Distribution family: `"binomial"`, `"poisson"`, `"neg_binomial_2"`, or `"gaussian"` (continuous-field geostatistics, with `phi` the observation-noise standard deviation).
`n_trials`	Integer vector of trial sizes (binomial only).
`range`	Spatial range parameter. If NULL, uses nested Laplace to integrate over range and sigma.
`sigma`	Marginal standard deviation. If NULL, uses nested Laplace.
`nested_laplace`	Logical. If TRUE (default when range/sigma are NULL), use nested Laplace approximation over hyperparameters.
`phi`	Dispersion passed to the family, held fixed. One convention at every door: for `gaussian` / `lognormal` this is the residual VARIANCE (the SD is `sqrt(phi)`), for `neg_binomial_2` the size, `gamma` the shape, `beta` the precision, `t` the scale; `binomial` and `poisson` ignore it. The compiled kernels parameterize the two variance families by the residual SD and are handed `sqrt(phi)` at the boundary. Defaulted, it conditions at 1 and says so with a warning, since for a family that reads a dispersion that is a modelling choice rather than a neutral value. `fit$phi_estimated` records whether the value on the fit was estimated or conditioned on.
`offset`	Optional fixed additive term on the linear predictor (`eta = offset + X beta + A w`), length `length(y)`; `NULL` -> no offset.

re_idx, n_re_groups, sigma_re

Optional single iid random-intercept (1 | g) term alongside the Matern field: re_idx is a length-length(y) 1-based group index, n_re_groups the number of groups, and sigma_re the (conditioned) random-effect SD. The field and the RE block are Laplace- marginalised jointly. n_re_groups = 0 (default) is no RE term. Not supported for a fractional-nu field.

mode

Inference method (the method is an argument, not a parallel verb): "laplace" (default) is the nested-Laplace integration over (range, sigma) documented here; "nuts" delegates to tulpa_nuts_spde() for exact HMC over the field (and, unless both range and sigma are fixed, the Matern hyperparameters). mode = "nuts" does not support an offset or a random-effect term, and its sampler knobs pass via control (see tulpa_nuts_spde()); it returns that sampler's draws object.

hyperprior

"proper" (default) or "flat", the prior the nested integration puts on (range, sigma). "proper" carries the spec's PC range and PC sigma priors. "flat" keeps only the anchors the spec was given explicitly (prior_range / prior_sigma passed to spatial_spde()); an axis whose anchor was defaulted carries no density, is named in log_hyperprior_declined as "flat_hyperprior", and the evidence declines. Not read by mode = "nuts", where "flat" errors.

control

A named list of numerical / tuning knobs (statistical arguments stay in the signature above). Recognized entries:

- method: hyperparameter integration backend when nested Laplace is active. "ccd" (default) uses a central composite design centered on the joint posterior mode of (range, sigma), oriented by the local Hessian (9 design points instead of n_grid^2), folding the PC priors from spatial$prior_range / spatial$prior_sigma into the integrated marginal and falling back to "grid" if the surface is too flat for a Hessian-based design. "grid" uses a rectangular grid in log(range) x log(sigma) around the prior modes.
- n_grid: grid points per hyperparameter dimension for method = "grid" (ignored under "ccd"). Default 5.
- diagnose_k: if TRUE (default), compute the outer Pareto-$\hat{k}$ accuracy diagnostic ($pareto_k) by importance sampling the joint (range, sigma) posterior on the log scale against the Gaussian proposal that orients the integration. See tulpa_psis().
- k_samples: importance draws for diagnose_k. Default 500, each one extra batched SPDE marginal evaluation. It is a precision knob: the GPD tail size is held at the fraction the default budget implies, so a larger budget sharpens the same k-hat rather than moving it to a deeper quantile of the weight distribution (gcol33/tulpa#631).
- k_tail_points: expert override for that tail size, in upper-tail order statistics. Silently capped at 20% of the draws, beyond which body ratios enter the tail and bias the shape.
- mode_find: tuning for the outer (range, sigma) mode-find under method = "ccd", as list(factr =, ndeps =, maxit =); supply any subset. ndeps is the central-difference step for optim()'s numerical gradient on the log scale (default 1e-2): it must clear the inner solver's convergence tolerance, and a step wide enough that its truncation error exceeds the reduc-

tion the line search chases near a flat optimum leaves L-BFGS-B aborting at the mode it just reached, in which case the CCD design declines to the rectangular grid. `factr` is the relative-reduction stop in units of `.Machine$double.eps` (default 1e5); `maxit` the iteration cap (default 300).

- `max_iter`: maximum Newton iterations. Default 100.
- `tol`: Newton convergence tolerance. Default 1e-6.
- `n_threads`: OpenMP threads. Default 1.
- `checkpoint`: grid-cell checkpoint/resume spec `list(path =, resume =)`. Each solved `(range, sigma)` cell is appended to `path`; a `resume = TRUE` run loads the finished cells and re-solves only the rest, so a killed or rebooted fit resumes instead of restarting. `resume = FALSE` starts a fresh file. Default `NULL` (off).

Value

A list with:

- `mode`: mode of the latent field (beta + mesh node effects)
- `log_marginal`: log marginal likelihood
- `converged`: convergence flag
- `spatial`: the spatial specification (for prediction)
- `pareto_k`, `pareto_k_is_ess`: outer Pareto-$\hat{k}$ and its importance-sampling ESS (`NA` when `diagnose_k = FALSE`)
- `pareto_k_proposal_source`, `pareto_k_first_pass`: which proposal family the reported k-hat came from (several candidates are scored and the best kept), and the k-hat of the first pass – the proposal exactly as placed, before refinement. A large gap says the placement is poor even where the verdict is fine.
- `nested`: nested Laplace results (if used)

References

Lindgren, Rue & Lindstrom (2011). An explicit link between Gaussian fields and Gaussian Markov random fields: the stochastic partial differential equation approach. *JRSS-B* 73(4):423-498. Rue, Martino & Chopin (2009). Approximate Bayesian inference for latent Gaussian models by using integrated nested Laplace approximations. *JRSS-B* 71(2):319-392.

Examples

```
if (requireNamespace("fmesher", quietly = TRUE)) {
  set.seed(1)
  n <- 200L
  coords <- cbind(runif(n), runif(n))
  mesh <- fmesher::fm_mesh_2d(loc = coords, max.edge = c(0.15, 0.4), cutoff = 0.05)
  fem  <- fmesher::fm_fem(mesh)
  A    <- as(fmesher::fm_basis(mesh, loc = coords), "CsparseMatrix")
  spec <- spatial_spde_custom(C = fem$c0, G = fem$g1, A = A, nu = 1,
                              prior_range = c(0.3, 0.5), prior_sigma = c(0.6, 0.05))
  w <- as.numeric(rnorm(spec$n_mesh, 0, 0.6)); w <- w - mean(w)
```

```
  x <- rnorm(n)
  y <- rpois(n, exp(2.0 + 0.5 * x + as.numeric(spec$A %*% w)))
  fit <- fit_spde(y = y, X = cbind(1, x), spatial = spec, family = "poisson")
  fit$nested$range_mean
}
```

`fit_st_nested` *Fit an additive spatiotemporal GLM by nested Laplace*

Description

Fits `y ~ X beta + u_spatial[s] + v_temporal[t]` with a spatial field – areal (`icar` / `bym2` / `car_proper`) or continuous (`hsgp` / `nngp`) – and a temporal field (`rw1` / `rw2` / `ar1`), integrating the spatial hyperparameter(s), temporal precision, and (for `ar1`) the temporal autocorrelation over a hyperparameter grid via the `cpp_nested_laplace_st_*` kernels. The fixed-effect posterior is the grid-marginalised mixture; the spatial and temporal field posterior means are the grid-weighted latent modes.

Usage

```
fit_st_nested(
  y,
  X,
  spatial_idx,
  adjacency,
  temporal_idx,
  n_times,
  spatial_type = c("icar", "bym2", "car_proper", "hsgp", "nngp"),
  temporal_type = c("ar1", "rw1", "rw2"),
  family = "binomial",
  n_trials = NULL,
  phi = 1,
  cyclic = FALSE,
  re_idx = NULL,
  n_re_groups = 0L,
  sigma_re = 1,
  hyperprior = c("proper", "flat"),
  control = list(),
  coords = NULL,
  nn = 10L,
  cov_type = 2L,
  hsgp_m = 6L,
  hsgp_c = 1.5
)

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

Arguments

- `y` — Response vector.
- `X` — Fixed-effects design matrix (`nrow(X) == length(y)`).
- `spatial_idx` — Integer per-observation spatial-unit index (1-based). For `spatial_type = "icar"/"bym2"/"car_proper` an areal unit in `[1, nrow(adjacency)]`; for `"nngp"`, a location in `[1, nrow(coords)]`. Ignored for `"hsgp"` (the field is evaluated directly at each observation's own coordinates); pass any placeholder (e.g. `seq_len(N)`).
- `adjacency` — Spatial adjacency (a symmetric 0/1 matrix or `sparseMatrix`). Required for `spatial_type = "icar"/"bym2"/"car_proper"`; ignored (pass `NULL`) for `"hsgp"/"nngp"`, which take `coords` instead.
- `temporal_idx` — Integer per-observation time index (1-based).
- `n_times` — Number of distinct time points.
- `spatial_type` — `"icar"` (default), `"bym2"`, `"car_proper"`, `"hsgp"` (Hilbert-space GP basis), or `"nngp"` (nearest-neighbour GP).
- `temporal_type` — `"ar1"` (default), `"rw1"`, or `"rw2"`.
- `family` — Response family (see `family_names()`).
- `n_trials` — Binomial denominators, or `NULL` (= 1).
- `phi` — Dispersion passed to the family, held fixed. One convention at every door: for `gaussian` / `lognormal` this is the residual VARIANCE (the SD is `sqrt(phi)`), for `neg_binomial_2` the size, `gamma` the shape, `beta` the precision, `t` the scale; `binomial` and `poisson` ignore it. The compiled kernels parameterize the two variance families by the residual SD and are handed `sqrt(phi)` at the boundary.

  Defaulted, it conditions at 1 and says so with a warning, since for a family that reads a dispersion that is a modelling choice rather than a neutral value. `fit$phi_estimated` records whether the value on the fit was estimated or conditioned on.
- `cyclic` — Logical; wrap the temporal field (seasonal). Default `FALSE`.
- `re_idx, n_re_groups, sigma_re` — Optional single iid random-intercept term alongside the fields (conditioned on `sigma_re`); `n_re_groups = 0` (default) is no RE term.
- `hyperprior` — The prior an outer hyperparameter axis carries when the call states no density for it, `"proper"` (default) or `"flat"`. `"proper"` folds a normalised density on the axis's integration coordinate into `log_marginal`: the PC prior `P(sigma > 3) = 0.01` on a standard deviation, variance or precision axis, the PC range prior at 0.2 times the coordinates' bounding-box diagonal on a range or lengthscale axis, a uniform on a bounded axis, and the PC + LKJ prior over a free-covariance block. An axis with no sourced density is named in `log_hyperprior_declined`. `"flat"` folds no density of the engine's own, so such an axis is integrated under its cell measure alone, is named in `log_hyperprior_declined` as `"flat_hyperprior"`, and the fit's `log_evidence` declines with `"improper_hyperprior"`. A density the call states – a `prior_*` argument, a block's `rho_prior`, `prior_range` or `prior_sigma`, the copy coefficient's slab, a `tgmrf()` block's own prior – applies under either choice. The same two choices, under the same names, are offered by `tulpa()`, `tulpa_eb()`, `tulpa_re_cov_nested()` and `fit_spde()`.

control

A list of numerical / grid knobs: n_grid_spatial, n_grid_temporal (default 4 each), n_grid_rho (ar1 only, default 3), tau_lower / tau_upper (icar / car_proper precision grid bounds, default 0.25 / 16), sigma_lower / sigma_upper (bym2's spatial SD grid bounds in place of tau_lower / tau_upper, default 0.1 / 3 – n_grid_spatial sizes this axis too; its mixing-weight axis rho_spatial is always the fixed default node set and is not a control knob), rho_lower / rho_upper (ar1 grid, default 0.1 / 0.9), max_iter, tol, n_threads, auto_recenter (default TRUE; FALSE holds the grid exactly as specified – the per-axis policy names tulpa_nested_laplace() takes are refused here with an error, since this driver recentres on the grid's collapsed-edge regime rather than on a per-axis rail; declines outright for spatial_type = "bym2", whose (sigma, rho) spatial axes this recenter has no transform for yet, and for "hsgp"/"nngp", same reason), rho_spatial (the proper-CAR mixing value the car_proper axis is held at, default .NL_ST_GRID$rho_spatial; unrelated to bym2's own integrated rho_spatial grid axis) and within_cell ("log_quadratic" / "box_uniform" / "chord", the within-cell construction the reported per-axis intervals are read with; defaults to .NL_DIAG$within_cell, as on every other nested door).

For spatial_type = "hsgp"/"nngp", the spatial axes are the field variance (sigma2) paired with the lengthscale (lengthscale) or NNGP range (phi_gp), read off the same shared default bounds the single-field spatial_gp() path uses (gp_var / gp_lengthscale, R/settings.R) – there is no separate sigma2_lower/upper knob here, only n_grid_spatial, which sizes the pair as it does for the areal families.

The (tau_lower, tau_upper) span (and, for ar1, (rho_lower, rho_upper)) is a starting axis, not a hard ceiling: when the fitted precision (or, for ar1, autocorrelation) posterior mode rails a boundary node (that axis's own marginal is maximal there, or the whole grid collapsed onto it: pareto_k_regime = "collapsed_edge", see below), the driver fits a mode-Hessian via a derivative-free optim() over the grid and refits a grid re-centred on it (one attempt).

A grid knob PINS the axes it shapes, and a pin always wins – but pinning is decided by value, not by presence: a knob set to the engine's own default, or marked with auto_grid(), expresses no preference and leaves its axes free. That is what lets a wrapper package thread its own n_grid-style argument through control without silently disabling the recenter for every fit it makes. Pinning is also per axis: tau_lower / tau_upper hold the two precision axes, n_grid_spatial / n_grid_temporal one each, and n_grid_rho / rho_lower / rho_upper the ar1 autocorrelation axis, so pinning one axis leaves the others free to be recentred. A pinned axis keeps its nodes exactly and is named in outer_grid_pinned_axes; with EVERY axis pinned the recenter declines outright and outer_grid_recenter_declined records which reason applied.

coords

Coordinate matrix for a continuous spatial field, required when spatial_type is "hsgp" or "nngp" (ignored otherwise). For "hsgp", an N x 2 matrix (one row per observation, matching spatial_gp(approx = "hsgp")'s basis convention – the basis is built by cpp_hsgp_basis_2d(), 2D only). For "nngp", an n_spatial x d matrix of unique locations that spatial_idx indexes into (any d, matching spatial_gp()'s NNGP convention).

nn

Number of nearest neighbours per location, spatial_type = "nngp" only. Default 10 (clamped to nrow(coords) - 1).

cov_type	Integer NNGP covariance code (spatial_type = "nngp" only): 0 = exponential, 1 = Matern 3/2, 2 = Matern 5/2 (default), 3 = Gaussian.
hsgp_m, hsgp_c	Hilbert-space GP basis size (per dimension, default 6) and boundary factor (default 1.5), spatial_type = "hsgp" only – the same parameterisation and defaults as spatial_gp(approx = "hsgp").
x	A fit_st_nested() fit.
...	Ignored.

Value

A tulpa_fit (subclass tulpa_nested_laplace) carrying the fixed-effect posterior (draws via the grid mixture), spatial_effects, temporal_effects, log_marginal, weights, theta_grid over (tau_spatial, tau_temporal, rho) for an areal spatial field or (sigma2, lengthscale, tau_temporal, rho) / (sigma2, phi_gp, tau_temporal, rho) for "hsgp" / "nngp", and family / n_trials / phi (the R-level convention), which is what fitted(), residuals(), posterior_predict() and simulate() read the response family from. Also carries pareto_k_regime ("spread" / "collapsed_interior" / "collapsed_edge", see tulpa_nested_laplace_joint()'s return docs for the definition) and outer_grid_placement ("fixed" or "auto_recentered") plus, on a "fixed" placement, outer_grid_recenter_declined ("grid_knobs_overridden" / "grid_not_collapsed" / "no_usable_curvature" / "refit_failed" / "sd_ceiling_unresolved" / "sd_floor_unresolved"). A recentred fit also carries outer_grid_pinned_axes, the axes whose knobs were pinned and whose nodes were therefore kept, and outer_grid_recenter_sd_clamp / _sd_raw / _sd_used – per moved axis, which mode-SD bound the placement hit, the SD the stencil measured, and the SD the axis was laid from. A bound-decline is PER AXIS here: the axes the mode-find did resolve are still re-placed, and outer_grid_recenter_sd_declined names the ones that kept their incoming nodes and on which bound, so a partially re-placed grid is not read as a fully re-placed one. With every free axis declined the pass reports the grid as the fixed one it still is.

See Also

tulpa() (front door), tulpa_nested_laplace() (single field).

Examples

```
set.seed(1)
n_s <- 16L; n_t <- 8L; N <- 400L
adj <- matrix(0, n_s, n_s)
for (i in 1:(n_s - 1)) adj[i, i + 1] <- adj[i + 1, i] <- 1
s <- sample(n_s, N, TRUE); tt <- sample(n_t, N, TRUE)
us <- as.numeric(scale(cumsum(rnorm(n_s)))); vt <- as.numeric(scale(cumsum(rnorm(n_t))))
x <- rnorm(N)
y <- rbinom(N, 1, plogis(0.2 + 0.5 * x + 0.7 * us[s] + 0.6 * vt[tt]))
fit <- fit_st_nested(y, cbind(1, x), s, adj, tt, n_t, family = "binomial")
```

`fixef` *Fixed-effect coefficients (lme4-compatible)*

Description

The fixed-effect point estimates, equivalent to `coef()` on a `tulpa_fit`. Provided under the `fixef` name for code written against the `lme4` / `nlme` interface: `lme4::fixef(fit)` and `nlme::fixef(fit)` dispatch here too when either package is installed, so a `tulpa_fit` can be dropped into an `lme4`-shaped workflow.

Usage

```
fixef(object, ...)

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

Arguments

`object`	A `tulpa_fit` object.
`...`	Ignored.

Details

Note the deliberate difference from `lme4::coef.merMod`, which returns per-group sums of the fixed and random effects. On a `tulpa_fit`, `coef()` returns the fixed effects alone and `fixef()` is its synonym; the random effects are `ranef()`.

Value

Named numeric vector of fixed-effect estimates.

See Also

`coef.tulpa_fit()`, `ranef()`

Examples

```
set.seed(1)
df <- data.frame(x = rnorm(80))
df$y <- rpois(80, exp(0.5 + 0.3 * df$x))
fit <- tulpa(y ~ x, data = df, family = "poisson")
fixef(fit)
```

geweke_test *Geweke convergence test*

Description

Performs Geweke's convergence diagnostic, comparing the mean of the first portion of a chain to the last portion. It is a within-chain diagnostic, so on a multi-chain fit it is computed for each chain separately.

Usage

```
geweke_test(fit, frac1 = 0.1, frac2 = 0.5, pars = NULL)

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

Arguments

`fit`	A `tulpa_fit` object.
`frac1`	Fraction of each chain for the first window (default: 0.1).
`frac2`	Fraction of each chain for the second window (default: 0.5). Both lie in `(0, 1)` and the windows may not overlap: `frac1 + frac2 <= 1`.
`pars`	Character vector of parameter names (default: all main params).
`x`	A `tulpa_geweke` result.
`...`	Ignored.

Details

The Geweke test computes a z-score comparing the means of early and late portions of a chain. Large z-scores ($|z| > 2$) indicate the chain has not converged. The chains are never pooled: concatenated, the "first" window would come from the first chain and the "last" from another, which compares two chains rather than a chain's start with its end (use `diagnostics()` for the between-chain R-hat). A parameter that is constant within a chain has an undefined z-score, reported as `NA`.

Value

A data frame of class `tulpa_geweke` with columns `chain`, `parameter`, `z_score` and `p_value`, one row per chain and parameter.

See Also

`diagnostics()`, `check_diagnostics()`

Examples

```
set.seed(123)
df <- data.frame(x = rnorm(60))
df$y <- rpois(60, exp(0.5 + 0.3 * df$x))
fit <- tulpa(y ~ x, data = df, family = "poisson", mode = "hmc",
             control = list(n_iter = 500L, warmup = 250L, n_chains = 2L,
                            seed = 1L))
geweke_test(fit)
```

---

glance.tulpa_fit *Model-level summary statistics (broom-compatible)*

Description

Model-level summary statistics (broom-compatible)

Usage

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

Arguments

x	A `tulpa_fit` object.
...	Ignored.

Value

Single-row data frame.

Examples

```
set.seed(1)
df <- data.frame(x = rnorm(100))
df$y <- rpois(100, exp(0.3 * df$x))
fit <- tulpa(y ~ x, data = df, family = "poisson")
glance(fit)
```

`imh_laplace` *Independence Metropolis-Hastings with Laplace proposal*

Description

Sample from a posterior using independence Metropolis-Hastings with a multivariate-normal proposal centred at the Laplace mode and scaled by the inverse Hessian. For posteriors that are well-approximated by a Gaussian near the mode this is dramatically cheaper than HMC: each iteration is one log-posterior evaluation plus one accept/reject step. Embarrassingly parallel across chains.

Use cases:

- Quick draws when Laplace is "almost right" but you want asymptotically-correct samples.
- Cross-validation, simulation studies, or other repeated-fit workflows where HMC startup cost dominates.
- Sanity check on the Laplace approximation: low IMH acceptance means the posterior is far from Gaussian and Laplace is biased.

Usage

```
imh_laplace(
  log_posterior,
  mode,
  hessian,
  n_iter = 2000L,
  warmup = n_iter%/%2L,
  scale = 1,
  init = NULL,
  thin = 1L,
  seed = NULL,
  verbose = FALSE
)
```

Arguments

`log_posterior`	Function `function(theta) -> numeric` returning the *unnormalized* log posterior at a single parameter vector.
`mode`	Numeric vector of length `d`: the Laplace mode (i.e., `tulpa_laplace(...)$mode[seq_len(d)]` for the fixed-effects block, or any other mode you trust).
`hessian`	Symmetric positive-definite `d x d` matrix: the negative Hessian of `log_posterior` at `mode` (or `H_beta` from `tulpa_laplace`).
`n_iter`	Total iterations including warmup (default 2000).
`warmup`	Warmup iterations to discard (default `n_iter / 2`).
`scale`	Optional inflation factor on the proposal covariance (default 1.0). Values slightly > 1 (e.g., 1.5) give heavier-tailed proposals that improve mixing when Laplace underestimates posterior spread.

init	Optional starting parameter vector (default = mode).
thin	Keep every thin-th post-warmup sample (default 1).
seed	Optional integer RNG seed. Scoped to this call: the caller's .Random.seed is restored on exit, so two calls with the same seed give identical draws regardless of the surrounding RNG stream.
verbose	Print acceptance rate at end (default FALSE).

Value

A list with class tulpa_fit carrying:

- draws: (n_iter - warmup) / thin x d matrix of draws.
- means: posterior means.
- accept_prob: per-iteration acceptance indicators.
- mean_accept: overall post-warmup acceptance rate.
- log_prob: per-draw log posterior values.
- inference_mode: "exact".
- inference_tier: 1L.
- backend: "imh_laplace".

Tier

Tier 1 (Exact). The MH accept/reject step makes the chain asymptotically correct under the standard MH conditions. Tier status does not depend on Laplace's quality – only its quality affects efficiency.

See Also

tulpa_laplace() for the mode + Hessian, bridge_sampling() for marginal-likelihood estimation on the resulting draws.

Examples

```
# Toy: Bernoulli logistic with one covariate.
set.seed(1)
n <- 100
x <- rnorm(n)
eta <- 0.3 + 1.2 * x
y <- rbinom(n, 1, plogis(eta))
X <- cbind(1, x)

lap <- tulpa_laplace(y, n_trials = rep(1L, n), X = X,
                     family = "binomial")

log_post <- function(beta) {
  eta <- as.numeric(X %*% beta)
  sum(y * eta - log1p(exp(eta))) +
    sum(dnorm(beta, 0, 10, log = TRUE))
}
```

```
fit <- imh_laplace(log_post, mode = lap$mode[1:2],
                   hessian = lap$H_beta, n_iter = 1000)
fit$mean_accept
colMeans(fit$draws)
```

---

inference_mode_info *Print inference mode information*

Description

Displays information about the inference modes available in tulpa, including their tiers, guarantees, and appropriate use cases.

Usage

```
inference_mode_info()
```

Value

NULL, invisibly. Called for the side effect of printing the available inference modes, their tiers, guarantees, and use cases to the console.

Examples

```
inference_mode_info()
```

---

is_auto_grid *Is an outer-grid setting marked as a default?*

Description

Is an outer-grid setting marked as a default?

Usage

```
is_auto_grid(x)
```

Arguments

x	Any object.

Value

TRUE when x carries the auto_grid() marker. This is the PROVENANCE question – whose choice the nodes are – and is TRUE whether or not the mark also asked for them to be integrated as written; that is auto_grid_place().

See Also

auto_grid(), auto_grid_place()

Examples

```
is_auto_grid(auto_grid(c(0.5, 1, 2)))
is_auto_grid(c(0.5, 1, 2))
```

---

latent *Mark an expression as a latent block in a tulpa formula*

Description

Wraps a user-defined latent block (currently a tgmrf() object) so the formula parser can recognise and route it to the inference layer. The call is structural – it is never executed at fit time. The parser evaluates the inner expression in the formula's environment, removes the latent(...) term from the fixed-effects formula, and attaches the resulting object to parsed$latent_blocks.

Usage

```
latent(block)
```

Arguments

block	A tgmrf (or, in the future, tgeneric) object.

Value

Returns block invisibly. Outside a formula context the call is a no-op pass-through.

Examples

```
n <- 8
blk <- tgmrf(
  Q = function(theta) Matrix::Diagonal(n, exp(-2 * theta[1])),
  prior = function(theta) stats::dnorm(theta[1], 0, 1, log = TRUE),
  init = c(log_sigma = 0))
parsed <- tulpa_parse_formula(y ~ x + latent(blk))
parsed$n_latent_blocks
```

latent_factor *Create a latent factor specification*

Description

Define latent factors for capturing unmeasured shared structure between all processes. Factors are observation-level random effects that enter both linear predictors when `shared = TRUE` (default).

Usage

```
latent_factor(
  n_factors = 1L,
  prior = NULL,
  shared = TRUE,
  constraint = c("sum_to_zero", "first_zero"),
  scale = TRUE
)

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

Arguments

`n_factors`	Integer; number of latent factors. Default is 1. More factors capture more complex unmeasured structure but increase computational cost and risk overfitting.
`prior`	Prior for factor standard deviations. Default is a PC prior with P(sigma > 1) = 0.01, which shrinks toward simpler models.
`shared`	Logical; if TRUE (default), latent factors enter both all process linear predictors identically. If FALSE, factors only affect the first process.
`constraint`	Identifiability constraint for factors: • `"sum_to_zero"` (default): Factors sum to zero across observations • `"first_zero"`: First observation's factor is fixed to zero
`scale`	Logical; if TRUE (default), factor loadings are standardized to have unit variance before applying sigma.
`x`	A `tulpa_latent` specification.
`...`	Ignored.

Details

Why Use Latent Factors?:

When modeling multiple processes, they often share unmeasured confounders. For example, in relative abundance data, both the focal species count and total count might be affected by:

- Observer skill (unmeasured)
- Local microhabitat conditions (unmeasured)

- Weather on sampling day (unmeasured)

Without accounting for these shared drivers, estimates can be biased. Latent factors capture this shared structure without requiring the confounders to be measured.

Mathematical Model:

For observation i with K latent factors, on each of two model arms (processes 1 and 2):

$$\eta_i^{(1)} = X_i^{(1)}\beta^{(1)} + \sum_{k=1}^{K} f_{ik}\sigma_k + \dots$$

$$\eta_i^{(2)} = X_i^{(2)}\beta^{(2)} + \sum_{k=1}^{K} f_{ik}\sigma_k + \dots$$

where:

- $f_{ik} \sim N(0,1)$ are standardized factor scores
- σ_k are factor standard deviations with PC prior
- Identifiability: $\sum_i f_{ik} = 0$ for each k

Because factors enter both linear predictors identically (when shared), they cancel in derived quantities (e.g., ratios, differences):

$$\eta_i^{(1)} - \eta_i^{(2)}$$

This means factors capture shared effects that would otherwise bias derived quantities.

Choosing n_factors:

- Start with `n_factors = 1` for simple unmeasured confounding
- Use `n_factors = 2-3` if you suspect multiple independent confounders
- More than 3 factors is rarely needed and risks overfitting
- The PC prior provides regularization, shrinking unneeded factors toward zero

Relationship to Random Effects:

Latent factors differ from random effects in several ways:

- Random effects are grouped (e.g., site-level), factors are observation-level
- Random effects require grouping structure, factors don't
- Factors capture residual correlation not explained by observed predictors

You can use both together: random effects for known grouping, factors for residual unmeasured confounding.

Value

A `tulpa_latent` object consumed by ratio / multi-arm model packages built on tulpa (e.g. tulpaRatio); not read by the single-response `tulpa()` front door (use `latent(tgmrf(...))` for a single-response latent Gaussian block).

See Also

`prior_pc()` for prior specification; `latent()` / `tgmrf()` for a single-response in-formula latent Gaussian block

Examples

```
# Basic latent factor (single shared factor)
latent_factor()

# Two latent factors
latent_factor(n_factors = 2)

# Custom prior (more regularization)
latent_factor(n_factors = 1, prior = prior_pc(U = 0.5, alpha = 0.01))

# Numerator-only factor (not shared)
latent_factor(n_factors = 1, shared = FALSE)

# The spec is consumed by a ratio / multi-arm model package (tulpaRatio owns
# the two-arm `species | total ~ ...` formula and the negbin/negbin ratio
# family); it is that package's fitter, not the single-response tulpa() front
# door, that reads the `latent_factor()` object.
```

---

latent_factors *Extract latent factor posteriors from fit*

Description

Extract latent factor posteriors from fit

Usage

```
latent_factors(fit, ...)

## Default S3 method:
latent_factors(fit, summary = TRUE, probs = c(0.025, 0.5, 0.975), ...)
```

Arguments

`fit`	A fit sampled with latent factors: its draw matrix (`$draws` or `$samples`) holds columns named `factor[observation,factor]` and `$.internal$latent_info` records the layout. The default method reads any fit, of any class, that stores them this way.
`...`	Passed to methods.
`summary`	Logical; if TRUE, return summary statistics. If FALSE, return full posterior draws.
`probs`	Quantiles for summary. Default is c(0.025, 0.5, 0.975).

Value

If `summary = TRUE`, a data frame with columns for observation, factor, and summary statistics. If `summary = FALSE`, a matrix of posterior draws.

Examples

```
# A fit sampled with latent factors comes from a model package (tulpaRatio
# reads the latent_factor() spec). The accessor reads two things off it:
# draw columns named factor[observation,factor] and the latent layout. Here
# both are simulated for three observations on two factors.
set.seed(1)
cols <- sprintf("factor[%d,%d]", rep(1:3, 2), rep(1:2, each = 3))
draws <- matrix(rnorm(200 * 6, mean = rep(c(-1, 0, 1), 2), sd = 0.3),
                200, 6, byrow = TRUE, dimnames = list(NULL, cols))
fit <- structure(
  list(draws = draws,
       .internal = list(latent_info = list(type = "latent", n_factors = 2L,
                                           n_obs = 3L))),
  class = "tulpa_fit")

latent_factors(fit)
dim(latent_factors(fit, summary = FALSE))
```

`logLik.tulpa_fit` *The fit's log-scale goodness quantity*

Description

What this returns depends on what the fit computed, and the returned object names it in a `quantity` attribute:

Usage

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

Arguments

`object`	A `tulpa_fit` object.
`...`	Ignored.

Details

`"log_posterior_mean"` a sampler fit: the mean over the draws of the log joint posterior density, prior included, in that backend's own parameterization: the unconstrained coordinates with their Jacobians for the ModelData samplers (`hmc`, `ess`, `sghmc`, `sgld`, `mclmc`, `smc`, `vi`), and the natural scale each quantity is sampled on for the RE-covariance and Polya-Gamma Gibbs samplers. Values are therefore comparable across fits of the same backend only.

`"log_evidence"` a deterministic fit that estimated no hyperparameter from the data: the log marginal probability of the data under the model as specified. A hyperparameter the fit integrated counts, and so does one the caller supplied (a `phi`, a `sigma_re`, an outer grid laid at one value), which is part of the model rather than an estimate. A Laplace fit of a model with nothing to integrate reports its log marginal likelihood here, which already is that quantity.

"log_marginal_likelihood" a deterministic fit that estimated some hyperparameters by maximising over the same data (empirical Bayes, or estimate_phi = TRUE): the log marginal likelihood at those estimates. The conditioned_on attribute names them.

"log_likelihood" a fit that maximised over every parameter it reports, with the random effects integrated out (agq_fit()): the maximised log-likelihood, with df the number of maximised parameters.

On a nested-Laplace fit the value is the log evidence of its outer grid, log sum_k exp(log_marginal_k + log_cell_k), where log_marginal already carries each axis's hyperprior density on its integration coordinate and log_cell_k is the cell's absolute volume there. Every default outer axis carries a proper prior (a PC prior on each scale and range, a uniform on each bounded axis), so the value is the evidence under that prior and does not move with where the nodes were laid or how many there are, provided the grid covers the posterior. A fit carrying an axis with no proper prior (a flat random-effect covariance prior, a dispersion whose family has no PC prior yet, an axis on a block that does not carry its coordinates) reports NA with declined = "improper_hyperprior" and names the axes in a declined_axes attribute. A central-composite (CCD) design reproduces moments and carries no cell volume, so a fit integrated on one reports NA with a declined attribute.

A sampler fit whose producer kept no per-draw log posterior reports NA with quantity = "log_posterior_mean" and declined = "no_log_posterior_recorded". The Polya-Gamma Gibbs routes that recentre a proper field every sweep (the NNGP and multiscale-GP fields, and the negative-binomial kernel's iid random-effect block) leave no written density invariant and record none; a fit carrying neither draws nor a log marginal declines with "no_goodness_quantity_recorded". A value of NA always carries a declined attribute.

Values with a different quantity or conditioned_on are not on one scale, and compare_models(criterion = "loglik") refuses such a set. Only a "log_likelihood" is what AIC and BIC penalise, so AIC.tulpa_fit() and BIC.tulpa_fit() refuse every other quantity; compare those fits by their evidence, or by compare_models(criterion = "waic") / "loo", which score the pointwise predictive density.

Value

A logLik object with quantity and conditioned_on attributes, and a declined attribute naming the reason when no value could be read.

---

mala *Metropolis-adjusted Langevin algorithm (MALA)*

Description

Sample from a posterior using MALA: a Langevin proposal driven by the gradient of the log posterior, corrected by a Metropolis- Hastings accept/reject step. Each iteration is one log-posterior + one gradient evaluation. The natural stepping stone between random-walk MH and HMC: cheaper per iteration than HMC (no leapfrog integration), better-mixing than RWMH because the drift term moves toward higher density.

Step size epsilon is adapted via dual averaging during warmup to target an acceptance rate of 0.574 (Roberts & Rosenthal 1998 optimal for high-dimensional Gaussians).

Usage

```
mala(
  log_posterior,
  grad_log_posterior,
  init,
  n_iter = 2000L,
  warmup = n_iter%/%2L,
  epsilon = 0.1,
  target_accept = 0.574,
  mass_diag = NULL,
  mass_matrix = NULL,
  thin = 1L,
  seed = NULL,
  verbose = FALSE
)
```

Arguments

`log_posterior` Function `function(theta) -> numeric` returning the *unnormalized* log posterior.

`grad_log_posterior` Function `function(theta) -> numeric vector` returning the gradient at `theta`.

`init` Numeric vector: initial state (must give finite log_posterior).

`n_iter` Total iterations including warmup (default 2000).

`warmup` Warmup iterations (step-size adaptation + discarded; default `n_iter / 2`).

`epsilon` Initial step size (default 0.1). Adapted during warmup.

`target_accept` Target acceptance during warmup adaptation (default 0.574, the Roberts & Rosenthal 1998 optimum).

`mass_diag` Optional preconditioner: a length-d vector of per-dimension **variances**, used as the diagonal inverse-mass `M^-1`. The proposal is `N(theta + (eps^2/2) * M^-1 * grad, eps^2 * M^-1)`, so entry `j` scales the proposal variance along dimension `j` and the noise standard deviation is `eps * sqrt(mass_diag[j])`. Default `rep(1, d)`. For posteriors with very different scales across dimensions, set this to (an estimate of) the posterior variances, i.e. the squared posterior SDs.

`mass_matrix` Optional dense preconditioner: a `d x d` symmetric positive-definite inverse mass `M^-1`, typically (an estimate of) the posterior covariance. The proposal is then `N(theta + (eps^2/2) * M^-1 * grad, eps^2 * M^-1)`, which also reaches the correlations a diagonal cannot: in a random-effect model the fixed intercept and the group effects are strongly correlated a posteriori, and an identity or diagonal metric crosses that ridge at the rate of its narrowest direction. Supply at most one of `mass_diag` / `mass_matrix`.

`thin` Keep every `thin`-th post-warmup sample (default 1).

`seed` Optional integer RNG seed. Scoped to this call: the caller's `.Random.seed` is restored on exit, so two calls with the same `seed` give identical draws regardless of the surrounding RNG stream.

`verbose` Print acceptance + step-size summary at end (default FALSE).

Value

A list with class `tulpa_fit` carrying:

- `draws`: post-warmup draws.
- `means`, `n_params`, `n_samples`, `log_prob`.
- `accept_prob`: per-iteration accepted indicators (post-warmup).
- `mean_accept`: post-warmup acceptance rate.
- `epsilon`: final adapted step size.
- `inference_mode`: `"exact"`.
- `inference_tier`: `1L`.
- `backend`: `"mala"`.

Tier

Tier 1 (Exact). The MH step makes the chain asymptotically correct. Mixing depends on whether the gradient gives useful local geometry – poor for posteriors with very different scales across dimensions (use a preconditioner or HMC instead).

References

Roberts, G. O., & Tweedie, R. L. (1996). Exponential convergence of Langevin distributions and their discrete approximations. *Bernoulli*, 2(4), 341-363.

Roberts, G. O., & Rosenthal, J. S. (1998). Optimal scaling of discrete approximations to Langevin diffusions. *JRSS B*, 60(1), 255-268.

Examples

```
log_post <- function(t) -0.5 * sum((t - c(1, 2))^2)
grad <- function(t) -(t - c(1, 2))
fit <- mala(log_post, grad, init = c(0, 0), n_iter = 1000)
colMeans(fit$draws)  # near c(1, 2)
fit$mean_accept      # should adapt toward 0.574
```

`mcmc_draws` *MCMC chain draws from a fit*

Description

Returns a fit's posterior draws only when they form a genuine MCMC chain (`$draws_kind == "chain"`, or an untagged legacy fit); for an i.i.d. / approximation fit (nested Laplace, VI, SMC, ...) it returns `NULL`, because chain diagnostics do not apply. This is the accessor `diagnostics()` gates on. For the provenance-agnostic posterior sample used by summaries, see `posterior_sample()`.

Usage

```
mcmc_draws(fit)
```

Arguments

fit	A tulpa_fit (or subclass) carrying posterior $draws.

Value

The chain draws matrix/array, or NULL for a non-chain fit.

See Also

posterior_sample(), diagnostics()

Examples

```
set.seed(1)
d <- data.frame(x = rnorm(60))
d$y <- rpois(60, exp(0.5 + 0.3 * d$x))
fit <- tulpa(y ~ x, data = d, family = "poisson", mode = "hmc",
             control = list(n_iter = 400L, warmup = 200L, n_chains = 2L,
                            seed = 1L))
dim(mcmc_draws(fit))
```

---

model_average *Model-averaged predictions*

Description

Combine fitted values from several models using native model weights computed from the pointwise PSIS-LOO (or WAIC) elpd via tulpa_psis() – no loo package dependency. "loo" / "waic" give stacking weights (the simplex-optimal predictive combination); "pbma" / "pbma+" give pseudo-BMA(+) weights. Every model must carry an [n_draws x n_obs] pointwise log-likelihood (fit$draws$log_lik) over the same observations.

Usage

```
model_average(
  ...,
  weights = c("loo", "waic", "pbma", "pbma+"),
  fitted_fn = fitted
)
```

Arguments

...	Named tulpa_fit objects fitted to the same observations.
weights	"loo" (stacking, default), "waic", "pbma", or "pbma+".
fitted_fn	Function extracting a length-n_obs fitted vector from a fit (default fitted()).

Value

A list with averaged (the weighted fitted vector), weights (the named model weights), and comparison (the compare_models() table).

References

Yao, Vehtari, Simpson & Gelman (2018). Using stacking to average Bayesian predictive distributions. *Bayesian Analysis* 13(3):917-1007.

See Also

compare_models().

Examples

```
set.seed(1)
df <- data.frame(x = rnorm(120))
df$y <- rpois(120, exp(0.4 + 0.5 * df$x))
f1 <- tulpa(y ~ x, data = df, family = "poisson", mode = "hmc",
            control = list(n_iter = 500L, warmup = 250L, seed = 1L))
f2 <- tulpa(y ~ 1, data = df, family = "poisson", mode = "hmc",
            control = list(n_iter = 500L, warmup = 250L, seed = 1L))
ma <- model_average(full = f1, null = f2, weights = "waic")
ma$weights
```

---

moran_i — *Moran's I test for spatial autocorrelation in residuals*

Description

Tests whether residuals exhibit spatial structure after model fitting. Supports inverse-distance and k-nearest-neighbour weight matrices. No external dependencies (uses normal approximation for inference).

Usage

```
moran_i(
  object,
  coords,
  weights = c("inverse", "knn"),
  k = 10L,
  resid_type = "pearson",
  alternative = c("two.sided", "greater", "less")
)
```

Arguments

`object`	A fitted model, or a numeric vector of residuals
`coords`	N x 2 coordinate matrix (required)
`weights`	Weight scheme: `"inverse"` or `"knn"`
`k`	Number of neighbours for knn (default 10)
`resid_type`	Residual type if extracting from model (default `"pearson"`)
`alternative`	`"two.sided"`, `"greater"`, or `"less"`

Value

An `htest` object with Moran's I, expected I, and p-value

Examples

```
set.seed(1)
coords <- cbind(runif(50), runif(50))
resid  <- rnorm(50)
moran_i(resid, coords)
```

`nobs.tulpa_fit` *Number of observations in a tulpa fit*

Description

Number of observations in a tulpa fit

Usage

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

Arguments

`object`	A `tulpa_fit` object.
`...`	Ignored.

Value

Integer observation count.

`node_index` *Map cell identifiers to graph node indices*

Description

Translate a vector of original cell identifiers into the 1-based node indices of a `tulpa_adjacency` graph, by key (not by row order). Use it to add the node-index column the model's spatial grouping bar needs to the observation data, which typically has many rows per cell and a different row order than the graph.

Usage

```
node_index(graph, ids)
```

Arguments

`graph`	A `tulpa_adjacency` object from `adjacency()`.
`ids`	A vector of cell identifiers to look up (matched against `graph$ids`).

Value

An integer vector the same length as `ids`, giving each one's node index in `graph` (`NA` for identifiers absent from the graph).

See Also

`adjacency()`.

Examples

```
grid <- expand.grid(x = 1:3, y = 1:3)
grid$cell <- paste0("c", seq_len(nrow(grid)))
g <- adjacency(grid, id = "cell")

obs <- data.frame(cell = c("c5", "c1", "c9"))
obs$cell_idx <- node_index(g, obs$cell)
obs
```

n_divergent — *Number of divergent transitions*

Description

Counts divergent transitions recorded by an HMC/NUTS fit, reading whichever field the back-end populated (the top-level `$divergent` flag vector every sampler in this package writes, or `$diagnostics$divergent`, `$diagnostics$divergent_idx`, `$diagnostics$n_divergent`, `$n_divergent`). It reads the same record `plot_divergences()` locates the divergent rows from, so the two always agree on one fit.

Usage

```
n_divergent(fit)
```

Arguments

`fit`	A `tulpa_fit` object.

Value

Integer count of divergent transitions (0 if none are recorded).

See Also

`plot_divergences()`, `check_diagnostics()`

Examples

```
set.seed(1)
d <- data.frame(x = rnorm(60))
d$y <- rpois(60, exp(0.5 + 0.3 * d$x))
fit <- tulpa(y ~ x, data = d, family = "poisson", mode = "hmc",
             control = list(n_iter = 400L, warmup = 200L, n_chains = 2L,
                            seed = 1L))
n_divergent(fit)
```

pathfinder *Pathfinder: variational warm-start via L-BFGS + ELBO scoring*

Description

Single-path Pathfinder (Zhang, Carpenter, Gelman, Vehtari 2022): run L-BFGS toward the posterior mode, fit a Gaussian at the optimum using the inverse-Hessian estimate, and report draws plus the ELBO. Cheap, derivative-only, embarrassingly parallel – meant as an HMC warm-start, an initialiser for `imh_laplace()`, or a quick sanity check on the Laplace approximation.

This implementation is **single-path only**. The full multi-path Pathfinder (K parallel L-BFGS runs + mixture proposal + Pareto- smoothed importance reweighting) is a follow-on. The single-path version is what most users want as a Laplace-equivalent diagnostic.

Usage

```
pathfinder(
  log_posterior,
  init,
  grad_log_posterior = NULL,
  n_draws = 1000L,
  max_iter = 100L,
  tol = 1e-06,
  seed = NULL,
  verbose = FALSE
)
```

Arguments

`log_posterior`	Function `function(theta) -> numeric` returning the *unnormalized* log posterior at `theta`.
`init`	Numeric vector: initial point for L-BFGS. Should be in the support of the posterior (finite log_posterior).
`grad_log_posterior`	Optional function returning the gradient of `log_posterior` at `theta`. If `NULL`, gradients are computed numerically via stats::optim's built-in finite differences.
`n_draws`	Number of draws from the fitted Gaussian (default 1000).
`max_iter`	L-BFGS iteration cap (default 100).
`tol`	Gradient-norm tolerance for L-BFGS convergence (default 1e-6).
`seed`	Optional integer RNG seed. Scoped to this call: the caller's `.Random.seed` is restored on exit, so two calls with the same `seed` give identical draws regardless of the surrounding RNG stream.
`verbose`	Print L-BFGS / ELBO summary at end (default FALSE).

Value

A list with class `tulpa_fit` carrying:

- `draws`: `n_draws x d` matrix of Gaussian draws at the mode.
- `means`: posterior means (= mode for a Gaussian fit).
- `mode`: the L-BFGS mode.
- `cov`: the proposal covariance (`solve(-hessian)`).
- `elbo`: Monte-Carlo estimate of `E_q[log p - log q]`.
- `n_iter`: L-BFGS iterations.
- `converged`: logical.
- `inference_mode`: `"structured"`.
- `inference_tier`: `2L`.
- `backend`: `"pathfinder"`.

Tier

Tier 2 (Structured). The output is a Gaussian approximation, not samples from the exact posterior – same epistemic class as `tulpa_laplace()`. Pair with `imh_laplace()` for an exact-tier upgrade.

References

Zhang, L., Carpenter, B., Gelman, A., & Vehtari, A. (2022). Pathfinder: parallel quasi-Newton variational inference. *Journal of Machine Learning Research*, 23(306), 1-49.

See Also

`imh_laplace()` for an exact-tier MH using the Pathfinder Gaussian as proposal; `bridge_sampling()` for marginal-likelihood estimation on the resulting draws.

Examples

```
# Toy: 2-D conjugate normal.
y <- c(0.5, -0.7)
log_post <- function(t) {
  sum(dnorm(y, t, 1, log = TRUE)) +
    sum(dnorm(t, 0, sqrt(10), log = TRUE))
}
pf <- pathfinder(log_post, init = c(0, 0), n_draws = 2000)
pf$mode      # near c(0.45, -0.64)
pf$elbo
```

`pit_residuals` *PIT (probability integral transform) residuals*

Description

For each observation, computes the quantile of the observed value within the posterior predictive distribution. If the model is correct, PIT residuals follow Uniform(0, 1).

Usage

```
pit_residuals(object, ...)

## Default S3 method:
pit_residuals(object, observed = NULL, nsim = 250L, seed = 123L, ...)
```

Arguments

`object`	A fitted model with a `simulate()` method, or a matrix of simulated values (n_obs x nsim)
`...`	Passed to methods.
`observed`	Observed response vector (required if `object` is a matrix)
`nsim`	Number of simulations (default 250)
`seed`	Random seed (default 123)

Details

For integer-valued responses, a randomisation step avoids discrete artefacts: the residual is drawn uniformly between P(sim < obs) and P(sim <= obs).

Value

Numeric vector of length n_obs with values in `[0, 1]`

Examples

```
# From a matrix of replicates (n_obs x nsim) and the observed response:
set.seed(1)
y <- rpois(30, 3)
sims <- matrix(rpois(30 * 100, 3), nrow = 30)
r <- pit_residuals(sims, observed = y)
summary(r)

# From a fitted model with a simulate() method:
set.seed(1)
d <- data.frame(x = rnorm(80))
d$y <- rpois(80, exp(0.3 + 0.5 * d$x))
fit <- tulpa(y ~ x, data = d, family = "poisson", mode = "laplace")
head(pit_residuals(fit, nsim = 50))
```

plot.tulpa_fit *Plot fixed-effect posteriors*

Description

Plot fixed-effect posteriors

Usage

```
## S3 method for class 'tulpa_fit'
plot(x, type = c("density", "trace", "pairs", "smooth"), term = NULL, ...)
```

Arguments

`x`	A `tulpa_fit` object.
`type`	One of `"density"`, `"trace"`, `"pairs"`, `"smooth"`. The Laplace tier has no draws, so it always shows the Gaussian densities of the fixed effects. `"smooth"` draws the fitted curve of each `s(...)` term and requires a fit carrying one.
`term`	For `type = "smooth"`, which smoother to draw: index or covariate name. `NULL` (default) draws every one. Ignored by the other types.
`...`	Passed to plotting functions.

Value

The input `x`, returned invisibly. Called for the side effect of producing base-graphics plots of the fixed-effect posteriors.

plot.tulpa_prior_predict *Plot method for tulpa_prior_predict*

Description

Density overlay of prior predictive draws. Requires bayesplot; falls back to base graphics matplot of a subset of draws otherwise.

Usage

```
## S3 method for class 'tulpa_prior_predict'
plot(x, process = 1L, max_draws = 50L, ...)
```

Arguments

`x`	A tulpa_prior_predict object
`process`	Process index or name (multi-process families)
`max_draws`	Maximum draws to overlay. Default 50.
`...`	Passed through.

Value

A `ggplot` object (via bayesplot) when bayesplot is installed; otherwise `NULL` invisibly, after drawing a base-graphics overlay. Called for the density overlay of the prior predictive draws.

`plot.tulpa_st_summary` *Plot method for spatiotemporal effects*

Description

Plot method for spatiotemporal effects

Usage

```
## S3 method for class 'tulpa_st_summary'
plot(x, type = "heatmap", ...)
```

Arguments

`x`	Spatiotemporal effects object
`type`	Plot type: `"heatmap"` (default), `"time_series"`, or `"spatial_map"`
`...`	Additional arguments passed to plotting functions

Value

A `ggplot` object when ggplot2 is installed; otherwise `NULL` invisibly, after drawing a base-graphics plot. Called for the side effect of visualizing the spatiotemporal interaction effects.

plot.tulpa_svc_posterior

Plot method for tulpa_svc_posterior

Description

Plot method for tulpa_svc_posterior

Usage

```
## S3 method for class 'tulpa_svc_posterior'
plot(x, term = 1, type = "mean", ...)
```

Arguments

x	A tulpa_svc_posterior object
term	Which term to plot (name or index). Default: first term.
type	Plot type: "mean" (default), "sd", or quantile (e.g., "q50")
...	Additional arguments passed to plotting functions

Value

A `ggplot` object when ggplot2 is installed; otherwise `NULL` invisibly, after drawing a base-graphics map. Called for the side effect of mapping the selected spatially-varying coefficient.

plot.tulpa_temporal_posterior

Plot method for tulpa_temporal_posterior

Description

Plot method for tulpa_temporal_posterior

Usage

```
## S3 method for class 'tulpa_temporal_posterior'
plot(x, component = NULL, type = "ribbon", ...)
```

Arguments

x	A tulpa_temporal_posterior object
component	Which component to plot (for multi-scale). Default: first.
type	Plot type: "ribbon" (default) or "line"
...	Additional arguments passed to plotting functions

Value

A `ggplot` object when ggplot2 is installed; otherwise `NULL` invisibly, after drawing a base-graphics plot. Called for the side effect of plotting the temporal-effect posterior.

---

plot.tulpa_tvc_posterior *Plot method for tulpa_tvc_posterior*

Description

Plot method for tulpa_tvc_posterior

Usage

```
## S3 method for class 'tulpa_tvc_posterior'
plot(x, term = 1, type = "ribbon", ...)
```

Arguments

`x`	A tulpa_tvc_posterior object
`term`	Which term to plot (name or index). Default: first term.
`type`	Plot type: "ribbon" (default) or "line"
`...`	Additional arguments passed to plotting functions

Value

A `ggplot` object when ggplot2 is installed; otherwise `NULL` invisibly, after drawing a base-graphics plot. Called for the side effect of plotting the selected temporally-varying coefficient.

---

plot_acf *Plot autocorrelation functions*

Description

Creates autocorrelation function (ACF) plots for selected parameters. High autocorrelation indicates slow mixing and low effective sample size.

Usage

```
plot_acf(fit, pars = NULL, lags = 25, n_pars = 6)
```

Arguments

fit	A tulpa_fit object.
pars	Character vector of parameter names. If NULL, selects worst-mixing parameters based on ESS.
lags	Maximum number of lags to compute (default: 25).
n_pars	Maximum number of parameters to plot (default: 6).

Details

Ideal ACF plots show rapid decay to zero. Slow decay indicates high autocorrelation, which reduces effective sample size and may indicate poor mixing.

Value

A ggplot object (if ggplot2 available) or base R plot (invisible).

See Also

plot_ess(), diagnostics()

Examples

```
set.seed(123)
df <- data.frame(x = rnorm(60))
df$y <- rpois(60, exp(0.5 + 0.3 * df$x))
fit <- tulpa(y ~ x, data = df, family = "poisson", mode = "hmc",
             control = list(n_iter = 500L, warmup = 250L, n_chains = 2L,
                            seed = 1L))
plot_acf(fit)
plot_acf(fit, lags = 10)
```

---

plot_divergences *Plot divergent transitions*

Description

Creates visualizations to investigate divergent transitions. Parallel coordinates and scatter plots highlight where in parameter space divergences occur.

Usage

```
plot_divergences(fit, pars = NULL, type = c("parcoord", "scatter"))
```

Arguments

`fit`	A `tulpa_fit` object (HMC backend).
`pars`	Character vector of parameter names. If NULL, uses variance parameters.
`type`	Plot type: "parcoord" (parallel coordinates) or "scatter".

Details

Divergent transitions indicate regions of high posterior curvature that the sampler cannot efficiently explore. Common causes:

- Very narrow funnels (hierarchical models)
- Strong correlations
- Multi-modality

Value

A ggplot object or base R plot (invisible).

See Also

`plot_pairs()`, `n_divergent()`

Examples

```
set.seed(123)
df <- data.frame(x = rnorm(60))
df$y <- rpois(60, exp(0.5 + 0.3 * df$x))
fit <- tulpa(y ~ x, data = df, family = "poisson", mode = "hmc",
             control = list(n_iter = 500L, warmup = 250L, n_chains = 2L,
                            seed = 1L))
plot_divergences(fit)
plot_divergences(fit, type = "scatter")
```

`plot_energy` *Plot energy diagnostic (E-BFMI)*

Description

Creates overlaid histograms of marginal energy and energy transition, along with the E-BFMI statistic. Low E-BFMI indicates poor exploration of the posterior.

Usage

```
plot_energy(fit)
```

Arguments

`fit`	A `tulpa_fit` object (HMC backend).

Details

E-BFMI (Energy Bayesian Fraction of Missing Information) compares the distribution of energy levels to energy transitions. Values below 0.3 indicate the sampler may not be exploring the full posterior.

Value

A ggplot object (if ggplot2 available) or base R plot.

See Also

`diagnostic_summary()`, `check_diagnostics()`

Examples

```
set.seed(123)
df <- data.frame(x = rnorm(60))
df$y <- rpois(60, exp(0.5 + 0.3 * df$x))
fit <- tulpa(y ~ x, data = df, family = "poisson", mode = "hmc",
             control = list(n_iter = 500L, warmup = 250L, n_chains = 2L,
                            seed = 1L))
plot_energy(fit)
```

`plot_ess` *Plot effective sample size diagnostic*

Description

Creates a visual display of effective sample size (ESS) for all parameters, expressed as a ratio of ESS to total samples. Low ESS indicates high autocorrelation.

Usage

```
plot_ess(fit, type = c("bulk", "tail"), threshold = 400, pars = NULL)
```

Arguments

`fit`	A `tulpa_fit` object.
`type`	Type of ESS: "bulk" (default) or "tail".
`threshold`	Minimum acceptable ESS (default: 400).
`pars`	Character vector of parameter names to include.

Details

ESS/iter ratio interpretation:

- Green: ESS >= threshold
- Yellow: threshold/2 <= ESS < threshold
- Red: ESS < threshold/2

Value

A ggplot object (if ggplot2 available) or base R plot (invisible).

See Also

plot_rhat(), diagnostic_summary(), diagnostics()

Examples

```
set.seed(123)
df <- data.frame(x = rnorm(60))
df$y <- rpois(60, exp(0.5 + 0.3 * df$x))
fit <- tulpa(y ~ x, data = df, family = "poisson", mode = "hmc",
             control = list(n_iter = 500L, warmup = 250L, n_chains = 2L,
                            seed = 1L))
plot_ess(fit)
plot_ess(fit, type = "tail")
```

---

plot_map *Plot spatial predictions as a map*

Description

Create publication-ready maps from a tulpa spatial fit: the fitted response surface or the prediction uncertainty (credible-interval width), on the response scale.

Usage

```
plot_map(
  x,
  coords = NULL,
  what = c("fitted", "uncertainty"),
  summary = c("median", "mean", "q2.5", "q97.5", "sd"),
  newdata = NULL,
  title = NULL,
  palette = "viridis",
  points = FALSE,
  point_color = "grey30",
```

```
  point_size = 0.5,
  na_color = "transparent",
  crs = NULL,
  legend_title = NULL,
  ...
)
```

Arguments

x	A tulpa_fit object with spatial structure, or a data frame containing predictions with coordinate columns.
coords	A data frame or matrix with spatial coordinates (columns named 'x'/'y', 'X'/'Y', 'lon'/'lat', 'longitude'/'latitude', or 'Easting'/'Northing'). Required if x is a data frame.
what	What to plot: "fitted" (default, the response-scale point prediction) or "uncertainty" (the 95% credible-interval width).
summary	Which summary statistic for what = "fitted": "median" (default) / "mean" (both the point prediction), "q2.5" / "q97.5" (the credible bounds), or "sd" (the link-scale standard error).
newdata	Optional data frame with prediction locations and covariates. If NULL and x is a tulpa_fit, uses fitted values at observed locations.
title	Plot title. If NULL, auto-generated based on what.
palette	Color palette: "viridis" (default), "magma", "plasma", "inferno", "cividis", "mako", "rocket", or a custom vector of colors.
points	Logical; if TRUE, overlay observation points. Default FALSE.
point_color	Color for observation points. Default "grey30".
point_size	Size for observation points. Default 0.5.
na_color	Color for NA values. Default "transparent".
crs	Coordinate reference system (proj4 string or EPSG code). If NULL, uses planar coordinates.
legend_title	Title for the color legend. If NULL, auto-generated.
...	Additional arguments passed to ggplot2 theme functions.

Details

This function provides a streamlined workflow for visualizing spatial predictions from tulpa models. It handles:

- Extracting predictions from tulpa_fit objects
- Converting to appropriate spatial format (stars/sf)
- Creating publication-quality maps with sensible defaults
- Uncertainty visualization via credible interval width

For custom maps or more control, extract predictions using predict() and use ggplot2 directly with geom_stars() or geom_sf().

Value

A ggplot2 object that can be further customized.

Required packages

This function requires `ggplot2`. For raster-style maps, `stars` and `sf` are also needed. Install with:

```
install.packages(c("ggplot2", "stars", "sf"))
```

See Also

`predict.tulpa_fit()` for predictions at new locations

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  set.seed(123)
  n_sites <- 20
  df <- data.frame(
    y = rbinom(n_sites, 20, 0.4),
    elevation = rnorm(n_sites),
    site = factor(seq_len(n_sites)),
    lon = runif(n_sites),
    lat = runif(n_sites)
  )
  adj <- matrix(0, n_sites, n_sites)
  for (i in 1:(n_sites - 1)) adj[i, i + 1] <- adj[i + 1, i] <- 1
  fit <- tulpa(
    y ~ elevation + spatial(site),
    data = df,
    family = "binomial",
    n_trials = rep(20L, n_sites),
    spatial = spatial_car(adj, group_var = "site"),
    mode = "laplace"
  )
  # Areal (CAR/ICAR) fits carry no point coordinates, so pass `coords`:
  cc <- df[, c("lon", "lat")]
  plot_map(fit, coords = cc)                      # fitted response surface
  plot_map(fit, what = "uncertainty", coords = cc)
  plot_map_panel(fit, coords = cc)                # both side by side
}
```

---

plot_map_panel *Plot multiple maps in a grid*

Description

Create a multi-panel figure with the fitted-value map and the prediction-uncertainty map side by side.

Usage

```
plot_map_panel(x, newdata = NULL, ncol = 2, ...)
```

Arguments

x	A tulpa_fit object with spatial structure.
newdata	Optional data frame with prediction locations.
ncol	Number of columns in the grid. Default 2.
...	Additional arguments passed to plot_map().

Value

A patchwork object (if patchwork is installed) or a list of ggplots.

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  set.seed(123)
  n_sites <- 20
  df <- data.frame(
    y = rbinom(n_sites, 20, 0.4),
    elevation = rnorm(n_sites),
    site = factor(seq_len(n_sites)),
    lon = runif(n_sites),
    lat = runif(n_sites)
  )
  adj <- matrix(0, n_sites, n_sites)
  for (i in 1:(n_sites - 1)) adj[i, i + 1] <- adj[i + 1, i] <- 1
  fit <- tulpa(
    y ~ elevation + spatial(site),
    data = df,
    family = "binomial",
    n_trials = rep(20L, n_sites),
    spatial = spatial_car(adj, group_var = "site"),
    mode = "laplace"
  )
  cc <- df[, c("lon", "lat")]
  plot_map_panel(fit, coords = cc)
  plot_map_panel(fit, coords = cc, ncol = 1)
}
```

`plot_pairs` *Plot bivariate parameter posteriors (pairs plot)*

Description

Creates a pairs plot showing bivariate relationships between parameters. Divergent transitions (if present) are highlighted to help identify problematic posterior regions.

Usage

```
plot_pairs(
  fit,
  pars = NULL,
  highlight_divergent = TRUE,
  n_pars = 5,
  alpha = 0.3
)
```

Arguments

`fit`	A `tulpa_fit` object.
`pars`	Character vector of parameter names. If NULL, selects main variance parameters.
`highlight_divergent`	Logical; highlight divergent transitions in red (default: TRUE).
`n_pars`	Maximum number of parameters (default: 5).
`alpha`	Point transparency (default: 0.3).

Details

Pairs plots help identify:

- Strong correlations between parameters (potential non-identifiability)
- Multimodality
- Regions where divergences cluster (indicating problematic geometry)

Value

A ggplot object (if ggplot2/GGally available) or base R plot.

See Also

`plot_divergences()`, `diagnostics()`

Examples

```
set.seed(123)
df <- data.frame(x = rnorm(60))
df$y <- rpois(60, exp(0.5 + 0.3 * df$x))
fit <- tulpa(y ~ x, data = df, family = "poisson", mode = "hmc",
             control = list(n_iter = 500L, warmup = 250L, n_chains = 2L,
                            seed = 1L))
plot_pairs(fit)
plot_pairs(fit, n_pars = 2)
```

---

plot_rhat *Plot Rhat convergence diagnostic*

Description

Creates a visual display of Rhat values for all parameters, with color-coding to highlight convergence issues. Rhat values > 1.01 indicate potential convergence problems.

Usage

```
plot_rhat(fit, threshold = 1.01, pars = NULL)
```

Arguments

`fit`	A `tulpa_fit` object.
`threshold`	Rhat threshold for warnings (default: 1.01).
`pars`	Character vector of parameter names to include. If NULL (default), includes all main parameters (excludes high-dimensional RE/spatial).

Details

Color coding:

- Green: Rhat < 1.01 (converged)
- Yellow: 1.01 <= Rhat < 1.05 (borderline)
- Red: Rhat >= 1.05 (not converged)

Value

A ggplot object (if ggplot2 available) or base R plot (invisible).

See Also

`plot_ess()`, `diagnostic_summary()`, `diagnostics()`

Examples

```
# Diagnostic plots require a fitted tulpa model
# See tulpa() examples for fitting models


set.seed(123)
df <- data.frame(x = rnorm(60))
df$y <- rpois(60, exp(0.5 + 0.3 * df$x))
fit <- tulpa(y ~ x, data = df, family = "poisson", mode = "hmc",
             control = list(n_iter = 500L, warmup = 250L, n_chains = 2L,
                            seed = 1L))
plot_rhat(fit)
```

`posterior_predict` *Posterior predictive replicates*

Description

Draw replicated responses from the posterior predictive distribution: the in-sample linear predictor is drawn with every component the fit estimated – fixed effects, formula random effects, the offset, and any spatial or temporal field – and pushed through the family's sampling distribution. The same draws give the pointwise log-likelihood behind `compare_models(criterion = "waic")` / `"loo"`.

Where the draws come from follows what the fit carries:

- A ModelData sampler fit (`mode = "hmc"` and its siblings) evaluates the engine's own linear predictor at each draw, so a field is drawn jointly with everything else.
- A nested-Laplace fit draws from its outer grid: a replicate picks a cell by its weight and draws each observation from that cell's Gaussian for the linear predictor (`fitted_eta`, `fitted_eta_var`). The cell's joint covariance across observations is not retained, so observations within a replicate are independent given the cell. A fit run with `control$fitted_var = FALSE` carries no `fitted_eta_var`, and its replicates hold the across-cell spread only.
- Any other fit carries its coefficients rather than its linear predictor. Fits with posterior draws use them (fixed and random effects jointly per draw). A Laplace or EB fit (`mode = "laplace"` / `"eb"`) samples the fixed effects from `N(coef(fit), vcov(fit))` and each draw's random effects from their Gaussian conditional given those fixed effects under the joint Laplace precision of `[beta | b]` at the mode, so the two carry their posterior correlation (on a conditional Laplace fit this is the joint Gaussian draw; an EB fit is conditional on the estimated variance components). A conditional-Laplace spatial fit holds its random effects and areal field at their posterior mode, which over-disperses the linear predictor wherever the intercept is aliased with them; an SPDE field enters at its posterior mean.

At `newdata` the prediction is population level (random effects at zero), matching `predict.tulpa_fit()`.

A zero-inflated fit (`ziformula`) draws the structural-zero logit from the same posterior draw as the count predictor; each replicate observation is a structural zero with probability `plogis(X_zi beta_zi)` and otherwise a draw from the family (a zero-truncated family gives the hurdle model).

Usage

```
posterior_predict(object, ...)

## S3 method for class 'tulpa_fit'
posterior_predict(
  object,
  newdata = NULL,
  ndraws = NULL,
  n_trials = NULL,
  seed = NULL,
  ...
)
```

Arguments

`object`	A `tulpa_fit` object from `tulpa()`.
`...`	Passed to methods.
`newdata`	Optional data frame of covariates to predict at. Population level (fixed effects only); `NULL` (default) replicates at the training data with every fitted component included.
`ndraws`	Number of posterior draws to use. Defaults to all stored draws, or 400 on the draw-free Laplace tier.
`n_trials`	Binomial / beta-binomial trial counts for the replicates. Defaults to the training trials when `newdata` is `NULL`, else 1.
`seed`	Optional integer seed (RNG state is restored on exit).

Value

A `ndraws x n_obs` numeric matrix of replicated responses.

See Also

`pp_check()`, which uses these replicates; `simulate.tulpa_fit()`.

Examples

```
set.seed(1)
d <- data.frame(y = rpois(100, 4), x = rnorm(100))
fit <- tulpa(y ~ x, data = d, family = "poisson", mode = "laplace")
yrep <- posterior_predict(fit, ndraws = 100)
dim(yrep)  # 100 x 100
```

posterior_sample — *Posterior parameter sample from a fit*

Description

Returns a fit's posterior draws for summary purposes (quantiles, derived quantities, density plots), regardless of how they were produced – an MCMC chain, a nested-Laplace node mixture, or a variational sample all answer here. For the chain-only view used by convergence diagnostics, see `mcmc_draws()`.

Usage

```
posterior_sample(fit)
```

Arguments

`fit`	A `tulpa_fit` (or subclass) carrying posterior `$draws`.

Details

A fit that carries no draws returns `NULL` with a message naming its backend, the posterior representation it carries instead, and the accessor that turns that representation into draws where one exists – an absent draws matrix is a property of the backend, not a failure of the accessor, and saying which is the difference between a diagnosable answer and a bare `NULL`.

Value

The posterior draws matrix/array, or `NULL` if the fit carries none.

See Also

`mcmc_draws()`, `tulpa_posterior_draws()`, `diagnostics()`

Examples

```
set.seed(1)
df <- data.frame(x = rnorm(80))
df$y <- rpois(80, exp(0.5 + 0.3 * df$x))
fit <- tulpa(y ~ x, data = df, family = "poisson")
dim(posterior_sample(fit))
```

post_hoc_lm *Fit a post-hoc linear model on estimated parameters*

Description

Useful for exploring drivers of occupancy/detection/abundance variation after model fitting. Fits a weighted linear model and optionally generates bootstrap confidence intervals.

Usage

```
post_hoc_lm(
  formula,
  data,
  weights = NULL,
  n_boot = 1000L,
  probs = c(0.025, 0.975)
)

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

Arguments

formula	Model formula (e.g., psi_hat ~ trait1 + trait2).
data	A data.frame with response and predictors.
weights	Optional weights (e.g., inverse of standard errors).
n_boot	Number of bootstrap replicates for CI (default 1000, 0 to skip).
probs	Quantile probabilities for bootstrap CI (default 0.025, 0.975).
x	A post_hoc_lm result.
...	Ignored.

Value

A list of class "post_hoc_lm" with:

summary data.frame of coefficient estimates and CIs

lm_fit the underlying lm object

boot_coefs matrix of bootstrap coefficient samples (if n_boot > 0)

R2 R-squared from the fitted model

Examples

```
# Explore drivers of per-site estimates after fitting a model.
site <- data.frame(
  psi_hat = c(0.2, 0.5, 0.8, 0.4, 0.6, 0.3),
  se      = c(0.05, 0.04, 0.06, 0.05, 0.03, 0.05),
  trait   = c(1.0, 2.5, 3.8, 1.9, 3.1, 1.2)
)
fit <- post_hoc_lm(psi_hat ~ trait, data = site,
                   weights = 1 / site$se^2, n_boot = 200L)
fit
```

---

pp_check *Posterior predictive check*

Description

Generate posterior predictive checks for a fitted tulpa model. Compares observed data to replicated data from the posterior predictive distribution.

Usage

```
pp_check(object, ...)

## S3 method for class 'tulpa_fit'
pp_check(
  object,
  type = c("dens_overlay", "scatter", "intervals", "stat"),
  component = NULL,
  stat = mean,
  ndraws = 50,
  ...
)
```

Arguments

object	A tulpa_fit object
...	Additional arguments passed to bayesplot functions
type	Type of check: "dens_overlay", "scatter", "intervals", "stat"
component	Which component: "numerator", "denominator", or "both"
stat	Function for "stat" type (default: mean)
ndraws	Number of posterior draws to use (default: 50 for plots)

Value

A ggplot object

Examples

```
# pp_check is a generic; model packages (e.g. tulpaObs, tulpaRatio) provide
# the posterior-predictive method for their fits.

set.seed(123)
n <- 200L
df <- data.frame(y = rpois(n, 5), x = rnorm(n),
                 site = factor(rep(1:10, each = 20)))
fit <- tulpa(y ~ x + (1 | site), data = df, family = "poisson",
             mode = "hmc", control = list(n_iter = 500, warmup = 250))
# Density overlay (dispatches to the model package's pp_check method).
if (requireNamespace("bayesplot", quietly = TRUE)) {
  pp_check(fit, type = "dens_overlay")
}
```

`predict.tulpa_fit` *Predict at new covariate values (population level)*

Description

Prediction of the linear predictor at `newdata`, on the link or response scale. The fixed-effect part is `X beta` with credible bounds from the fixed-effect covariance (`vcov()`). For a fit carrying a continuous spatial field, the posterior-mean field is interpolated (kriged) to the `newdata` coordinates and added to the linear predictor by default, so `predict()` gives the conditional (location-specific) prediction. Three continuous field families are supported: an SPDE Matern field (`spatial_spde()`), projected through the mesh; a Hilbert-space GP field (`spatial_gp(approx = "hsgp")`), where the Laplacian basis is re-evaluated at the new coordinates (with the training centring / boundary); and a GP / NNGP field (`spatial_gp()`), interpolated by the NNGP conditional mean at each new location's nearest training locations. The HSGP and GP/NNGP fields are marginalised over the hyperparameter grid (not plugged in at the posterior mean). Ordinary random effects are held at zero (population level); add group effects from `ranef()` when needed. An areal (ICAR / BYM2 / CAR) or temporal (RW1 / RW2 / AR1) field is held at zero in the same way, at the training design too; the in-sample linear predictor with every fitted component is what `posterior_predict()` draws and what `compare_models(criterion = "waic")` / `"loo"` score.

Usage

```
## S3 method for class 'tulpa_fit'
predict(
  object,
  newdata = NULL,
  type = c("link", "response"),
  se.fit = FALSE,
  level = 0.95,
  include_field = TRUE,
  ...
)
```

Arguments

`object`	A `tulpa_fit` object.
`newdata`	Data frame of covariates (and, for an SPDE fit, the coordinate columns named in the spec's coordinate formula). If `NULL`, predicts at the training design (requires `$model_matrix`).
`type`	`"link"` (linear predictor) or `"response"` (mean scale, `E[y]`). For a binomial fit the `"response"` scale here is the per-trial success probability `g^{-1}(eta)` (there is no `n_trials` at `newdata`); this differs from `fitted()`, which returns the trial-scaled expected count at the training design. On a zero-inflated fit (`ziformula`) the `"link"` scale is the count predictor and the `"response"` scale is the mixture mean `(1 - pi) E[y | eta]`, `pi = plogis(X_zi beta_zi)`, with the zero-inflation design rebuilt from the fit's `ziformula` at `newdata`.
`se.fit`	If `TRUE`, also return the link-scale standard error and credible bounds. With an included SPDE field the SE propagates the joint (fixed-effect, field) posterior precision at the fitted hyperparameters – including the cross term – conditional on `(range, sigma)` (a nested fit's hyperparameter-grid spread is not propagated, so the bound is mildly optimistic when that posterior is wide). Integer-nu, no-RE SPDE fits only; other layouts decline with an explanation. On a zero-inflated fit `se.fit` is the count predictor's, and the response-scale bounds are quantiles of the mixture mean over draws of the count and zero-inflation coefficients jointly (pinned, so repeated calls agree).
`level`	Credible-interval level (default 0.95).
`include_field`	For a continuous-spatial fit (SPDE, HSGP, or GP/NNGP), add the kriged field to the prediction (default `TRUE`). `FALSE` gives the fixed-effect (population) prediction. Ignored for fits with no continuous field. For an HSGP or GP/NNGP fit the field is added to the point prediction but its uncertainty is not yet propagated into `se.fit` (the interval reflects the fixed-effect covariance only).
`...`	Ignored.

Value

If `se.fit = FALSE`, a numeric vector. If `se.fit = TRUE`, a data frame with `fit`, `se.fit` (link scale), `lower`, `upper` on the requested scale.

`print.tulpa_fit` *Summarize and print a tulpa fit*

Description

`print()` shows a fit in a few lines: how it was produced (backend and inference tier), the call, the observation count and the fixed-effect point estimates. `summary()` returns the full report as a `summary.tulpa_fit` object, which prints as a sectioned display and whose parts are read by name; `coef()` on it returns the fixed-effect table.

Usage

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

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

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

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

Arguments

`x`	A `tulpa_fit` object (`print.tulpa_fit()`) or a `summary.tulpa_fit` object (`print.summary.tulpa_fit()`).
`...`	Ignored.
`object`	A `tulpa_fit` object.
`level`	Credible-interval level of the fixed-effect table, a single number in `(0, 1)` (default 0.95).
`digits`	Significant digits in the printed tables.

Details

The fixed-effect table reads the same posterior `coef()`, `confint()` and `vcov()` read. Sampler tiers report empirical quantiles; the Laplace tier reports the Gaussian approximation, and so does any fit whose reported posterior is a closed-form Gaussian (expectation propagation, the multinomial Laplace fit), whose draws are then samples from that Gaussian and are not what the summary reads.

On a nested-Laplace fit the estimate and standard error are the hyperparameter-grid-marginalized moments, and the bounds invert the Gaussian mixture `sum_k w_k N(mu_kj, V_kjj)` that grid defines, rather than reading `mu +/- z sigma` off the single Gaussian matching those moments. The table's `interval_source` attribute records which read produced them (`"mixture_cdf"`, `"gaussian_moment"`, `"skew_map_cell"`, or `"skew_map_cell/mixture_cdf"` when the two are in play on different coefficients) and `interval_declined` says why, whenever the mixture read did not run. Its `retained_mass` attribute gives the share of the grid weight whose cells retained a fixed-effect block: 1 on a complete grid, and below 1 on one that dropped a positive-weight cell, whose report is then the posterior conditional on the cells that remain.

Where no per-cell block was retained the estimate falls back to the grid-weighted average of the per-cell modes, restricted to the cells whose inner solve reached a mode. A fit where none did reports `NA` with `interval_declined = "not_converged"`, rather than the vector its Newton started from as an estimate.

With `control$skew_correct = TRUE` a coefficient whose inner-Laplace `gamma_3` is in the band it is valid on reports Cornish-Fisher quantiles instead, and the table's `skew_applied` attribute names which coefficients took the correction. That correction is measured at the MAP cell, so it is reported

on its own and is not composed with the mixture read; a coefficient it declines keeps the mixture read rather than falling back further.

The reliability headline is read from `diagnostics()`, so it answers the question that applies to how the draws were produced: Rhat and effective sample size for a chain, the approximation-reliability band (outer Pareto-k, grid quadrature, inner Laplace, combined verdict) for i.i.d. draws from a deterministic approximation, and nothing for a point summary.

Value

`print.tulpa_fit()` and `print.summary.tulpa_fit()` return their input invisibly.

`summary.tulpa_fit()` returns a list of class `summary.tulpa_fit`:

`call`, `formula`, `family` as the fit records them, `NULL` where it records none.

`inference` how the fit was produced: a list of `backend`, `inference_tier`, `inference_mode` and `selection_reason` (why `mode = "auto"` chose that backend).

`n_obs`, `n_draws` the observation count and the number of posterior draws the fixed-effect summaries read (`NULL` without draws).

`level` the interval level of `coefficients`.

`coefficients` data frame, one row per fixed effect named by the term: `estimate`, `std.error` and the lower / upper credible bounds, labelled as `confint.tulpa_fit()` labels them. Carries the `interval_source`, `interval_declined`, `retained_mass` and `skew_applied` attributes described in Details. Also returned by `coef()` on the summary.

`random` the random-effect standard deviations as `VarCorr()` reports them, with their estimated / sampled / conditioned label; `NULL` on a fit with no random-effect terms.

`hyperparameters` data frame of the integrated hyperparameters, one row each: `parameter`, `mean`, `sd`, `median` and the 2.5% / 97.5% bounds `ci_lo` / `ci_hi`. On a nested-Laplace fit these are the outer-grid axes; on the random-effect covariance integrators the derived standard deviations and correlations. `NULL` where the fit integrated none.

`sigma` the residual / dispersion scale a fit reports as one number, else `NULL`.

`smooth_terms` the fit's `s(...)` term metadata, `NULL` without smoothers; `smooth_effects()` returns the fitted curves.

`reliability` a list: `kind` (`"chain"`, `"iid"` or `"point"`, the draws provenance `diagnostics()` routes on); on a chain `rhat_max`, `ess_bulk_min` and `ess_tail_min` over the parameters; on i.i.d. draws `band`, the one-row headline `diagnostics()` attaches as its `summary` attribute (outer `pareto_k` and its `band`, grid quadrature ESS, inner-Laplace scores, `reliability`), and `moment_source`, which read the per-parameter moments came from; `verdict`, the whole-fit reading (`"converged"` / `"not converged"` on a chain that records its convergence check, the combined `reliability` verdict on i.i.d. draws, `NA` otherwise); `convergence`, the fit's own convergence record; and the backend-specific quality numbers `mean_accept` (Metropolis acceptance rate), `n_divergent` (divergent transitions) and `elbo` (the variational objective), each `NULL` where the backend has none.

`axis_fields_dropped` the grid axes the fit's own resolved path could not read, one row per dropped field (block, type, field, path, integrates, reason); `NULL` whenever every supplied axis was used. `diagnostic_summary()` reads the same record in sentences.

`prior` a list whose `beta` is the Gaussian fixed-effect prior the fit ran under, as `list(mean, sd)`: the engine default `prior_normal(0, 2.5)` whenever the caller supplied none, and `NULL` on the paths that express no Gaussian prior on the fixed effects.

`coef.summary.tulpa_fit()` returns the `coefficients` data frame.

See Also

`coef.tulpa_fit()`, `confint.tulpa_fit()`, `VarCorr()`, `diagnostics()`, `tulpa_outer_grid()`

Examples

```
set.seed(1)
df <- data.frame(x = rnorm(60))
df$y <- rpois(60, exp(0.5 + 0.3 * df$x))
fit <- tulpa(y ~ x, data = df, family = "poisson", mode = "laplace")
fit
s <- summary(fit)
s
coef(s)
s$inference$backend
```

---

print.tulpa_nested_laplace

Nested Laplace approximation for latent Gaussian models

Description

Generic outer-grid nested Laplace driver. Builds a grid over the hyperparameters of a single latent prior block (spatial or temporal), runs an inner Laplace at each grid point with warm-starting, and integrates over the grid to give proper hyperparameter marginals.

Supported priors:

- Spatial (areal): `"icar"` (1D grid on tau), `"bym2"` (2D on (sigma, rho)), `"car_proper"` (2D on (tau, rho); rho lives in the eigenvalue interval (1/lambda_min, 1/lambda_max) of `D^{-1} W`).
- Spatial (continuous): `"nngp"` (2D on (sigma2, phi_gp)), `"hsgp"` (2D on (sigma2, length-scale)).
- Temporal: `"rw1"`, `"rw2"` (1D grid on tau), `"ar1"` (2D on (tau, rho))
- SPDE continuous spatial: see `cpp_nested_laplace_spde()` (separate entry, rebuilds Q via SPDE Q-builder).

Usage

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

tulpa_nested_laplace(
```

```
  y,
  n_trials,
  X,
  prior = NULL,
  spec = NULL,
  data = NULL,
  re_idx = NULL,
  n_re_groups = 0L,
  sigma_re = 1,
  family = "binomial",
  phi = 1,
  offset = NULL,
  likelihood = NULL,
  hyperprior = c("proper", "flat"),
  control = list()
)
```

Arguments

`x`	A nested-Laplace fit.
`...`	Ignored.
`y`	Response vector.
`n_trials`	Trial sizes (binomial). Pass `1L`-vector otherwise.
`X`	Fixed-effects design matrix.
`prior`	A list describing the latent prior block. Required field `type` one of {"icar", "bym2", "car_proper", "rw1", "rw2", "ar1"}. Type-specific fields: • icar: `spatial_idx`, `n_spatial_units`, `adj_row_ptr`, `adj_col_idx`, `n_neighbors` (CSR adjacency, 0-based); optional `tau_grid`. Any block accepts an optional `svc_weight`, one weight per observation, which makes it a varying coefficient: observation `i` contributes `svc_weight[i] * f_i` instead of `f_i`, where `f_i` is the block's field at that row (`z[spatial_idx[i]]` for an areal block, `(A u)_i` for an SPDE one). Absent, the field enters unweighted. • bym2: same adjacency; `scale_factor`; optional `sigma_grid`, `rho_grid`. • car_proper: same adjacency; optional `tau_grid`, `rho_grid`, `rho_bounds = c(lower, upper)` (defaults to (0, 1)). • rw1/rw2: `temporal_idx` (1-based), `n_times`; optional `tau_grid`, `cyclic` (default FALSE). • ar1: `temporal_idx`, `n_times`; optional `tau_grid`, `rho_grid` (each `rho_grid` value strictly inside (-1, 1)). An `icar`, `bym2`, `car_proper` or `nngp` block may carry `projector`, an `N x n_spatial_units` (`N x n_spatial` for `nngp`) matrix (dense or a **Matrix** sparse matrix), in place of `spatial_idx`: observation `i` then receives `(A z)_i`, so restricted spatial regression is `A = S P_perp` with `S` the observation-to-unit incidence. The block is fitted by the multi-block driver, alone or in a list of blocks. An intrinsic field's level is placed from `A 1`: removed with no fold where it is zero, folded into the

intercept where it is the intercept column, otherwise left to the augmentation. The nngp and car_proper priors are proper and carry their level themselves.

Grids of a two-axis type (bym2, car_proper, ar1) are PAIRED when both are supplied: the i-th entries of tau_grid / sigma_grid and rho_grid form the i-th outer-grid cell, so the two must have equal length, and a tensor grid is written as expand.grid() columns. A single supplied axis is kept as an axis (its distinct values) and crossed with the default of the other.

A default grid axis is a starting axis, not a hard ceiling: for icar (tau_grid) and bym2 (sigma_grid) a posterior mode that rails a boundary node (pareto_k_regime = "collapsed_edge") triggers one mode-Hessian recenter-and-refit, reported through outer_grid_placement / outer_grid_recenter_declined – see tulpa_nested_laplace_jo return docs. An axis the caller pinned is never moved; mark a grid your own code defaulted with auto_grid() to keep the recenter live on it.

spec — Optional tulpa_temporal or tulpa_spatial spec object (output of temporal_rw1(), temporal_rw2(), temporal_ar1(), spatial_car(), spatial_bym2(), etc.). When supplied alongside data, the prior list is built automatically via prior_from_spec() – pass either prior or spec, not both.

data — Data frame used to validate spec and resolve time/group/site indices. Required when spec is supplied.

re_idx — Optional 1-based RE group index per obs (defaults to no RE).

n_re_groups — RE group count (default 0).

sigma_re — RE standard deviation (default 1).

family — "binomial", "poisson", "neg_binomial_2", etc.

phi — Dispersion passed to the family, held fixed. One convention at every door: for gaussian / lognormal this is the residual VARIANCE (the SD is sqrt(phi)), for neg_binomial_2 the size, gamma the shape, beta the precision, t the scale; binomial and poisson ignore it. The compiled kernels parameterize the two variance families by the residual SD and are handed sqrt(phi) at the boundary.

Defaulted, it conditions at 1 and says so with a warning, since for a family that reads a dispersion that is a modelling choice rather than a neutral value. fit$phi_estimated records whether the value on the fit was estimated or conditioned on.

offset — Optional per-observation offset added to the linear predictor (length length(y)), as an offset() term in a model formula. Every single- and multi-block kernel carries it, and fitted_eta includes it.

likelihood — Optional model-supplied likelihood, replacing the built-in family. Pass an external pointer to a tulpa::NestedLikelihood (built in a model package's own C++ from a LikelihoodSpec); the inner Laplace solve then reads the per-observation score, Fisher weight, and log-likelihood from that spec instead of family, so family/phi are ignored. Used by model packages to fit a custom response without adding a family to tulpa – for example tulpaObs threads its marginalized single-season occupancy likelihood (a scaled Bernoulli, with the latent occupancy state integrated out) through this. Multi-block prior only. Default NULL (use family).

`hyperprior` The prior an outer hyperparameter axis carries when the call states no density for it, `"proper"` (default) or `"flat"`. `"proper"` folds a normalised density on the axis's integration coordinate into `log_marginal`: the PC prior `P(sigma > 3) = 0.01` on a standard deviation, variance or precision axis, the PC range prior at 0.2 times the coordinates' bounding-box diagonal on a range or lengthscale axis, a uniform on a bounded axis, and the PC + LKJ prior over a free-covariance block. An axis with no sourced density is named in `log_hyperprior_declined`. `"flat"` folds no density of the engine's own, so such an axis is integrated under its cell measure alone, is named in `log_hyperprior_declined` as `"flat_hyperprior"`, and the fit's `log_evidence` declines with `"improper_hyperprior"`. A density the call states – a `prior_*` argument, a block's `rho_prior`, `prior_range` or `prior_sigma`, the copy coefficient's slab, a `tgmrf()` block's own prior – applies under either choice. The same two choices, under the same names, are offered by `tulpa()`, `tulpa_eb()`, `tulpa_re_cov_nested()` and `fit_spde()`.

`control` Optional list of perf/numerical tuning knobs (statistical arguments stay top-level), following the `control` convention of `tulpa()`. Recognised elements (defaults in parentheses):

- `max_iter` (`50L`), `tol` (`1e-6`) – inner Newton iteration budget and tolerance.
- `n_threads` (`1L`) – inner-loop OpenMP threads.
- `x_init` (`NULL`) – warm-start for the first grid point's inner solve.
- `keep_grid_hessians` (`TRUE`) – when `TRUE`, retain per-grid-point fixed-effects marginal Hessian H_β and mode $\hat{\beta}$ on the return list as `$grid_hessians` (list of dense $p \times p$ matrices) and `$grid_modes` (list of length-p vectors). Used downstream by simplified-Laplace (SLA) callers to assemble skew-aware marginals – see the cumulant pooling in `rubins_pool()`.
- `diagnose_k` (`TRUE`), `k_samples` (`500L`) – compute the outer Pareto-$\hat{k}$ accuracy diagnostic (`$pareto_k`) by importance sampling the hyperparameter posterior against the Gaussian proposal fitted to the grid, drawing `k_samples` extra inner-marginal evaluations. `k_samples` is a precision knob: the GPD tail size is held at the fraction the default budget implies, so a larger budget sharpens the same k-hat instead of moving it to a deeper quantile of the weight distribution (gcol33/tulpa#631). `k_tail_points` overrides that tail size directly (capped at 20% of the draws). Computed for a single-block, single positive-scale-axis grid; left `NA` (with the grid's quadrature ESS as the fallback diagnostic) for multi-block, multi-axis, or bounded-parameter grids. See `tulpa_psis()`.
- `diagnose_skew` (`TRUE`), `skew_idx` (`NULL`) – compute the inner-Laplace skewness diagnostic (`$inner_skew`, gamma_3, Rue Martino & Chopin 2009 Sec 3.2.3) at the fitted MAP grid cell: one extra Newton solve, scoring the `p` fixed-effects latent indices by default (pass `skew_idx`, 1-based latent indices, to score additional ones, e.g. specific spatial units – the full latent field is not scored by default since it costs one linear solve per index). This is the complementary layer to `diagnose_k`: the outer diagnostic scores the hyperparameter-grid integration around a FIXED inner Laplace, this scores whether that inner Gaussian approximation is itself a good fit to the latent-field conditional posterior. See `diagnostics()` for the combined whole-fit verdict.

- `within_cell` (`"log_quadratic"`) – the WITHIN-CELL construction the reported per-axis hyperparameter intervals are read with. The outer grid's weights say how much mass each cell holds; they do not say how it is spread inside the cell, and a quantile needs both. `"log_quadratic"` joins each row's node log densities by overlapping quadratics, exact for a Gaussian at any spacing, and continues the end quadratic past the outer node instead of stopping at the box edge; it runs on the rows that resolve their own conditional (`h / sd <= 2`) and reads every other row as boxes, declining to `"box_uniform"` where no row qualifies. `"box_uniform"` puts the cumulative FULL mass at each cell EDGE and interpolates between edges; `"chord"` puts the cumulative MID-mass at each cell coordinate and interpolates between coordinates. The default moved from `"box_uniform"` at gcol33/tulpa#932: a placed axis sits at 1.25 posterior SDs per cell, where the box read is narrow by construction, and against dense references of a two-arm ICAR fixture the log-quadratic read's summed error is 0.038 against the box read's 0.110 on the same grids. `outer_grid_h_over_sd` is how wide a cell is on each axis, and `theta_within_cell` is what each axis was actually read with. Only a `"density"` support admits the box and log-quadratic reads – a CCD design, a locally refined grid and a posterior sample are not cell partitions that tile – and an axis they decline on reports the read that ran with a reason rather than erroring. Nothing else moves: point estimates, moments, draws and weights are untouched.

- `skew_correct` (`TRUE`) – consume the inner-Laplace expansion instead of only grading it: report Cornish-Fisher marginal quantiles at each coefficient's own `gamma_3`, about the centre `gamma_1 + gamma_3 / 2` that Rue, Martino & Chopin (2009) eq. (22) implies, from `summary()` / `confint()` wherever the combined inner band says the leading-order expansion is in its regime, and the Gaussian quantiles everywhere else. It is post-processing on the reported quantiles: draws, modes and weights are untouched, so a fit run with it off is bit for bit the fit it was before. A coefficient whose location term could not be formed declines rather than reading it as zero. The band that bounded the relocation itself (`centre_unreliable`) is off (`Inf`): scored over seven fixtures with an exact reference, every finite cutoff declines the coefficients the correction helps most, because a large centre carrying a small `gamma_3` is uniformly weak correlation rather than an expansion out of its regime. `$skew_correction` records the per-coefficient `gamma_3`, `gamma_1` and the centre they form, the band, the inner importance k-hat, the combined band, the eligibility and the reason behind it; the `skew_applied` attribute on `summary()` / `confint()` records what was actually used at the requested level. RMC fit a skew normal here instead; the series correction is the same-order alternative, and unlike a skew normal its skewness does not saturate inside the band it is applied on.
  MEASURED. Against exact quadrature quantiles of rare-event binomial-logit posteriors it cuts total absolute endpoint error 69.2%, improving both endpoints in every case. Scored over the WHOLE marginal – paired CRPS against the exact posterior in a 400-replicate prior-predictive experiment – it reads -0.01643 against the uncorrected Laplace at t = -1.89, essentially all of the -0.01662 the exact posterior itself achieves, and its PIT re-enters the

simultaneous SBC band. Applied about the Laplace mode instead of about gamma_1 + gamma_3 / 2 the same reshaping scored +0.00775 at t = +3.54, a net loss; the location term is what supplies that centre.
IT IS ON BY DEFAULT, so summary() / confint() on a nested-Laplace fit report the corrected quantiles wherever the combined inner band admits the coefficient; skew_correct = FALSE restores the uncorrected report exactly. Scored against the mixture read a correction-off fit gives, the flip is t = -1.895 on the rare-event intercept and -3.765 / -3.201 on the small-group Bernoulli design. Across twelve model classes read off one solve per seed, pooled 95% coverage moves 0.9510 -> 0.9542 at a standard error of 0.0070, with every class inside the acceptance the shipped gates use. A fit the correction cannot help – a coupled one, whose location term is unreachable – reports what it reported before, to the bit.

- subspace_debias (FALSE) – correct only the latent directions the inner-layer diagnostics flagged, by exact Metropolis, and leave the rest at their Gaussian conditional. TRUE takes every default; a list overrides band (the inner-reliability floor a coordinate is selected at, default "ok"), idx (pin the corrected set explicitly, skipping the selector), closure / closure_max (grow the set by strongly coupled precision-graph neighbours: a coordinate strongly coupled to a member of the corrected set and left out of it is carried linearly, which is the error the correction removes, so requesting the closure also retains the modal cell's joint precision for the selector to read), the sampler budget n_iter / warmup / thin, and n_draws. The selector reads the per-index bands diagnose_skew already attached, so it costs no extra solve; the correction itself re-runs the settled grid once with the sampler on, and the fit then reports $draws – the per-cell Metropolis sample for the selected coordinates, the rest from the Gaussian conditional given them – instead of the Gaussian-mixture moments. An EMPTY selection leaves the fit bit-for-bit identical to the plain path. $subspace_debias records what was selected, the bands it was read from, and the per-cell acceptance rate. Requesting it turns keep_grid_hessians on, since the recombination reads exactly those per-cell pieces.
- cila (FALSE) – corrected integrated Laplace, the second inner-layer debias (after Lai, Margossian and Sheldon, arXiv:2605.20345). Where subspace_debias selects coordinates and runs exact Metropolis on them, this selects nothing: at every outer cell it draws n_points points from the whole inner Gaussian, weights each by the exact joint density it came from, and reports the weighted particles. TRUE takes the defaults; a list overrides n_points (1024L), variant ("qmc", a Sobol net; "is" for iid draws, "rqmc" for the net under n_shift random shifts), n_shift (8L), n_draws and seed. Below 512 points a cell's particle set is too coarse to be a marginal at all and the request is refused. The corrected per-cell masses become the fit's own weights / log_marginal and weights_source reports "cila"; the pre-correction pair is kept as $cila$laplace. A cell whose inner solve factorized sparsely draws through the CHOLMOD factor's own triangular and permutation solves; an LDL' factor has no square root to draw with and is declined with "sparse_factor_not_ll".
- auto_recenter (TRUE) – outer-grid placement policy. TRUE re-centres the

movable default axes on the posterior mode and refits when the grid either RAILS (an axis's own marginal is maximal at one of its own endpoints) or does not RESOLVE its posterior (an axis's node spacing exceeds 2 posterior SDs in its own coordinate). Both tests read the weights the fit already stored, so a grid that brackets and resolves its mode costs nothing beyond them; when the pass does fire it is a second full grid solve plus a finite-difference mode/Hessian stencil. A recentred axis is `mode +/- 2.5 sd` over 5 nodes, a cell width of 1.25 posterior SDs by construction, against a census median of 3.9 on the fixed spans.

Measured over 200 fixed-truth seeds on each of six configurations (icar chain / icar lattice / rw1 / bym2 / iid / nngp), the default moves mean |coverage - nominal| from 0.043 to 0.030 at the 95% level, 0.171 to 0.084 at 80% and 0.243 to 0.129 at 50% against the rail-only policy, at 0.63 times the 95% interval width and 0.76 times the median bias, for 1.71 times the wall clock.

Three other values. `"rail"` is the rail test alone, which is what `TRUE` meant before the sizing measurement settled the default. `FALSE` integrates over the grid exactly as given, whatever it is, and records `outer_grid_recenter_declined = "auto_recenter_disabled"`. `"always"` re-centres every movable default axis whatever the fit did, at 2.04 times the wall clock; it agrees with the default seed for seed on five of the six measured configurations and differs on the one whose default axes already resolve their posterior, where it takes 50% coverage to 0.135 against a nominal 0.5.

Which families carry movable axes is `.NL_REGISTRY_AXIS_FIELD`: icar, rw1, rw2, iid, bym2, nngp, hsgp and spde. car_proper, ar1 and hsgp_mo each carry a correlation axis with no guessable coordinate, and mcar / miid / tgmrf hold their axes in one matrix field; a fit of any of them records which through `outer_grid_recenter_declined` rather than passing in silence. The per-axis policy names are the standalone registry path only – `tulpa_nested_laplace_joint()` and `fit_st_nested()` refuse them rather than accept them and ignore them.

- `max_grid_cells` (`NULL`, no ceiling) – a ceiling on a multi-block outer grid's cell count, one inner Newton solve per cell; a grid past it is refused before any solve, naming the axes that produced the count. Unset, a grid of any size is solved, and one past 2048 cells is announced with a warning first.
- `checkpoint` (`list(path =, resume =)`) – grid-cell checkpoint / resume. Each solved outer cell is appended to `path`, keyed by its hyperparameter coordinate; `resume = TRUE` loads the finished cells and solves only the rest, and a fingerprint mismatch (different data, settings or grid) errors rather than resuming onto a stale result. See `?tulpa_nested_laplace` "Checkpoint / resume".
- `progress`, `progress.every`, `progress.file`, `progress.throttle` – the outer-grid progress reporter: whether to print, how often (in cells), where to, and the minimum seconds between lines.
- `prune` (`TRUE`), `prune_tol` (`1e-3`), `screen_iters` (`.NL_SCREEN$iters`, 2 – the depth gcol33/tulpa#640 measured the screen down to) – cheap-pass screening of the outer grid. The driver first sweeps the lattice running a `screen_iters`-step inner Newton per cell, each warm-started from the

previous screened cell's quasi-mode, computes a screening Laplace log-marginal, softmax-normalises it, and skips the full inner Newton on every cell whose screened weight is below `prune_tol`. The neighbour warm-start keeps each cheap mode near its cell's own mode, so the cheap ranking tracks the full-solve ranking even where the latent mode moves across the grid. Pruned cells get `log_marginal = -Inf`, `n_iter = 0` and inherit the pilot mode; the pilot cell is never pruned, and at least `prune_min_keep` cells (5) are solved in full whatever the tolerance says – the highest-ranked dropped cells are restored up to that floor, because the outer grid placement pass reads a finite-difference curvature off the cells that were SOLVED and a kept set of one carries none. After the full pass the posterior mass the dropped cells may carry is bounded by lifting each by the worst screening error measured on a kept cell (`prune_dropped_mass_bound`); past 1% the driver solves the dropped cells that contribute most to it until it is under 1% (`prune_n_repaired`), and a safety gate falls back to the full grid (with a warning) if the bound still fails. The bound assumes a dropped cell is screened no worse than the worst kept one. On by default (gcol33/tulpa#925); `FALSE` solves every cell. `prune_tol` must be a finite numeric in `[0, 1)` and `screen_iters` a single integer `>= 1`; both are validated whatever `prune` says. `prune` / `prune_tol` reach both the single-block and the multi-block dispatch; the screening DEPTH is declared by the single-block kernels only, so a multi-block prior refuses a pinned `screen_iters` rather than accepting it and ignoring it.

- `prune_log_gap` (unset) – the screening cut stated in nats instead of as a normalised weight: keep every cell within this many nats of the best screened cell. `prune_tol` is a weight, so what it cuts at is the gap `-(log(prune_tol) + log(Z))`, and the whole range a caller would type (`1e-3` to `1e-12`) is 7 to 28 nats of it – on an outer surface whose log-marginal spans thousands of nats every setting returns the same kept set. Stating the gap directly keeps the knob's resolution wherever the surface is steep. It reaches the kernel as `prune_tol = exp(-gap)`, so the realised cut is the requested gap less `log(Z)`, at most `log(n_grid)` nats narrower; the fit reports the realised value. Setting both this and `prune_tol` is an error – they state one cut in two units.
- `fitted_var` (`TRUE`) – fill the per-row predictive variance `fitted_eta_var`, the companion of `fitted_eta` a caller marginalises the per-row linear predictor with. It costs one back-solve per distinct loading vector per cell, which on a design with few repeated rows is the dominant cost of a cell, so a caller reading only `fitted_eta` or the coefficient summaries sets `FALSE` and the fit then carries no `fitted_eta_var`. Every single-block field reports it, the NNGP, HSGP, SPDE and spatiotemporal fields included, each read off its cell's own precision at the mode; a multi-block prior refuses `FALSE` rather than accepting it and ignoring it.
- `var_of_means_consistency` (`TRUE`) – the consistency pass of `tulpa_nested_laplace_joint()`: a field SD (`tau` of an `icar`, `rw1` or `rw2` block, `sigma` of an `iid`, `bym2` or `spde` block) whose marginal has collapsed onto too few nodes to carry a spread has levels added in every row of the other axes that holds the posterior until it resolves, and the new cells are solved and carried like the rest

(var_of_means_consistency_info records the pass). Other axes keep the nodes they were laid on.

- adaptive_grid (FALSE), adaptive_grid_edge_thresh (0.02), adaptive_grid_max_passes (1L) – the opt-in boundary / interior pass, on the same axes.
- axis_refine (NULL) – per-axis override of how far those axes may be moved, named by the grid's axes ("tau"; "b1.tau" on a multi-block prior): "none", "densify" (inside the declared span) or "extend". Nodes the caller wrote down densify by default and an engine-placed axis extends, as in tulpa_nested_laplace_joint().

Details

print() shows the integrated hyperparameters, the outer-grid size, the outer Pareto-$\hat{k}$ diagnostic when present and the wall-clock timing line ("fit in 5h 25m (grid 2h 09m)") when fit$timing is attached, then the fixed effects.

Value

A list with:

- theta_grid: matrix or vector of grid hyperparameter values.
- log_marginal: log p(y, mode | theta_k) at each grid point.
- weights: integration weights normalising to sum 1.
- theta_mean, theta_sd: posterior moments per hyperparameter.
- n_iter: inner Newton iterations per grid point.
- modes: matrix [n_grid x n_x] of inner modes, when stored.
- pareto_k, pareto_k_is_ess: outer Pareto-$\hat{k}$ and its importance-sampling ESS (NA when not computed for the grid; see control$diagnose_k).
- pareto_k_proposal_source, pareto_k_first_pass: which proposal family the reported k-hat came from (the outer k-hat is scored against several candidates and the best is kept), and the k-hat of the FIRST pass – the proposal exactly as this backend placed it, before any candidate refined or replaced it. A large gap between the two says the nodes are badly scaled around the hyperparameter posterior even though the verdict is fine; no gap says the placement was already right.
- inner_skew, inner_skew_idx, inner_skew_dropped: the inner-Laplace skewness diagnostic (gamma_3) at each scored latent index and its 1-based index, plus a count of (index, observation) contributions dropped for a non-finite third derivative (see control$diagnose_skew).
- timing: named numeric of wall-clock seconds (total, setup, grid, postproc, diagnostics); the grid phase is the inner Laplace pass that scales with grid size. Surfaced one-line in print.
- prune_cheap_log_marginal, prune_mask, prune_n_pruned, prune_tol: present when the grid was screened and the safety gate did not trip – the cheap-screen log-marginal per cell (the truncated solve's value plus half its Newton decrement, the second-order estimate of the converged value the cells are ranked by; the decrement itself is prune_screen_decrement), the logical mask of pruned cells, the pruned-cell count, and the threshold actually applied.

- prune_log_gap_cut, prune_cheap_lm_spread, prune_min_keep, prune_n_floor_restored: the same screen read on the scale it operates on – how many nats below the best screened cell the tolerance cut at, how many nats the screened surface spans, the kept-cell floor applied, and how many cells that floor put back. A cut that is a sliver of the spread is a tolerance with no resolution on this grid.
- prune_dropped_mass_bound, prune_n_repaired, prune_gate_mass: the posterior mass the dropped cells may carry, with each lifted by the worst screening error measured on a kept cell; how many dropped cells the driver solved in full to bring that under the gate; and the gate.
- prune_fallback_triggered, prune_fallback_reason: present only when the gate did trip; the rest of the fit is then the full-grid result and the reason string records which check fired.
- prior: echoed input.

References

Rue, Martino & Chopin (2009). Approximate Bayesian inference for latent Gaussian models by using integrated nested Laplace approximations. *JRSS-B* 71(2):319-392.

Examples

```
set.seed(1)
S <- 30L                                    # spatial units arranged in a chain
nb <- lapply(seq_len(S), function(s) setdiff(c(s - 1L, s + 1L), c(0L, S + 1L)))
nn <- lengths(nb)
field <- as.numeric(scale(cumsum(rnorm(S, 0, 0.4))))   # smooth spatial field
idx <- rep(seq_len(S), each = 6L); n <- length(idx); x <- rnorm(n)
y <- rbinom(n, 1L, plogis(-0.2 + 0.6 * x + field[idx]))
prior <- list(type = "icar", n_spatial_units = S, spatial_idx = idx,
              adj_row_ptr = c(0L, cumsum(nn)), adj_col_idx = unlist(nb) - 1L,
              n_neighbors = nn, tau_grid = c(0.5, 1, 2, 4, 8))
fit <- tulpa_nested_laplace(y, rep(1L, n), cbind(1, x), prior = prior,
                            family = "binomial")
fit$theta_mean        # marginalized ICAR precision
```

---

print.tulpa_nested_laplace_joint

Joint multi-likelihood nested Laplace approximation

Description

Outer-grid nested Laplace driver for *joint* models – multiple response arms sharing one latent prior block, parameterized as a per-arm field amplitude (sigma) on a unit-precision latent.

Supported priors:

- "bym2" – outer grid over (sigma, rho [, alpha]). Latent: phi (n_s) | theta (n_s) with unit-precision ICAR + iid components.

- "icar" – outer grid over (sigma [, alpha]). Latent: phi (n_s) with unit-precision ICAR.
- "car_proper" – outer grid over (sigma, rho_car [, alpha]). Latent: phi (n_s) with Q = D - rho_car * W.

Other backends (NNGP, HSGP, RW1/2, AR1) follow the same interface and land under Phase 3.

Usage

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

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

tulpa_nested_laplace_joint(
  responses,
  prior,
  copy = NULL,
  phi_grid = NULL,
  prior_sigma = NULL,
  prior_alpha = NULL,
  prior_phi = NULL,
  hyperprior = c("proper", "flat"),
  cell_coupling = "separable",
  control = list()
)
```

Arguments

x — A joint nested-Laplace fit.

... — Ignored.

responses — A named list of arm specs (length >= 1). Each arm:

- y – numeric [N_arm] response.
- n_trials – integer [N_arm] (use rep(1L, N_arm) for non-binomial).
- X – numeric matrix [N_arm x p_arm] fixed-effects design.
- spatial_idx – integer [N_arm], 1-based map obs -> spatial unit.
- re_idx – optional numeric [N_arm] 1-based RE group index; defaults to rep(0, N_arm) (no RE).
- n_re_groups – optional integer (default 0L).
- sigma_re – optional numeric (default 1); ignored when n_re_groups == 0.
- family – one of "binomial", "gaussian", "poisson", "neg_binomial_2", "beta", "lognormal", "gamma", "inverse_gaussian". For "lognormal", y is on the natural scale and the linear predictor eta = E[log y] (identity link on the log scale); the -log(y) Jacobian is included in the kernel's log_lik.

- phi – numeric dispersion (gaussian/lognormal residual SD, negbin size, beta precision); default 1.
- field_coef – optional per-arm field coefficient controlling this arm's multiplier on the shared latent field's amplitude. One of: * numeric scalar (default 1) – constant multiplier. 0 means the arm carries NO field at all (the per-row scatter eta += field_coef * sigma * z is skipped for that arm). * character of length 1 – names an outer-grid hyperparam axis (currently "alpha"); the coefficient varies across the grid. * list(name = , grid =) – embedded axis declaration, equivalent to declaring the axis and naming it on this arm. At most one arm may declare a hyperparam-driven axis (the cover hurdle and occu_cover both need only one). Shared axes across multiple arms are deferred. A single-block copy coefficient is declared here, on the arm – not through a separate copy argument.

prior

A list describing the shared latent prior block. Required field type. Backend-specific fields:

- **bym2**: n_spatial_units, adj_row_ptr, adj_col_idx, n_neighbors, scale_factor (default 1); optional sigma_grid (donor-arm field amplitude, default 5 log-spaced values in [0.1, 3]), rho_grid (default c(0.2, 0.5, 0.8, 0.95, 0.99, 0.999)).
- **icar**: n_spatial_units, adj_row_ptr, adj_col_idx, n_neighbors; optional sigma_grid (default 5 log-spaced values in [0.1, 3]).
- **car_proper**: same as icar plus rho_car_grid (default c(0.5, 0.8, 0.95, 0.99)).

sigma_grid's default is a starting axis, not a hard ceiling: when the fitted field-SD posterior mode rails the top node (the axis's own marginal is maximal there, or the whole grid collapsed onto it: pareto_k_regime = "collapsed_edge", see below), the driver finds the outer posterior mode – a damped Newton search over every outer axis, started at the grid's heaviest cell and read through full inner solves, the same mode-find the CCD integrator places its design with – lays the axis at that mode +/- 2.5 of the SDs its curvature there gives, and refits (up to two attempts, the second adding a light default PC(U=3, alpha=0.01) prior on sigma unless prior_sigma was pinned – see there), so a sparse or strongly-identified species is not silently truncated at 3.0. This engages whether or not control$diagnose_k computed the full outer Pareto-k diagnostic, and only when an axis actually railed. A field SD still on a boundary of its grid after the attempts is named in a warning, with the reason it stayed, in outer_grid_axis_declined (gcol33/tulpa#919). The mode the placement was laid around is on the fit as outer_mode_u / outer_mode_cov_u (in each axis's outer_mode_axis_tags coordinate), with outer_mode_status, outer_mode_rounds and outer_mode_evals. A sigma_grid the caller PINNED always wins: auto-recenter engages when the field is left NULL, when it is marked with auto_grid() (how a wrapper package declares an axis it defaulted rather than one the user chose), or when its nodes are exactly the engine's own default axis. Declines gracefully (keeps the fixed-grid fit) when another axis in the same grid has unguessable support (car_proper's rho_car); whichever way it declines, outer_grid_recenter_declined says which (see below).

On the list-of-blocks path an icar, bym2, car_proper or nngp block may carry

projector in place of spatial_idx: a list of n_arms matrices (dense or a **Matrix** sparse matrix), arm k's of size N_k x n_spatial_units (N_k x n_spatial for nngp), or one matrix shared by every arm; a NULL entry is an arm the field does not reach. Arm k's predictor then receives A_k z with z on the block's own prior, so restricted spatial regression is A_k = P_k S_k, with P_k = I - X_k (X_k' X_k)^-1 X_k' and S_k the observation-to-unit incidence. An areal field's level is placed from A_k 1: where it is zero on every arm the level never reaches the predictor and is removed with no fold; where it is the intercept column it folds into the intercept. The nngp and car_proper priors are proper and carry their level themselves. Restricting the field changes the estimand to the marginal association (see spatial_rsr()).

An empty prior = list() is a list of zero blocks: the model carries no latent field, each arm's predictor is its fixed effects, and the outer grid is spanned by the per-arm dispersion axes of phi_grid alone (one cell when there are none).

copy

Multi-block copy specification (multi-block prior only). For a single-block fit there is no copy argument: declare the copy coefficient on the arm via responses[[X]]$field_coef = list(name = "alpha", grid = G). On the multi-block path copy is an unnamed list of specs – list(list(arm, block, alpha_grid),...) – coupling N distinct shared latent fields, each onto its own arm with its own α axis, integrated over the product outer grid. Each spec must name a distinct block.

The α axis takes EITHER alpha_grid, which states its nodes, OR alpha_n (field_coef$n on the single-block path), which re-reads the engine's own axis at a higher RESOLUTION – more nodes between the same bounds, keeping the atom at 0 that gives the "no copy" base model posterior mass. They are different requests and supplying both is an error. Use alpha_n to integrate the same axis more accurately: it is the only outer axis a copy fit could not otherwise raise, since stating nodes for it also restates the prior structure it carries (gcol33/tulpa#633). The copy block may be any of icar / bym2 / car_proper / rw1 / rw2 / ar1 / iid; blocks with their own per-arm scaling (lf, hsgp_mo) or a precomputed precision (tgmrf) do not take a copy. A copy block's own sigma_grid (the donor field amplitude, same default ceiling as the single-block prior$sigma_grid above) auto-recenters on collapsed_edge the same way, one block per attempt.

phi_grid

Optional list specifying per-arm dispersion axes on the outer grid. Accepts either a named list (keys = arm names) or a positional list of length n_arms. Each entry is one of:

- NULL or scalar – no axis for that arm; the kernel uses the parse-time scalar responses[[k]]$phi.
- numeric vector of length > 1 – adds a new outer-grid axis phi_<arm> taking those values; the kernel rewrites arms[k].phi at each grid point before the inner Newton solve.

Family-specific interpretation of arm$phi (the parse-time scalar and the grid values):

- gaussian – residual VARIANCE, the one engine convention (the SD is sqrt(phi)). Use phi_grid to estimate it as a hyperparameter instead of pinning it pre-fit.

- lognormal – residual variance on the log scale; identical kernel parameterization as gaussian plus the -log(y) Jacobian.
- neg_binomial_2 – dispersion (variance is mu + mu^2/phi).
- beta – precision (variance is mu(1-mu)/(1+phi)).
- gamma, inverse_gaussian – shape / dispersion.
- binomial, poisson – ignored.

A dispersion axis is PLACED like a prior block's scale axis when the caller declares it a default with auto_grid() – phi_grid = list(pos = auto_grid(nodes)). The engine has no default dispersion axis of its own, so an unmarked vector is read as a pin and integrated exactly as written; the fit records that per axis in outer_grid_axis_declined. Marked, the axis is re-laid on mode +/- span * sd from the same mode/Hessian stencil the field-SD axes are placed from, which on a span wide enough to hold no node near the posterior is the difference between an estimate and an endpoint.

Each phi_<arm> axis is appended to the Cartesian product and varies slowest (within-spatial warm starts hold). The axis appears as a regular hyperparameter in theta_grid, theta_mean, and theta_sd, and participates in adaptive-grid refinement.

prior_sigma, prior_alpha

Optional regularizing hyperpriors on the donor field amplitude σ and on the copy coefficient α. Each is NULL (default: the axis carries what hyperprior gives it) or a list of the form list(family, params):

- list("pc.prec", c(U, alpha)) – Penalized Complexity prior, calibrated by P(theta > U) = alpha. Closed-form density lambda * exp(-lambda * theta) with lambda = -log(alpha)/U. Drop-in for the weakly-identified small-n_pos regime . Pick U at the upper end of plausible values so the prior shrinks the tail without biasing the modal cell when the data identifies it: default-friendly choice on σ is c(U = 1.0, alpha = 0.01) (donor amplitude); on the dimensionless copy coefficient α the recommended choice is c(U = 4.0, alpha = 0.01). What counts as strong is set by the measure the axis already carries, and on the copy scale that is not flat: with no prior_alpha it takes the engine's own exponential slab, whose rate puts 5 % of the prior above the largest declared node (control$copy_slab). A prior_alpha REPLACES that slab, so it shrinks harder than the default only once its own lambda = -log(alpha)/U beats the slab's – on a copy grid reaching alpha = 2, lambda = 1.50, the strength of U = 3.07. Too small a U over-shrinks the copy coefficient past the modal cell and, through the alpha * sigma copy axis, inflates the coupled donor amplitude sigma: on a fixture with truth $\alpha = 1$ and sigma = 0.6, c(U = 2.0, alpha = 0.01) pulls the α geometric mean to 0.93 and lifts sigma to 0.68, and c(U = 0.25, alpha = 0.01) to 0.51 and 0.94.
- list("half_normal", scale) – half-normal with scale scale > 0. Sharper tail decay than PC; use when stronger regularization is desired and the truth is well inside the prior. The contribution is added to log_marginal cell-by-cell at the kernel-call boundary, so refinement passes (adaptive grid, var-of-means consistency) see the regularized posterior. When the data identifies the parameter (e.g. n_pos >= ~200) the prior is essentially harmless – the

lever is tail-shrinkage at small n_pos. prior_alpha only applies when copy is active; prior_sigma applies on any sigma-named axis.

WHICH AXES ONE SPEC REACHES. A list-of-blocks prior can carry the same axis on several blocks at once – two copied fields carry b1.alpha and b2.alpha, and each copy block carries its own donor b<k>.sigma. One spec applies to EVERY block carrying the axis it names, each block taking its own independent copy of the density (its atom / continuum split is normalised against its own axis, so nothing is shared between blocks). To regularize the blocks differently, or only some of them, key the prior by block the way a multi-block copy spec does:

```
prior_alpha = list(
  list(block = 1, prior = list("pc.prec", c(4.0, 0.01))),
  list(block = 2, prior = list("half_normal", 2.0))
)
```

A block the list does not name carries no prior on that axis. An entry naming a block that carries no such axis is an error, and the message names the blocks that do. The per-block shape is a multi-block one: on a single-block prior, where the axis is unique, pass the spec itself. A fit reports what it applied in hyperprior_axes: one record per role with the scope the caller stated it at ("every_block" / "per_block"), the block indices reached, and the grid columns those are.

prior_sigma also interacts with the auto-recenter above: its second attempt engages the engine's own weakly-informative PC(U = 3, alpha = 0.01) prior, and a prior_sigma the caller PINNED suppresses that (the caller's prior stands). Pinning is decided by provenance, not presence: a spec marked with auto_grid(), or one equal by value to the engine's own default, is a default and does not suppress the escalation. When it does, the fit carries outer_grid_prior_declined = "prior_pinned", so a second attempt that changed only the grid geometry is legible rather than looking like the full escalation.

prior_phi

Optional regularizing hyperprior on the per-arm dispersion axes declared through phi_grid (e.g. a Beta precision on a cover arm, a negbin dispersion, a Gaussian residual SD). Same families as prior_sigma – NULL (default: the axis carries what hyperprior gives it), list("pc.prec", c(U, alpha)), or list("half_normal", scale). A single spec re-weights every phi_<arm> axis on the grid, the way prior_sigma re-weights any sigma-named axis; with no phi_grid it is a no-op. The PC scale is the dispersion's own units (a precision for beta, a size for neg_binomial_2), so pick U at the upper end of plausible values.

hyperprior

The prior an outer hyperparameter axis carries when the call states no density for it, "proper" (default) or "flat". "proper" folds a normalised density on the axis's integration coordinate into log_marginal: the PC prior P(sigma > 3) = 0.01 on a standard deviation, variance or precision axis, the PC range prior at 0.2 times the coordinates' bounding-box diagonal on a range or lengthscale axis, a uniform on a bounded axis, and the PC + LKJ prior over a free-covariance block. An axis with no sourced density is named in log_hyperprior_declined. "flat" folds no density of the engine's own, so such an axis is integrated under its cell measure alone, is named in log_hyperprior_declined as "flat_hyperprior",

and the fit's log_evidence declines with "improper_hyperprior". A density the call states – a prior_* argument, a block's rho_prior, prior_range or prior_sigma, the copy coefficient's slab, a tgmrf() block's own prior – applies under either choice. The same two choices, under the same names, are offered by tulpa(), tulpa_eb(), tulpa_re_cov_nested() and fit_spde().

cell_coupling Character scalar naming a per-cell coupled likelihood registered against tulpa's process-global registry (see src/cell_coupling_registry.h). Defaults to "separable", the arm- separable per-obs sum every existing joint fit uses. Consumer packages (e.g. tulpaObs) compile a tulpa::CellCouplingSpec subclass in their own src/ and register it from R_init_<pkg> via the tulpa_register_cell_coupling C callable; the R driver validates the name against the registry and the inner Newton routes the per-cell contribution through evaluate_cell() when the spec couples at least one arm.

control Optional list of perf/numerical tuning knobs (statistical arguments stay top-level), following the control convention of tulpa(). Recognised elements (defaults in parentheses):

- max_iter (50L), tol (1e-6) – inner Newton iteration budget and tolerance.
- n_threads (1L) – inner-loop OpenMP threads (per-observation scatter, compute_eta, log-likelihood reduction). For typical joint workloads (N in the hundreds to a few thousand) inner parallelism is overhead-dominated; prefer n_threads_outer, which stacks better on many-core hardware. Capped at the physical performance-core count by default (see n_threads_scatter), as the inner per-observation loops oversubscribe a hybrid CPU's efficiency cores past that point. A fit is reproducible bit for bit at a GIVEN n_threads: the per-observation sum cuts its range into that many contiguous chunks and adds the chunk sums in chunk order, so nothing about the answer is left to the OpenMP runtime. Chunking imposes its own association, so two different n_threads agree only to floating-point tolerance (measured at 6e-14 on log_marginal over four families).
- n_threads_scatter (performance-core count) – cap on the inner per-observation threads. Overrides the default performance-core cap on n_threads; raise it to use all logical cores or lower it to leave headroom. No effect where the core topology cannot be resolved (off Windows), where n_threads is used as requested.
- n_threads_outer (1L) – outer-grid OpenMP threads. When > 1, a pilot Laplace at the centre cell warm-starts the remaining cells, each dispatched across n_threads_outer threads with its own CHOLMOD solver and NewtonScratch (inner OpenMP auto-disabled). 1L is serial, chained warm-starts – bitwise identical to the pre-speedup driver. Recommended on multi-core workstations: parallel::detectCores() - 1L. A solve that runs alone holds every thread the fit was granted, the larger of n_threads and n_threads_outer under the n_threads_scatter cap, on its inner loop: the pilot, every cell of a refinement or var-of-means consistency round, and the skew probe. A fit is therefore reproducible bit for bit at a given pair of n_threads and n_threads_outer. It is a REQUEST: the driver clamps it to the team the OpenMP environment hands out (omp_get_max_threads(), OMP_THREAD_LIMIT, and the check farm's core cap), because the per-cell

block cache sizes its slot array from that same number. A job script exporting `OMP_NUM_THREADS=1` to pin the INNER threads therefore runs the outer grid serially whatever is asked for here; the fit says so once, in a message, and records the pair in `n_threads_outer_requested` / `n_threads_outer_realised`.

- `tile_warm` (`TRUE`) – when `n_threads_outer` > `1` and a copy block is present, group outer cells into tiles sharing every axis except the copy coefficient `alpha`, solve one warm Tier-2 per tile from the centre pilot, and warm-start the rest from their tile pilot. Falls back to the single-tier path when no copy block / single tile / `n_threads_outer` <= `1L`. `FALSE` recovers the pre-tiling behaviour (e.g. regression testing).
- `prune` (`TRUE`), `prune_tol` (`1e-3`) – cheap-pass screening. The driver sweeps the outer lattice running a short inner Newton per cell, each warm-started from the previous screened cell's quasi-mode (lattice-adjacent), computes a screening Laplace log-marginal, softmax-normalises, and skips the full inner Newton on cells whose normalised weight is < `prune_tol`. The neighbour-warm-start sweep keeps every cheap mode near its cell's true mode, so the cheap ranking is faithful to the full-solve ranking even when the inner latent mode moves substantially across the grid. Pruned cells get `log_marginal = -Inf`, `n_iter = 0`, and inherit the pilot mode; the pilot cell is never pruned, and at least `prune_min_keep` cells (5) are solved in full whatever the tolerance says – the highest-ranked dropped cells are restored up to that floor, because the outer grid placement pass reads a finite-difference curvature off the cells that were SOLVED and a kept set of one carries none. After the full pass the kept cells say how far off the screen was, and the posterior mass the dropped cells may carry is bounded by lifting each by the worst error measured on a kept cell (`prune_dropped_mass_bound`). Past 1% the driver solves the dropped cells that contribute most to the bound until it is under 1% (`prune_n_repaired`), so what the fit returns holds that bound. Two fallbacks to the full grid remain, each with a warning: a bound the repair could not bring under 1%, and a screened fit whose grid placement declined for want of curvature. On by default: a grid placed around its mode crosses axes the posterior does not fill (gcol33/tulpa#925), and solving every one of those cells is most of a large fit's cost. `FALSE` solves every cell, and is the default when the fit asks for local CCD refinement (`local_ccd`, or `k_refine = "ccd"`), whose stencil reads each refined cell's axis neighbours. Stacks with `n_threads_outer`. `prune_tol` must be in `[0, 1)`.
- `prune_log_gap` (unset) – the screening cut in nats: keep every cell within this many nats of the best screened cell. `prune_tol` is a normalised weight, so what it cuts at is the gap `-(log(prune_tol) + log(Z))`, and the whole range a caller would type is a few tens of nats – on an outer surface spanning thousands every setting returns the same kept set. It reaches the kernel as `prune_tol = exp(-gap)`, so the realised cut is the requested gap less `log(Z)`; the fit reports the realised value in `prune_log_gap_cut`. Setting both this and `prune_tol` is an error.
- `screen_iters` (`5L`) – inner Newton steps the cheap screen takes per cell, read only when `prune = TRUE`. The screen only has to RANK the cells, and each one is warm-started from its already-screened lattice neighbour, so its

quasi-mode is near the cell's own mode after very few steps. Every step is paid on every cell of the grid, including the cells the screen keeps and then solves in full, so a depth above what the ranking needs makes screening cost more than the solves it avoids. Must be a single integer `>= 1`.

- `fitted_var` (`TRUE`) – also fill `fitted_eta_var`, the per-cell, per-observation predictive variance of the linear predictor. It is the WITHIN-cell spread a grid-mixture replicate is drawn with (`posterior_predict()`); the per-cell predictor `fitted_eta` itself is stored either way. Computed by a per-cell solve sweep on top of the inner Newton, so a fit that will not be predicted from can decline it; the replicates of a fit run without it then carry the across-cell spread only. Read on a single-arm fit, which is the one that carries a single linear predictor to report.
- `x_init` (`NULL`) – warm-start for the first grid point's inner solve.
- `verbose` (`FALSE`) – when `TRUE`, announce the engaged outer integrator for a multi-block prior in one line at selection time (see `integration`), and report each CCD decline reason.
- `store_Q` (`FALSE`) – also return the per-grid joint precision Q (lower triangle, CSC) as `Q_csc_p_per_grid`, `Q_csc_i_per_grid`, `Q_csc_x_per_grid`, `Q_csc_n`, letting callers compute INLA-style total-variance posterior moments (`Var-of-means + Mean-of-Var`) on inner latent coordinates such as fixed-effect betas.
- `keep_grid_hessians` (`TRUE`) – retain the per-grid-point fixed-effect mode and marginal precision as `$grid_modes` / `$grid_hessians`, which is what lets `summary.tulpa_fit()`, `confint.tulpa_fit()` and `vcov.tulpa_fit()` report the grid-marginalized fixed-effect covariance instead of `NA`. Memory is `O(n_fixed^2)` per cell. When the retention is not available the reason is recorded on `$grid_fixed_declined`.
- `adaptive_grid` (`FALSE`), `adaptive_grid_edge_thresh` (`0.02`), `adaptive_grid_max_passes` (`1L`) – when `adaptive_grid = TRUE`, a mode-tracked 1D refinement pass triggers on any axis whose marginal boundary weight exceeds `adaptive_grid_edge_thresh`. New points are added to that axis as levels (interior densification + outward log-spaced extension), each laid in every row of the other axes that holds the posterior – at most `n_new_points * n_rows` kernel solves. The edge score is `max(marginal_weight_at_boundary, exp(max_log_marginal_at _boundary - max_log_mar` catching both boundary pile-up and integrand truncation; `0.02` is ~4 log units of decay. `adaptive_grid_max_passes` caps the passes (one usually suffices). Fixes posterior CI under-coverage when truth sits near a grid edge. Both the single-block and the multi-block driver run the passes on a tensor grid; a CCD design, the adaptive lattice and a locally CCD-refined grid keep the nodes they were laid on.
  How far a given axis may be moved depends on where its nodes came from. An axis whose nodes the CALLER wrote down – a block's `sigma_grid`, `copy$alpha_grid` / `field_coef$grid`, an entry of `phi_grid` – states where the fit integrates, so refinement densifies it and never places a node past either end node; a mode outside that range shows up as mass at the edge. An axis the ENGINE placed (`alpha_n` / `field_coef$n`, or no nodes given) carries no such statement, so refinement may follow the posterior out past its ends. Nodes a wrapper package computed as a default of its own are declared with `auto_grid()` and count as engine-placed.

- axis_refine (NULL) – per-axis override of the rule above, named by outer-grid axis ("sigma", "alpha", "phi_<arm>"; "b<k>.sigma", "b<k>.tau" and "b<k>.alpha" on a multi-block grid): "none" (the axis takes no refinement nodes), "densify" (nodes inside the declared span only) or "extend" (nodes past the end nodes too). A single unnamed value applies to every refinable axis. An unknown axis name, or a request for nodes on an axis this driver does not refine, is an error rather than a silent no-op. Read only when adaptive_grid = TRUE or var_of_means_consistency = TRUE.
- var_of_means_consistency (TRUE) – run a post-integration consistency pass over the refinable axes (a block's field SD, the copy scale and the per-arm dispersions). An axis whose marginal has collapsed onto too few nodes to carry a spread, such as a field SD narrower than its placed cell, has levels added in every row of the other axes that holds the posterior until its marginal resolves, so the reported interval is read off the posterior and not off a cell box. Attaches var_of_means_consistency_info.
- inner_factorization ("auto") – which factorization the dense inner Newton applies to the Hessian it assembled: "auto" by the latent dimension, "sparse" for CHOLMOD, "dense" for the dense Cholesky. Independent of force_sparse, which selects the assembly path.
- force_sparse (FALSE) – linear-algebra backend for the inner joint solve. TRUE / FALSE select the sparse or dense path outright, regardless of the dense/sparse heuristic. "auto" selects by the latent dimension the fit will actually build, taking the sparse path above n_x > 1000 and the dense one at or below it, where the sparse symbolic-analysis and indirection overhead outweighs the fill-in saving. A model whose latent dimension cannot be determined resolves to dense.
- integration ("auto") – outer-grid node layout for a multi-block prior. A central composite design (CCD) integrates the hyperparameter posterior on 1 + 2d + 2^d nodes oriented by the Cholesky of the posterior covariance at the joint hyperparameter mode – far fewer inner solves than the d-dimensional tensor product (25 vs 81 at d = 4). "auto" (the default) uses the CCD only at >= 4 transformable axes, where the tensor product's k^d blow-up bites hardest, and keeps the cheaper, more ridge-robust tensor grid at <= 3 axes; "ccd" lowers the CCD threshold to >= 3 axes; "grid" always forces the full tensor product. "grid_adaptive" is the low-dimensional companion to the CCD: it seeds a coarse subsample of the SAME tensor lattice (latent block axes and phi axes together), floods outward from the posterior mode on the fine lattice, and evaluates only the cells within a log-density cutoff of the peak – a strict, uniform-weight subset of the dense tensor, so its posterior matches the dense grid to that cutoff at fewer inner solves when the hyperparameter posterior concentrates (a sharply-identified field SD / precision). It declines back to the dense tensor on a diffuse posterior (kept region would rival the tensor) or a degenerate lattice, so it never costs accuracy; tune it with adaptive_grid_cutoff / adaptive_grid_stride / adaptive_grid_max_frac. The CCD autofalls back to the tensor grid for an axis whose support is not safely transformable (a CAR_proper rho_car or a non-BYM2 rho), or a flat / ridged / degenerate outer mode-find or Hessian; an active phi_grid rides as a ten-

sor axis crossed on top of the CCD. Single-block joint priors always use the tensor grid. The CCD mode-find runs cheap warm-started inner solves and does not write to the checkpoint file. Under `verbose = TRUE` the engaged integrator is announced in one line at selection time (e.g. `outer integration: CCD (4 latent axes` or `tensor grid (72 cells)`, or `CCD declined -> tensor grid`), so the auto switch to the CCD at `>= 4` axes is never silent. The resolved integrator is also returned on the joint result as `$integration`, alongside `$integration_requested` and `$integration_declined` (see Value), so a fallback is on the fit itself and not only in a verbose message.

- `ccd_budget` (`1`) – ceiling on what PLACING a CCD may spend, as a multiple of the tensor grid the design replaces. Every mode-find probe is one full inner Newton solve, the same solve an outer grid cell costs, and a placement round is `1 + 2d + 4 C(d, 2)` of them (33 at `d = 4`), so at the round caps alone a placement can spend far more than the integration it avoids. At `1` a placement plus its design costs no more than that tensor grid; the fit declines to the tensor grid with `integration_declined = "placement_budget"` when the ceiling is reached, and reports what it spent as `ccd_modefind_evals` (see Value). `Inf` places without a ceiling. `ccd_budget_floor` (`TRUE`) exempts the designed first attempt – the eight rounds from the grid-median seed, whose cost is set by the axis count and not by the grid – so the ceiling bounds the escalation (grid seed, step calibration, the 30-round rescue mode-find) rather than the path the CCD was built for; `FALSE` makes `ccd_budget` a hard cap on the whole placement.
- `local_ccd` (`NULL`) – local CCD refinement of a multi-block tensor grid. `TRUE` (defaults) or a `list(max_cells =, f0 =, skew_max =, rank =)` refines a few high-weight, mutually non-adjacent interior cells, replacing each with a small curvature-aware CCD node cloud so a coarse base grid resolves the sharply-peaked directions without the `k^d` tensor blow-up. The local curvature is a diagonal finite difference of the outer log-marginal over the cell's own grid neighbours (no mode-find; only the off-centre nodes are new solves), warm-started from the cell's inner mode; each refined cell's sub-nodes carry partition-of-unity design weights so the total integration weight is conserved (no double-count). `max_cells` (`8L`) caps the refined cells, chosen in order of `rank`: `"weight"` (default), the cell's integration weight, or `"mass_moved"`, the weight times `|exp(log_box_ratio) - 1|` the box-mass rule predicts the cell's midpoint atom misplaces; `f0` (`1.1`) is the CCD radius. The design scale is shrunk per cell so the cloud fits the cell's Voronoi box (the local-Gaussian mass beyond it belongs to the neighbouring cells). A cell keeps its cloud only while the nodes' own log-marginals stay within `skew_max` (the `gamma3_ok` band, `0.5`) of the quadratic the cloud was placed from, measured as a standardized cubic magnitude; above it the cell is put back as its own mass atom, which on a skewed outer target is measurably closer than the design. Engages only on the tensor path (the curvature stencil needs axis neighbours), at `>= 4` transformable latent axes, with no active `phi_grid`; otherwise it is a no-op. The applied refinement is summarised on the result as `$local_ccd_info`. Also driven automatically by `k_refine = "ccd"`.
- `adaptive_grid_cutoff` (`10`), `adaptive_grid_stride` (`2L`), `adaptive_grid_max_frac`

(0.75), adaptive_grid_min_cells (48) – tuning for integration = "grid_adaptive". adaptive_grid_cutoff is the log-density keep / expand radius from the peak (larger keeps more cells, closer to the dense tensor); adaptive_grid_stride the coarse-seed subsample stride per axis; adaptive_grid_max_frac the kept-fraction ceiling past which the builder declines back to the dense tensor; adaptive_grid_min_cells the smallest dense tensor worth the adaptive machinery – below it the builder declines BEFORE any inner solve (on a small tensor the coarse seed is already most of the grid, so there is no tail to skip). Ignored by the other integrators. The kept-cell / dense / solve counts are returned as $adaptive_grid_info.

- hessian ("auto") – the curvature each inner Newton step is built from, where a cell-coupled arm makes the joint Hessian indefinite away from the mode. "auto" takes the observed Hessian wherever it factors as it stands, the Newton step, and the coupled cells' complete-data expected information (Fisher scoring, positive semi-definite) where it does not; "lm" always takes the observed Hessian, loading its diagonal until it factors; "fisher" always takes the expected information, which converges linearly; "psd" eigen-clamps the observed Hessian (small n_x only). The final mode-pass behind log_det, the log marginal and the SEs uses the observed Hessian under every choice, and a fit with no coupled arm runs "auto" and "fisher" as "lm".
- inner_refresh (1L) – inner-Newton Cholesky factor reuse interval (Shamanskii / chord method). For a non-quadratic positive arm (e.g. a beta cover arm) the latent Hessian changes every inner iteration, so the default re-factorizes the sparse Cholesky on each step – the dominant per-grid-cell cost. inner_refresh = m > 1 re-factorizes only every m-th inner step and reuses the cached factor in between (refreshing early whenever a reused solve fails). The gradient is exact on every step and each step is line-search safeguarded, so the converged mode is unchanged and the final mode-pass Hessian (log_det, SEs) is always fresh; only the path to the mode uses a stale curvature, which may cost a few extra inner iterations. Applies to the sparse joint path with an LM-guarded step (hessian "auto", "lm" or "fisher"); the dense small-n_x path re-factorizes a cheap Hessian and ignores it. 2L-4L is a good range for a slow beta arm.
- k_quality ("report") – the reliability intent for the outer Pareto-$\hat{k}$, a single statement of how reliable the fit should be. "report" (default) computes the diagnostic and reports the achieved band. "ok" / "good" additionally name a TARGET band (the $\hat{k}$ confidently usable, resp. good) and raise the default k_samples (to 800L / 2000L, unless you set it) so the bootstrap CI can resolve it. "none" disables the diagnostic. The fit carries an honest verdict – k_quality_requested, k_quality_reached, k_quality_best, k_quality_miss, k_quality_reason, k_quality_rounds, k_quality_k_trace – and never silently downgrades: if the requested band is not confidently met it reports the band actually reached and why. For "ok" / "good", when the first fit does not reach the band the engine escalates by REFINING THE INTEGRATION GRID (see k_refine), widening / densifying it where the posterior mass escapes its current bounds and re-diagnosing, up to k_max_rounds rounds. Refinement is the first lever on either kind of

miss because it lowers $\hat{k}$ itself rather than the uncertainty around it. Once refinement is exhausted – no boundary mass left to chase – what remains depends on the miss (`k_quality_miss`): a `"resolution"` miss ($\hat{k}$ confidently outside the band) ends the chase, while a `"precision"` miss (the bootstrap CI straddling a boundary, so $\hat{k}$ may already be inside the band) spends its remaining rounds doubling the importance-draw budget, the one lever refinement does not touch.

- `k_refine` (`"grid"`) – the integration-refinement rung for `k_quality` `"ok"` / `"good"`. `"grid"` (default) re-fits with adaptive grid refinement (`adaptive_grid`) each escalation round, driven by the bad $\hat{k}$, so a too-coarse / too-narrow grid is widened / densified where the importance weight concentrates until the band is reached or the budget is spent. `"ccd"` instead refines a few high-weight, mutually non-adjacent interior cells with local curvature-aware CCD node clouds (see `local_ccd`), the right rung when the grid is too coarse to resolve a sharply-peaked direction rather than too narrow; it forces a tensor base grid (the curvature stencil needs axis neighbours) and engages only on the multi-block path at >= 4 transformable latent axes. `"none"` disables refinement: the band is reported but not chased.
- `k_max_rounds` (`2L`) – the escalation round budget for `k_quality` `"ok"` / `"good"`: the maximum number of re-fit rounds after the first fit. A refinement round allows one more pass than the last refinement round did; a round taken after refinement is exhausted leaves the grid where it stands and doubles the importance-draw budget instead. `0L` disables escalation (single-shot, the band is reported but not chased).
- `diagnose_k` (`TRUE`), `k_samples` (`500L`) – compute the outer Pareto-$\hat{k}$ accuracy diagnostic by importance-sampling the joint hyperparameter posterior against the proposal the integrator fits (mixed per-axis transforms: `log` for positive scales, logit for the BYM2 mixing weight, identity for the copy coefficient α). `k_samples` is the number of importance draws, each one an extra inner joint solve, and is where more tail information comes from: a tighter k-hat needs MORE actual tail ratios, so increase `k_samples` (not `k_bootstrap`). It is not a pure precision knob, though. Under the automatic PSIS tail rule (`min(S/5, 3 sqrt(S))`) the fitted tail FRACTION shrinks as `3 / sqrt(S)`, so a larger budget describes a DEEPER quantile of the weight distribution and the reported k-hat can move materially – measured on a synthetic heavy-tailed outer target, 0.57 at 500 draws and 7.94 at 50000 (gcol33/tulpa#631). Set `k_tail_points` in proportion to `k_samples` to hold the fraction and get a strictly sharper estimate of the same number; the `k_quality` escalation's draw rung does exactly that. The draws are RNG-restored so the fit's modes / draws are unchanged. A fit carrying an axis whose support is not safely known (CAR_proper's `rho_car`) declines to the quadrature-ESS fallback (`pareto_k = NA`). `FALSE` skips the diagnostic. `"by_arm"` additionally computes a k-hat restricted to each arm's hyperparameter axes (the other arms held at their posterior mean), reported in `pareto_k_by_arm`, to localise which arm drives a tail-heavy joint k; the joint k itself is unchanged. Per-arm k is defined for the multi-block layout with two or more arms and declines for the single-block shared-field layout.
- `k_threads` (`NULL`) – outer-thread width for the diagnostic's importance

batch. The `k_samples` re-solves are independent and run after the grid (every core free), each solved single-threaded once the batch saturates the pool, so widening it is a bit-identical wall-clock speedup (the k-hat is unchanged). `NULL` follows the fit's own thread grant – the larger of `n_threads_outer` and the inner `n_threads` – so a serial fit keeps a serial diagnostic while a threaded fit gets a free parallel one. `"auto"` uses the physical performance-core count (capped at 2 under R CMD check); an integer pins the width (`1L` forces serial). Always capped at `k_samples`.

- `k_bootstrap` (`1000L`) – bootstrap replicates for the outer Pareto-$\hat{k}$ uncertainty. The k-hat is a single fixed number for a fit + proposal; its sampling uncertainty GIVEN the proposal is estimated by resampling the diagnostic's raw importance log-ratios with replacement and re-fitting the GPD tail `k_bootstrap` times (no new inner solves). Reports `pareto_k_se_boot` (bootstrap SE), `pareto_k_ci_low` / `pareto_k_ci_high` (the 2.5\ closed-form GPD-shape MLE asymptotic SE $(1+k)/\sqrt{M}$, a cross-check), and `pareto_k_band_confident` (TRUE iff the bootstrap CI lies within one reliability band). The bootstrap measures how UNSTABLE the current tail estimate is; it cannot create tail information. Increase `k_samples`, not `k_bootstrap`, to obtain more tail information. `0L` skips it (point k-hat only). The per-arm k carries the same fields.
- `k_tail_points` (`NULL`) – number of upper-tail order statistics for the GPD fit. `NULL` uses the automatic PSIS rule $\lceil \min(0.2N, 3\sqrt{N}) \rceil$. An explicit value is an EXPERT tail-threshold control, capped at the 20\ so the fit stays an extreme tail; it is NOT a precision knob (a request that drags body ratios into the tail lowers variance but biases the k-hat). The used and requested counts are reported in `pareto_k_tail_points` / `pareto_k_tail_points_requested`.
- `k_conf_bands` (`NULL`) – the reliability-band boundaries the bootstrap CI is tested against for `pareto_k_band_confident`. `NULL` (default) uses the sample-size-dependent boundaries $c(0.5, \min(1 - 1/\log_{10} S, 0.7))$ at the realised draw count S (Vehtari et al. 2024): the good cut is 0.5 and the usable cut tightens below 0.7 for small S (about 0.565 at $S = 200$, reaching 0.7 only past $S \approx 2154$). Supply a strictly-increasing numeric vector to fix the boundaries instead, e.g. `c(0.5, 0.7)` for the size-independent good / ok / unreliable split.
- `diagnose_skew` (`TRUE`), `skew_idx` (`NULL`) – compute the inner-Laplace skewness diagnostic (`$inner_skew`, gamma_3, Rue Martino & Chopin 2009 Sec 3.2.3) at the fitted MAP grid cell: one extra Newton solve via the same `kernel_fn` the outer diagnostic reuses, scoring every arm's fixed-effects coefficients by default (pass `skew_idx`, 1-based indices in the joint `[arm1_beta | arm1_re | arm2_beta | ... | blocks]` latent layout – see `$arm_layout` – to probe additional indices). This is the complementary layer to `diagnose_k`: that scores the outer hyperparameter-grid integration around a FIXED inner Laplace, this scores whether that inner Gaussian approximation is itself a good fit. A genuinely coupled arm (`cell_coupling != "separable"` on that arm, e.g. tulpaObs's `occu_cover`) has no per-obs likelihood for the separable formula to score; it is scored instead by the contraction of the cell third-derivative tensor, which differences the cross-arm Hessian the coupling spec already returns. A fit that can carry neither re-

ports NaN, not a silently wrong 0, and $inner_skew_declined says WHY – "coupled_arm" marks arms the inner layer could not score at all, distinct from a diagnostic that was simply switched off – while $inner_skew_arms_declined names the arms with no oracle, including on a partially scored fit. See diagnostics() for the combined verdict. Wired for both the single-block backends (icar/bym2/car_proper) and the multi-block path (a per-group RE, a trend field, or an arm-specific field block).

- within_cell ("log_quadratic") – the WITHIN-CELL construction the reported per-axis hyperparameter intervals are read with. The outer grid's weights say how much mass each cell holds; they do not say how it is spread inside the cell, and a quantile needs both. "log_quadratic" joins each row's node log densities by overlapping quadratics, exact for a Gaussian at any spacing, and continues the end quadratic past the outer node instead of stopping at the box edge; it runs on the rows that resolve their own conditional (h / sd <= 2) and reads every other row as boxes, declining to "box_uniform" where no row qualifies. "box_uniform" puts the cumulative FULL mass at each cell EDGE and interpolates between edges; "chord" puts the cumulative MID-mass at each cell coordinate and interpolates between coordinates. The default moved from "box_uniform" at gcol33/tulpa#932: a placed axis sits at 1.25 posterior SDs per cell, where the box read is narrow by construction, and against dense references of a two-arm ICAR fixture the log-quadratic read's summed error is 0.038 against the box read's 0.110 on the same grids. outer_grid_h_over_sd is how wide a cell is on each axis, and theta_within_cell is what each axis was actually read with. Only a "density" support admits the box and log-quadratic reads – a CCD design, a locally refined grid and a posterior sample are not cell partitions that tile – and an axis they decline on reports the read that ran with a reason rather than erroring. Nothing else moves: point estimates, moments, draws and weights are untouched.
- skew_correct (TRUE) – consume the inner-Laplace expansion instead of only grading it: report Cornish-Fisher marginal quantiles at each coefficient's own gamma_3, about the centre gamma_1 + gamma_3 / 2, from summary() / confint() wherever the combined inner band (gamma_3 and the inner importance k-hat) says the leading-order expansion is in its regime, and the Gaussian quantiles everywhere else. It is post-processing on the reported quantiles: draws, modes and weights are untouched, so a fit run with it off is bit for bit the fit it was before. The band that bounded the relocation itself (centre_unreliable) is off (Inf): every finite cutoff was measured to decline the coefficients the correction helps most. $skew_correction records the per-coefficient gamma_3, gamma_1 and the centre they form, the band, the k-hat, the combined band, the eligibility and the reason behind it; the skew_applied attribute on summary() / confint() records what was actually used at the requested level. A FULLY COUPLED fit declines it: the location term's contraction against a covariance block is not reachable from the cell third-derivative oracle, and an absent gamma_1 is never read as zero. Such a fit therefore reports what it reported before the default moved, to the bit – a declined coefficient keeps the grid-mixture read. See tulpa_nested_laplace() for the measurement behind the default.

- subspace_debias (FALSE) – correct only the latent directions the inner-layer diagnostics flagged, by exact Metropolis along the Gaussian-conditional-mean surface through each cell's mode, and leave the rest at their Gaussian conditional. Settings and semantics are the ones documented on tulpa_nested_laplace(). On a fully coupled fit gamma_3 is NaN for every arm, so the selector rests on the derivative-free inner Pareto-k-hat; where that also bands the coordinate reliable, idx pins the set explicitly. An EMPTY selection leaves the fit bit-for-bit identical to the plain path. A fit whose inner solve took the s2z rank-1 or the PSD eigen-clamp path carries no usable factor to build the surface from and is left uncorrected, the same two paths diagnose_skew declines on.
- cila (FALSE) – corrected integrated Laplace, the second inner-layer debias (after Lai, Margossian and Sheldon, arXiv:2605.20345). Where subspace_debias selects coordinates and runs exact Metropolis on them, this selects nothing: at every outer cell it draws n_points points from the whole inner Gaussian, weights each by the exact joint density it came from, and reports the weighted particles. The cell marginals and the latent posterior both converge to the exact ones as the effort grows, so n_points is the only dial. TRUE takes the defaults; a list overrides n_points (1024L), variant ("qmc", a Sobol net; "is" for iid draws, "rqmc" for the net under n_shift random shifts), n_shift (8L), n_draws (the reported draw count) and seed (the auxiliary stream, engine-owned so requesting the correction leaves every other posterior draw unchanged). Below 512 points a cell's particle set is too coarse to be a marginal at all and the request is refused. The correction reports DRAWS, so every coefficient-facing method reads them instead of the grid's Gaussian mixture, and the corrected per-cell masses become the fit's own weights / log_marginal with weights_source reporting "cila"; the pre-correction pair is kept as $cila$laplace and $cila$retained_mass is the original share of the cells that produced a usable particle set. $cila also carries the variant actually run and the PSIS grade (pareto_k, rel_ess) of the correction's own weights. A cell whose inner solve factorized SPARSELY draws through the CHOLMOD factor's own triangular and permutation solves; an LDL' factor carries no square root to draw with and is declined with "sparse_factor_not_ll".
- auto_recenter (TRUE) – re-centre a default outer grid axis on its posterior mode and refit when the fit rails against a boundary node. FALSE integrates over the grid exactly as given, whatever it is, and records outer_grid_recenter_declined = "auto_recenter_disabled". The joint FIELD rescues trigger on the whole grid's collapsed-edge regime rather than on a per-axis rail, so the per-axis policy names tulpa_nested_laplace() takes ("rail", "resolve", "always") are refused here with an error rather than accepted and ignored. A per-arm dispersion axis (phi_grid) is the exception and fires on its own sizing: it is crossed onto the tensor independently of the field's geometry, so whether the field's grid collapsed says nothing about whether the dispersion axis brackets its own posterior. The copy scale alpha fires the same way when its grid is the engine's own or a wrapper's auto_grid() default: its continuum is laid at the mode and its "no coupling" atom at 0 kept, and an axis whose posterior sits on that atom is left where it is. Every axis that

fires is laid from the ONE outer mode the placement mode-find reached, and the grid is refit once with all of them (gcol33/tulpa#925). `FALSE` holds these axes too.

- `recenter_pilot` (`FALSE`) – detect the placement above on a THINNED grid rather than on the full one. Placement reads two things, the argmax cell and an FD curvature stencil at it, and reads both off `log_marginal`; neither is read off the integration the detecting pass paid for, so every inner Newton solve of that pass is discarded the moment a placement fires. With the pilot on, the detecting grid keeps each axis's endpoints and at most `TRUE`'s three nodes between them (an integer names its own resolution), and the full grid is solved once, at the placed axes.
  Off by default because the trade is a property of the WORKLOAD. With `F` full cells, `P` pilot cells and a firing rate `p`, the un-piloted cost is `F + p (S + F)` against the pilot's `P + p S + F`, so the pilot pays exactly when `P < p F`. Measured on a 3-axis (sigma, alpha, phi) donor + copy fixture over 48 paired seeds: 0.78x the cells solved on the configuration that places on 10 of 12 seeds, and 1.29x on the one that never places.
  A coarse detector is not the same detector. The collapse trigger is `ess_grid < 2`, which falls with the cell count, so a pilot fires somewhat more readily than the full grid: on that same sweep the two disagreed on 9 of 48 pairs, every one of them the pilot placing where the full grid did not, and 7 of the 9 reached a HIGHER maximum inner log-marginal after placing (+0.35 to +2.14 nats) while 2 lost 0.46 and 0.72. It cannot go the other way: a declined pilot is followed by the full fit, which then gets its own detection, so the pilot only ever adds a placement. A fit that declines both is bit-identical to the same fit with the knob off. What the pilot detected on is recorded in `outer_grid_pilot`.

- `max_grid_cells` (`NULL`, no ceiling) – a ceiling on a multi-block tensor outer grid's cell count, refused before any inner solve with the axes that produced the count. The count is the number of inner solves, so it includes an active `phi_grid`: 2500 latent cells crossed with 4 dispersion values are 10000 cells. It bounds the dense tensor, including the one a declined `"grid_adaptive"` falls back to. Unset, a tensor of any size is solved, and one past 2048 cells is announced with a warning first.

- `checkpoint` (`NULL`) – grid-cell checkpoint/resume. Set `list(path = "fit.ckpt", resume = TRUE)` to make a killed or interrupted fit resumable: each completed outer-grid cell is appended to `path`, and a later call with the same responses + grid + control loads the finished cells and solves only the rest. EVA-scale joint fits run for hours, so a wrapper teardown, reboot, or OOM near the end otherwise loses the whole run. `resume = TRUE` (the default when a `path` is given) continues from an existing file; `resume = FALSE` removes it first and starts over. Adaptive-grid refinement cells are checkpointed under their own coordinate keys, so resume covers them too. A file written for different data or solver settings is rejected (fingerprint mismatch) rather than resumed onto a stale result.

Value

A list of class `c("tulpa_nested_laplace_joint", "tulpa_nested_laplace", "list")` with:

- `theta_grid`, `theta_names` – outer-grid hyperparameter values (includes the `alpha` axis when `copy` is set).
- `log_marginal`, `weights` – per-grid-point log-marginal and integration weights (sum to 1).
- `theta_mean`, `theta_sd` – posterior moments per hyperparameter, including `alpha` when `copy` is set.
- `theta_median`, `theta_ci_lo`, `theta_ci_hi` – weighted-quantile median and 2.5/97.5 empirical CI per hyperparameter axis (same names as `theta_mean`). Recommended summary for right-skewed scale-like axes (alpha at small n_pos, sigma/range/phi near a boundary), where the posterior mean is pulled by the right tail away from the bulk and `mean +/- 1.96 sd` mis-states the uncertainty.
- `theta_sd_source` – per axis, what the reported SD was read off: `"weighted"` (the spread of the weights, on a resolved axis), `"stencil"` (a parabola at the modal node) or `"within_cell"` (the modal node's mass spread over its cell, the read the interval of an unresolved axis comes from). `theta_sd_stencil_declined` says why a parabola was not formed.
- `modes` – `[n_grid x n_x]` matrix of inner modes.
- `fitted_eta` – `[n_grid x N]` matrix of the linear predictor at each cell's own mode, offset and latent field included. Present on a single-arm fit, which is the one that has a single linear predictor to report; a cell that was pruned or whose block preparation failed reads `NA`. `fitted_eta_var` is its per-cell within-cell variance, present under `control$fitted_var` (the default). Together these are the per-cell Gaussian `posterior_predict()` draws a replicate from.
- `n_iter` – inner Newton iterations per grid point.
- `arm_layout` – list with per-arm `beta_start`, `re_start`, spatial offset(s) and `n_x` for decoding modes.
- `prior`, `responses`, `copy` – echoed inputs.
- `timing` – named numeric of wall-clock seconds: `total` plus the `setup` (validation / encoding / grid construction), `grid` (inner Laplace solves, including adaptive-refinement and consistency passes), `postproc` (weight / moment / marginal assembly) and `diagnostics` (outer Pareto-$\hat{k}$) phases. The `grid` phase is the one that scales with grid size and core count. Surfaced one-line in `print`.
- `pareto_k`, `pareto_k_is_ess`, `pareto_k_scope` – outer Pareto-$\hat{k}$ accuracy diagnostic and its importance-sampling ESS (both `NA` when `control$diagnose_k = FALSE` or the fit declines; see the `diagnose_k` control knob). `pareto_k < 0.7` indicates the nested integration is reliable; `>= 0.7` that the (skewed / heavy- tailed) hyperparameter posterior is misfit by the Gaussian grid and the fit should escalate to an exact debias.
- `pareto_k_declined` – when `pareto_k` is `NA`, WHY: `"not_requested"` (`control$diagnose_k = FALSE`; nothing is wrong), `"unguessable_axis: <axis>"` (a support the engine will not guess, e.g. car_proper's `rho_car` – a PERMANENT limitation of that family, so read the quadrature ESS instead), `"draws_too_few"`, `"grid_too_small"`, `"no_varying_axis"`, `"degenerate_proposal"` (the outer mode curvature came back non-finite – a signal about the fit), `"not_applicable"`, or `"internal_inconsistency"` (an engine bug worth reporting). `NA` on a fit whose $\hat{k}$ WAS computed.

- pareto_k_proposal_source – how the outer importance proposal the $\hat{k}$ scores was built: "mode_hessian" from the Laplace curvature at the hyperparameter mode (the CCD design's, or a finite-difference Hessian when a sharp posterior collapses the grid), "grid_moment" from the grid-weighted covariance, "moment_matched" when the moment-matching refinement improved on either, "grid_mixture" when the faithful mixture-over-grid-cells proposal won, or "skew_normal" when the skew-normal rescue did. NA when the diagnostic is off or declines. The mode-Hessian source keeps the $\hat{k}$ meaningful when the grid concentrates on ~1 cell.
- pareto_k_regime – whether the outer grid actually integrated hyperparameter uncertainty: "spread", or "collapsed_interior" / "collapsed_edge" when the quadrature effective sample size falls below two cells so the integration has degenerated to a point evaluation at the modal hyperparameter. An interior collapse is benign (the grid bracketed the mode; the fit is empirical Bayes at that mode and only the integrated hyperparameter uncertainty is missing); an edge collapse means the mode sits at an extreme node, so the grid may be too narrow. Attached from the stored weights, so it is present even with control$diagnose_k = FALSE.
- pareto_k_grid_edge_axes, pareto_k_grid_edge_sides – for an edge collapse, which axes the dominant cell sits against and on which side ("lower" / "upper"); widen those axes and refit to confirm the mode is bracketed. Axes the grid pins to one value are excluded (pinned, not at a boundary).
- axis_span – per outer axis, the span the fit worked over and the two terms that separate it from the nodes it was given: nodes (the range of the axis's initial continuum nodes), declared (their support, i.e. those nodes plus the half node step the outermost cells own – for k equally spaced nodes that is k / (k - 1) times the node range on the axis's integration coordinate, so 2x at two nodes and 1.125x at nine), integrated (the region the final grid's cell measure integrates, the same interval axis_support reports), refine (the mode refinement ran under: "none" / "densify" / "extend") and n_nodes (initial and final continuum node counts).
- outer_grid_placement – "fixed" (the default sigma_grid axis was used as-is) or "auto_recentered" when a railed sigma axis triggered the mode-Hessian recenter-and-refit (see the prior argument above). outer_grid_recenter_attempts (integer) and outer_grid_prior_added (logical: whether the light default PC(U=3, alpha=0.01) sigma prior was applied) are set alongside it, plus outer_grid_prior_declined = "prior_pinned" when a second attempt ran but the caller's pinned prior_sigma held that prior back.
- outer_grid_recenter_declined – on a "fixed" placement, why the recenter did not run: "grid_not_collapsed" (the grid already brackets the mode – the common case, no refit needed), "axis_pinned" (the caller pinned sigma_grid; mark it with auto_grid() if it is a default rather than a choice), "default_axis_pinned" (a wrapper package declared the nodes with auto_grid(place = FALSE) and asked for them as written – nothing you pinned), or "no_usable_curvature" (the outer mode-find the recenter lays the axis from did not reach a mode with a usable curvature, e.g. a car_proper grid whose rho_car axis has unguessable support). Absent when the fit WAS recentred. A joint fit's field SD and its per-arm dispersion are placed by different passes over the same grid, and this slot reduces over them rather than holding the last to speak: an axis held because it was declared does not stand as the fit's answer while a pass with a movable axis has one.
- outer_grid_axis_declined – the same question PER AXIS, as a named character vector. The slot above holds one reason for the whole fit and is written only while the fit is unplaced, so on a fit where one axis moved and another did not it says auto_recentered and nothing about the axis that stayed – which is the axis grid_coarsest_axis then names. Written by the field-SD passes for sigma (b<k>.sigma on a copy block) and by the per-arm dispersion pass

for phi_<arm>, and carried across a later pass placing a different axis; absent on a fit no pass declined an axis of. An axis a pass DID move that still rails reads "attempts_exhausted" (or the reason its last attempt could not be laid), and is the axis the railed-mode warning names.

- outer_grid_pilot – present only when control$recenter_pilot ran: the pilot's resolution (n_pilot), its cell count (cells), the axes it thinned (axes) and those it could not (axes_kept), and what the placement was DECIDED from – the pilot's own regime, ess_grid and edge_axes. The reported grid is never the grid the trigger was read on, so a two-pass fit is legible from the object rather than only from the progress log. outer_grid_pilot_declined = "pilot_placement_declined" records a pilot whose detection placed nothing and which was therefore followed by the full fit and the full fit's own detection.
- outer_grid_recenter_sd_clamp, outer_grid_recenter_sd_raw, outer_grid_recenter_sd_used – on a recentred fit, one entry per moved axis: which mode-SD bound the placement hit ("none" / "floor" / "ceiling"), the SD the finite-difference stencil MEASURED, and the SD the axis was actually laid from. A clamped axis was laid from a substituted spread rather than a measured one, and the reported interval is read off that span; the two used to be indistinguishable on the fit.
- pareto_k_outer_skew – per-axis skewness of the hyperparameter marginal in the proposal's whitened coordinate, present only when a $\hat{k}$ above the good band triggered the skew-normal rescue pass. It is the EXPLANATION for an inflated $\hat{k}$: a symmetric Gaussian proposal against a right-skewed variance-component marginal has a heavy importance-ratio tail whatever the integration's quality. NULL when the Gaussian proposal already fit.
- pareto_k_se_boot, pareto_k_ci_low, pareto_k_ci_high, pareto_k_se_formula, pareto_k_tail_points, pareto_k_tail_points_requested, pareto_k_band_confident – the outer Pareto-$\hat{k}$ uncertainty, present whenever the diagnostic ran. pareto_k_se_boot is the bootstrap SE of the k-hat and pareto_k_ci_low / pareto_k_ci_high its 2.5\ quantiles – the estimator's sampling spread GIVEN the proposal, not a posterior credible interval; pareto_k_se_formula is the closed-form GPD-shape MLE asymptotic SE cross-check. pareto_k_tail_points is the GPD tail size used and pareto_k_tail_points_requested the k_tail_points request (NA when automatic). pareto_k_band_confident is TRUE iff the bootstrap CI lies within one reliability band, NA when the bootstrap was off or could not fit. diagnose_draws and diagnose_cost_ratio (the diagnostic's draw budget and its wall-clock cost relative to the fit) are attached at the top level.
- pareto_k_by_arm, pareto_k_by_arm_is_ess, pareto_k_by_arm_scope – present only with control$diagnose_k = "by_arm". Named (by arm) outer Pareto-$\hat{k}$ restricted to each arm's hyperparameter axes, the other arms held at their posterior mean, so a tail-heavy joint k can be localised to one arm. A per-arm entry is NA when that arm carries no varying axis. Each arm carries its own bootstrap uncertainty: pareto_k_by_arm_se_boot, pareto_k_by_arm_ci_low / pareto_k_by_arm_ci_high, pareto_k_by_arm_se_formula, pareto_k_by_arm_tail_points and pareto_k_by_arm_band_confident.
- k_quality_requested, k_quality_reached, k_quality_best, k_quality_miss, k_quality_reason, k_quality_rounds, k_quality_k_trace – the reliability verdict for the control$k_quality intent. k_quality_requested echoes the intent; k_quality_best is the band THIS fit reached ("good" / "ok" / "unreliable", or "uncertain" when the bootstrap CI crosses a boundary) – the band of the returned fit, not the best band seen while escalating, which k_quality_k_trace carries; k_quality_reached is TRUE/FALSE for an "ok" / "good" target (NA for "report" / "none"), never silently downgraded; k_quality_miss classifies a miss as "resolution" (the $\hat{k}$ confidently outside the band – a grid deficiency) or "precision" (the

bootstrap CI straddling a boundary – the estimator's variance), NA when there is no miss to classify, and is what the escalation reads once refinement is exhausted; k_quality_reason records why it stopped; k_quality_rounds is the number of escalation re-fits performed (0 when the first fit sufficed or escalation was off); and k_quality_k_trace is the outer $\hat{k}$ round by round, starting at the first fit's, so a chase that did not descend monotonically is visible. The refinement rungs rebuild the proposal from the grid they just changed, so a rising trace is a re-founded proposal on a strictly larger grid rather than a worse fit, which is why the LAST round is returned rather than the lowest-$\hat{k}$ one.

- adaptive_grid_info – when adaptive_grid = TRUE, a list with triggered_axes (character) and n_points_added (integer) describing the refinement passes. NULL otherwise.
- local_ccd_info – when local_ccd engaged (or k_refine = "ccd"), a list with n_cells_refined, n_nodes_added, the refined cells, the n_cells_before / n_cells_after grid sizes, n_design_nodes (the cells that carry an in-cell design weight rather than their own mass), design_mass (the share of the integration weight sitting on them) and cell_share (the share each refined cell held on the BASE grid, before any node was placed). The two shares differ: the replacement nodes sit nearer the peak than the cell's own coordinate did, so refining raises the share the refined region holds, and reading both separates how concentrated the base grid already was from how much the refinement concentrated it. Three per-cell readings come with it, on three orthogonal axes, none of them gating anything. Shape: misfit (the standardized cubic magnitude of the part of the cell the design cannot represent, the one reading skew_max gates on, with misfit_declined / cells_declined for the cells put back as their own mass atoms). Centring: offset (the norm of the whitened gradient, i.e. how far the cell's own peak sat from the cell's coordinate in units of its marginal spread) and mode_gain (the same displacement in the cell's own curvature units, 0.5 * g' (-H)^-1 g in nats, read off the same fit's quadratic coefficients and comparable across cells whose curvature differs; NA where -H is not positive definite, a cell with no interior peak to be displaced from). Mass: log_mass_ratio (the cell's refined mass over the coarse atom it replaced, log_mass_refined - log_mass_coarse, both carried on the log scale with the cell's outer design weight in them) and max_node_weight (the share the single largest node takes of its own cell's refined mass). Each carries a _declined twin for the cells the gate put back, whose nodes were evaluated before the score was read. misfit certifies that the design can represent the cell's shape and nothing more: a cell can be exactly quadratic and still be read at a point that is not representative of it (offset / mode_gain), and refining it can still move how much mass it competes for against its unrefined neighbours (log_mass_ratio, the embedded-rule local error indicator of adaptive cubature). NULL when local CCD was off or declined (single-block, < 4 axes, an active phi_grid, or no peaked interior cell).
- integration_requested, integration_declined – what integration asked for, and why the requested CCD or adaptive lattice did not run. $integration names the integrator that RAN, and .nl_node_support() keys the interval construction off it, so a caller who asked for a moment rule and received a density grid reads the reason here rather than inferring it. integration_declined is NA_character_ when nothing was declined, and otherwise one of "axis_count" (fewer transformable latent axes than the requested mode's threshold), "unguessable_axis" (a CAR_proper rho_car or a non-BYM2 rho), "degenerate_axis" (a single-valued axis), "modefind_ridge" / "modefind_boundary" / "modefind_degenerate" / "modefind_failed", "hessian_singular", "hessian_not_pd", "copy_atom_components" (more copy atoms than the design splits into, see copy_atom_mass), "copy_atom_mass" (a declared atom mass outside [0, 1)) or "placement_budget" (placing the design would cost more inner solves than integrating the tensor grid it replaces; see the ccd_budget control

knob). Under `integration = "grid_adaptive"` it is one of `"adaptive_small_grid"` (fewer outer cells than `adaptive_grid_min_cells`), `"adaptive_degenerate_lattice"`, `"adaptive_seed_failed"` (no finite marginal in the coarse seed), `"adaptive_diffuse_seed"` / `"adaptive_flood_cap"` / `"adaptive_kept_cap"` (the posterior mass spreads over more than `adaptive_grid_max_frac` of the dense grid, read at the seed, during the flood, or on the closed kept region), or `"adaptive_error"` (the flood raised an error, reported as a warning carrying its message); the fit then integrates the dense tensor. Multi-block fits only.

- `ccd_modefind_evals`, `ccd_modefind_budget`, `ccd_modefind_rounds`, `ccd_modefind_seconds` – what PLACING the CCD cost: inner Newton solves spent by the mode-find (seed, step calibration and every round of every mixture component), the ceiling they were spent against, the rounds taken and the wall-clock seconds. Present on any multi-block fit that attempted a placement, including one that declined – a decline reports what it had already paid – and absent when no placement was attempted, which is not the same as a placement that cost nothing. Read alongside `$integration` to judge whether the CCD earned its design on this model rather than arguing it from the round caps.
- `weight_kind` – one entry per outer-grid cell, `"mass"` or `"design"`. A tensor cell holds the mass of its own cell and a CCD node holds a design weight, so a fit integrated by one rule reports one value throughout; a locally refined grid carries both, and this says which per cell instead of leaving it to be read off `integration`.
- `theta_interval_read`, `theta_interval_design_mass` – what the reported per-axis `theta_median` / `theta_ci_lo` / `theta_ci_hi` were read off, and how much of the integration weight sits on nodes whose cumulative sum is not a CDF. `"density"` is a weighted quantile over cells that discretize the posterior (`design_mass` 0), `"moment_rule"` an interval from the moments a central-composite design delivers (`design_mass` 1), and `"mixed"` the locally CCD-refined grid, which carries both kinds at once. A mixed support still reports the weighted quantile, which measured best against a converged reference in both `design_mass` regimes; the share is the regime variable, since on that part of the support the quantile is bounded by the refined cells' own grid neighbourhoods rather than by the posterior. Every nested path stamps the pair, not only the multi-block driver.
- `within_cell_requested`, `theta_within_cell`, `theta_within_cell_declined` – the WITHIN-CELL construction the same intervals were read with. The kind above says what the integrator left; this says how each cell's mass was spread inside its own box when the grid was read back. `"log_quadratic"` is the default and interpolates each row's node log densities; `"box_uniform"` (`control$within_cell`) puts the cumulative full mass at each cell edge; `"chord"` puts the cumulative mid-mass at each cell coordinate, the same masses over the same boxes with the knots moved half a cell. The construction is recorded per axis, and an axis whose cell partition could not be built falls back to `"chord"` on its own with the reason in `theta_within_cell_declined` (`"support_<kind>"`, `"single_node"`, `"boxes_do_not_tile"`, `"no_usable_node"`). A `"log_quadratic"` request falls back to `"box_uniform"` first, with `"unresolved"` (no row resolves its own conditional), `"not_a_grid_axis"` or `"no_node_density"` as the reason.
- `theta_cell_edge_coord`, `theta_cell_edge_declined` – per axis, the COORDINATE that axis's outer cell edges were mirrored in, and why the support the axis declares did not produce them. A declared support is authoritative: its mirrored edge is used when finite and inside it, and otherwise the axis reports its extreme grid coordinate – which the containment test has already placed inside the support – rather than falling through to a coordinate guessed from the node values, which is what put a lower bound of 0 on a `positive` axis and a bound above

1 on a unit one. An axis that declares no support at all keeps the guess by design, but its mirrored edge is checked for being a representable double that brackets the coordinates, and falls back to the extreme grid coordinate with "mirrored_edge_not_representable" when it is not. theta_cell_edge_declined is NA where nothing was declined.

- outer_grid_cell_width, outer_grid_axis_sd, outer_grid_h_over_sd – per axis, the cell width and the posterior SD in that axis's own coordinate, and their ratio. A within-cell reconstruction resolves an interval endpoint to within one cell, so how much of the reported width and of its realized coverage is a property of where the grid fell rather than of the posterior is governed by this ratio; below .nl_diag("grid_resolved") the cells are narrower than the posterior they discretize and the two constructions converge to each other. NA on an axis whose own marginal is maximal at an endpoint, which is the placement question outer_grid_railed_axes reports.
- outer_grid_resolution_declined – per axis, why that axis carries no ratio, from the closed vocabulary too_few_nodes, mode_at_edge, coord_not_finite, spacing_not_finite, stencil_degenerate, curvature_not_negative; NA on an axis that scored. The reasons are not interchangeable: mode_at_edge says the grid does not contain that axis's own posterior mode, which is a stronger statement about the fit than any ratio, and the whole-grid resolved verdict is withheld while one is present rather than being read off the axes that did score.
- prune_cheap_log_marginal, prune_mask, prune_n_pruned, prune_tol, prune_screen_iters – present only when prune = TRUE and the safety gate did not fall back. Cheap-pass log-marginals at every cell – the truncated solve's value plus half its Newton decrement, the second-order estimate of the converged value the screen ranks by, so a cell whose inner mode had far to move from its warm start is not ranked on where the truncation left it (gcol33/tulpa#919); that decrement is prune_screen_decrement – a logical mask of pruned cells, the pruned-cell count, the threshold actually applied, and the screening depth the driver actually ran at. Pruned cells have log_marginal = -Inf so they get zero weight under .nl_normalise_weights_safe.
- prune_log_gap_cut, prune_cheap_lm_spread, prune_min_keep, prune_n_floor_restored – the screen read on the scale it operates on: how many nats below the best screened cell the tolerance cut at, how many nats the screened surface spans, the kept-cell floor applied, and how many cells that floor put back. A cut that is a sliver of the spread is a tolerance with no resolution on this grid.
- prune_dropped_mass_bound, prune_n_repaired, prune_gate_mass – the posterior mass the dropped cells may carry, with each lifted by the worst screening error measured on a kept cell; how many dropped cells the driver solved in full to bring that under the gate; and the gate.
- outer_mode_carried – TRUE on a placement refit that read the outer mode it was laid from rather than finding it again, which it does unless the placed grid rails.
- prune_fallback_triggered, prune_fallback_reason – present only when a fallback to the full grid fired. The returned fit is the full-grid (unpruned) result; the reason string records whether the cheap-pass safety gate tripped or the screened grid left the placement pass no curvature.
- n_threads_outer_requested, n_threads_outer_realised – the outer-grid width asked for and the width the grid ran at. The driver clamps the request to the team the OpenMP environment hands out, so a fit launched under OMP_NUM_THREADS=1 with n_threads_outer = 10 runs serial; the pair is what a timing has to be recorded against.

- converged, n_iter, score_max – per cell, whether the inner Newton reached a mode within control$max_iter, the iterations it took, and the largest absolute component of the joint penalized score at the mode it reported (the residual a cell that stopped short stopped at). A cell that stopped without a mode is solved again from the mode of its nearest converged cell (nonconverged_resolve counts the cells that first stalled and those the re-solve recovered); one that still has no mode stays in the integration at the value it stopped at, is listed in nonconverged_cells with the posterior mass such cells carry in nonconverged_mass, and is named by diagnostics(). Past the screen's mass gate the fit warns.

(sigma, alpha) parameterization

Each arm's linear predictor reads

$$\eta_{arm} = X_{arm}\beta_{arm} + \sigma_{arm} \cdot z_s,$$

where z_s is a unit-precision latent (ICAR(tau=1) for ICAR/BYM2, or the BYM2 mix; CAR_proper uses the structure of $D - \rho_{car}W$). All arms share the donor amplitude σ from the outer-grid sigma_grid axis. The copy arm's amplitude is $\sigma_{arm} = \alpha \cdot \sigma$, where α is a direct outer-grid axis taken from copy$alpha_grid. The Cartesian product is over (sigma, [rho/rho_car,] alpha[, phi]). Direct α as an outer-grid axis (rather than a post-hoc ratio $\alpha = \sigma_{pos}/\sigma_{occ}$) avoids plug-in bias on the weakly-identified ratio at small n_pos and lets a regularizing hyperprior land on α directly.

References

Rue, Martino & Chopin (2009). Approximate Bayesian inference for latent Gaussian models by using integrated nested Laplace approximations. *JRSS-B* 71(2):319-392.

See Also

tulpa_nested_laplace() for the single-arm engine.

Examples

```
set.seed(1)
S <- 25L                                  # shared spatial units (chain graph)
nb <- lapply(seq_len(S), function(s) setdiff(c(s - 1L, s + 1L), c(0L, S + 1L)))
nn <- lengths(nb)
field <- as.numeric(scale(cumsum(rnorm(S, 0, 0.4))))
mk_arm <- function(m, fam) {
  si <- sample(S, m, replace = TRUE); x <- rnorm(m)
  lin <- 0.2 + 0.5 * x + 0.8 * field[si]
  y <- if (fam == "binomial") rbinom(m, 1L, plogis(lin)) else lin + rnorm(m, 0, 0.5)
  list(y = as.numeric(y), n_trials = rep(1L, m), X = cbind(1, x),
       spatial_idx = si, family = fam, phi = if (fam == "gaussian") 0.5 else 1)
}
prior <- list(type = "icar", n_spatial_units = S,
              adj_row_ptr = c(0L, cumsum(nn)), adj_col_idx = unlist(nb) - 1L,
              n_neighbors = nn, sigma_grid = c(0.3, 0.7, 1.4))
fit <- tulpa_nested_laplace_joint(
  responses = list(occ = mk_arm(200L, "binomial"), pos = mk_arm(200L, "gaussian")),
  prior = prior)
```

```
fit$theta_mean        # shared field amplitude, integrated across both arms
```

---

print.tulpa_temporal_field_fit

Extract temporal effects from a fitted model

Description

Extract posterior distributions of temporal effects from a fitted tulpa model with temporal specification.

Usage

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

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

temporal(
  object,
  component = "all",
  summary = FALSE,
  probs = c(0.025, 0.5, 0.975),
  ...
)

## S3 method for class 'tulpa_fit'
temporal(
  object,
  component = "all",
  summary = FALSE,
  probs = c(0.025, 0.5, 0.975),
  ...
)

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

## S3 method for class 'tulpa_temporal_posterior'
summary(object, probs = c(0.025, 0.5, 0.975), ...)

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

Arguments

x	A `temporal()` field specification, the fit `tulpa()` returns for one, a `tulpa_temporal_posterior` or a `tulpa_temporal_summary`.
...	Ignored
object	A `tulpa_fit` object fitted with `temporal` argument
component	Which component to extract for multi-scale models: `"all"` (default), `"trend"`, `"seasonal"`, or `"short_term"`.
summary	Logical; if TRUE, return summary statistics instead of full posterior draws.
probs	Quantiles to compute if `summary = TRUE`.

Details

`temporal()` is overloaded. Given a fitted model it is the accessor described here. Given a one-sided formula (or a named `formula =` / `structure =` argument) it is instead the inline varying-coefficient field constructor used in a `tulpa()` model formula, the temporal mirror of `spatial()`: `temporal(formula = ~ 1 + x || time, structure = "rw1")` declares a smooth temporal level (the intercept column) plus a temporally varying slope on each covariate column. `structure` is one of `"rw1"` (default), `"rw2"`, or `"ar1"`; only the double bar `||` (independent fields) is supported.

On a sampler fit the posterior is the field's own draws. On a nested-Laplace fit it is the Gaussian mixture the outer grid defines: each hyperparameter cell contributes its conditional mode and marginal variance, weighted by the cell's posterior weight, and `summary()` reports that mixture's exact mean, SD and quantiles. The `draws` of such a fit are sampled from the same mixture, each from its cell's joint Gaussian, so they carry the correlation between time points as well as each point's marginal, and a quantity derived from them (a trend, a difference between two times) has the right spread. A field so large that its per-cell covariance was not retained (more than 4 million values over the grid) is drawn with independent time points within each cell instead. The `time` column of the summary holds the time values the field was fitted on.

Value

A `tulpa_temporal_posterior` object

`summary()` on a `tulpa_temporal_posterior` returns a `tulpa_temporal_summary` data frame with one row per time point (per component for multi-scale fits), holding the posterior mean, SD and requested quantiles of the temporal effect.

See Also

`temporal_multiscale()`, `temporal_rw1()`

Examples

```
set.seed(131)
df <- data.frame(year = 1:40, x = rnorm(40))
df$count <- rpois(40, exp(1 + 0.2 * df$x))

fit <- tulpa(
  count ~ x,
```

```
  data = df,
  family = "poisson",
  temporal = temporal_multiscale("year", trend = "rw2", seasonal = 12),
  mode = "exact",
  control = list(n_iter = 200L, n_warmup = 100L, seed = 1L)
)

# Extract all temporal effects
temp_post <- temporal(fit)
summary(temp_post)
```

`priors_default` *Show default priors for a tulpa family*

Description

Display the default `tulpa_priors()` specification: the priors the simulation tools (`prior_predict()`, `tulpa_simulate()`) draw from, and a starting point for customizing them. `tulpa()` does not read a `tulpa_priors` object; the no-family printout closes with the defaults a `tulpa()` fit does use, which differ for the random-effect SDs (see `tulpa_priors()`).

Usage

```
priors_default(family = NULL, spatial = FALSE, temporal = FALSE)
```

Arguments

`family`	`NULL` (default) for the general printout, or a `tulpa_family` object – a model package's family constructor, such as a tulpaRatio ratio family, or one built with `tulpa_family()` – for the parameters that family carries. A family NAME (`"poisson"`) is not a `tulpa_family` and is refused; the built-in `family_names()` share the general printout.
`spatial`	Logical; if TRUE, include spatial priors. Default FALSE.
`temporal`	Logical; if TRUE, include temporal priors. Default FALSE.

Details

The family-specific overdispersion lines name the ratio families a model package registers (`negbin_negbin`, `negbin_gamma`, `poisson_gamma`, `gamma_gamma`, `beta_binomial`); they are not among tulpa's own `family_names()`.

Default priors in `tulpa_priors()` follow these principles:

- **Fixed effects (beta)**: Normal(0, 2.5) - weakly informative, allows coefficients roughly in [-5, 5] on the link scale.
- **Random effect SD (sigma)**: PC prior with P(sigma > 1) = 0.01 - favors simpler models with smaller variance components.

- **Overdispersion (phi)**: PC prior with P(phi > 10) = 0.01 on the NB2 size `phi` (larger `phi` is less overdispersion; `phi -> Inf` is the Poisson limit). The default keeps `phi` finite, allowing overdispersion while penalising extreme values.
- **Temporal correlation (rho)**: Beta(2, 2) - symmetric prior centered at 0.5, appropriate for AR(1) correlation.
- **Spatial mixing (rho_spatial)**: Beta(1, 1) = Uniform(0, 1) - no prior preference for structured vs. unstructured spatial variation.

Value

Invisibly returns a `tulpa_priors` object with the defaults. Primarily called for its side effect of printing.

See Also

`tulpa_priors()` for creating custom priors

Examples

```
# Defaults for all families (no family argument)
priors_default()

# Family-specific defaults take a tulpa_family object. Model packages
# (e.g. tulpaRatio) register rich families; a minimal one is enough here.
fam <- tulpa_family(
  name = "poisson_gamma",
  simulate_fn = function(eta, params, n_obs, ...) rpois(n_obs, exp(eta[[1]]))
)
priors_default(fam)

# Including spatial parameters
priors_default(fam, spatial = TRUE)

# Use as a starting point for customization
my_priors <- priors_default(fam)
my_priors$beta <- prior_normal(0, 1)  # Tighter prior on fixed effects
```

`prior_beta` *Beta prior*

Description

Specify a beta prior for a parameter bounded in (0, 1).

Usage

```
prior_beta(alpha = 1, beta = 1)
```

Arguments

`alpha`	First shape parameter. Must be positive.
`beta`	Second shape parameter. Must be positive.

Details

- alpha = beta = 1: Uniform(0, 1)
- alpha = beta = 2: Symmetric, peaked at 0.5
- alpha > beta: Skewed toward 1
- alpha < beta: Skewed toward 0

Value

A `tulpa_prior` object

Examples

```
prior_beta(1, 1)   # Uniform
prior_beta(2, 2)   # Symmetric, centered at 0.5
prior_beta(5, 2)   # Skewed toward 1 (for high autocorrelation)
```

`prior_exponential` *Exponential prior*

Description

Specify an exponential prior for a positive parameter.

Usage

```
prior_exponential(rate = 1)
```

Arguments

`rate`	Rate parameter (lambda). Must be positive.

Value

A `tulpa_prior` object

Examples

```
prior_exponential(1)
```

`prior_from_spec` *Build a* `prior` *list for* `tulpa_nested_laplace()` *from a tulpa spec object*

Description

Validates a `tulpa_temporal` or `tulpa_spatial` specification against `data`, then converts it to the prior list shape consumed by `tulpa_nested_laplace()`. Mainly an internal helper for callers that already have a fitted spec; users typically pass `spec` + `data` directly to `tulpa_nested_laplace()` instead.

Supported spec types:

- `tulpa_temporal` with `type in {"rw1", "rw2", "ar1"}`
- `tulpa_spatial`, areal `type in {"car", "icar", "car_proper", "bym2"}`. An areal spec returns the graph fields only – `type`, `spatial_idx`, `n_spatial_units`, `adj_row_ptr`, `adj_col_idx`, `n_neighbors` – plus `scale_factor` for bym2 and the eigenvalue-derived `rho_bounds` for proper CAR. No grid field is returned for any areal type: the outer axes are built later by the family's registry `defaults()` closure, and proper CAR's `(tau, rho)` grid is built there from those bounds.
- `tulpa_gp` / `tulpa_hsgp`, continuous `type in {"gp", "nngp", "hsgp"}`: validated against `data` and routed to the `nngp` / `hsgp` nested kernel through the shared converter.

SPDE is the one continuous field not built here – it carries its own (range, sigma) FEM integrator; call `fit_spde()` with the `spatial_spde()` spec.

Usage

```
prior_from_spec(spec, data)
```

Arguments

`spec`	A `tulpa_temporal` or `tulpa_spatial` object.
`data`	Data frame the spec resolves time/group/site indices against.

Value

A `prior` list ready for `tulpa_nested_laplace()`.

Examples

```
d <- data.frame(year = rep(1:10, each = 3))
pr <- prior_from_spec(temporal_ar1("year"), d)
str(pr)
```

prior_gamma — *Gamma prior*

Description

Specify a gamma prior for a positive parameter.

Usage

```
prior_gamma(shape = 2, rate = 0.1)
```

Arguments

shape	Shape parameter (alpha). Must be positive.
rate	Rate parameter (beta). Must be positive.

Details

Mean = shape/rate, Variance = shape/rate^2.

Value

A `tulpa_prior` object

Examples

```
prior_gamma(2, 0.1)  # Mean = 20, weakly informative
prior_gamma(1, 1)    # Exponential(1)
```

prior_half_cauchy — *Half-Cauchy prior*

Description

Specify a half-Cauchy prior for a positive parameter. Has heavier tails than half-normal, often used for variance parameters.

Usage

```
prior_half_cauchy(scale = 2.5)
```

Arguments

scale	Scale parameter. Must be positive.

Value

A `tulpa_prior` object

Examples

```
prior_half_cauchy(2.5)
```

---

prior_half_normal *Half-normal prior*

Description

Specify a half-normal (truncated at 0) prior for a positive parameter.

Usage

```
prior_half_normal(sd = 1)
```

Arguments

`sd`	Scale parameter. Must be positive.

Value

A `tulpa_prior` object

Examples

```
prior_half_normal(1)
```

---

prior_normal *Normal prior*

Description

Specify a normal prior for a parameter.

Usage

```
prior_normal(mean = 0, sd = 2.5)
```

Arguments

`mean`	Prior mean. Default 0.
`sd`	Prior standard deviation. Must be positive.

Value

A `tulpa_prior` object

Examples

```
prior_normal(0, 2.5)
prior_normal(0, 1)
```

---

prior_pc *Penalized complexity (PC) prior*

Description

Specify a PC prior for a positive parameter (typically a standard deviation). PC priors shrink toward simpler models by penalizing deviation from a base model.

Usage

```
prior_pc(U = 1, alpha = 0.01)
```

Arguments

`U`	Upper bound. P(x > U) = alpha.
`alpha`	Tail probability. Default 0.01.

Details

The PC prior is specified via: P(sigma > U) = alpha

- U is the upper bound you consider "large"
- alpha is the probability of exceeding it

This implies an exponential prior with rate = -log(alpha) / U.

Value

A `tulpa_prior` object

References

Simpson, D., Rue, H., Riebler, A., Martins, T. G., & Sorbye, S. H. (2017). Penalising model component complexity: A principled, practical approach to constructing priors. Statistical Science, 32(1), 1-28.

Examples

```
prior_pc(U = 1, alpha = 0.01)    # P(sigma > 1) = 0.01
prior_pc(U = 0.5, alpha = 0.05)  # Tighter, P(sigma > 0.5) = 0.05
```

prior_predict — *Prior predictive simulation*

Description

Draw datasets from the prior predictive distribution: parameters are sampled from their priors (no data conditioning) and pushed through the model's linear predictor and the family's simulator.

Useful for checking whether priors imply plausible data ranges before fitting.

Usage

```
prior_predict(
  formula,
  family,
  data,
  priors = NULL,
  n_draws = 100,
  seed = NULL,
  ...
)

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

Arguments

formula	A model formula (e.g., `y ~ x + (1 | g)`). For multi-process families, a list of formulas keyed by process name.
family	A `tulpa_family` object exposing a `simulate_fn` (see `tulpa_family()`). Model packages (tulpaRatio, tulpaObs) provide families; tests can build a minimal one with `tulpa_family()`.
data	Data frame containing covariates and grouping factors. Used for dimensions and design matrices; the response column may be absent or NA.
priors	Prior specification (`tulpa_priors()`). If `NULL`, uses defaults.
n_draws	Number of prior parameter draws. Default 100.
seed	Optional integer seed for reproducibility.
...	Passed to `family$simulate_fn`.
x	A `tulpa_prior_predict` object.

Value

A `tulpa_prior_predict` object: a list with

- `y`: list of length `n_draws`, each element the simulated response for that draw (matching whatever shape `family$simulate_fn` returns).

- theta: list of length n_draws of parameter draws (beta, sigma, RE coefficients u, family-specific extras).
- linpred: list of length n_draws, each a list of linear predictor vectors per process.
- family: the family used.
- n_draws, n_obs.

Examples

```
# Toy Gaussian family for illustration
fam <- tulpa_family(
  name = "gaussian",
  simulate_fn = function(eta, params, n_obs, ...) {
    rnorm(n_obs, eta[[1]], params$sigma_y)
  },
  extra_params = list(sigma_y = prior_half_normal(1))
)
df <- data.frame(y = rep(0, 20), x = rnorm(20))
pp <- prior_predict(y ~ x, fam, df, n_draws = 50, seed = 1)
length(pp$y)  # 50
```

---

ranef *Random-effect summaries*

Description

Random-effect summaries

Usage

```
ranef(object, ...)

## S3 method for class 'tulpa_fit'
ranef(object, ...)
```

Arguments

object	A tulpa_fit object.
...	Ignored.

Details

What each backend reports for a group effect follows what it computes:

- Sampler tier and the RE-covariance Gibbs debias (tulpa_re_cov_gibbs(), reached by tulpa(..., control = list(re_cov = "gibbs"))) draw the random effects jointly with everything else, so estimate / sd / bounds are the empirical posterior summaries.

- The RE-covariance integrator (tulpa_re_cov_nested()) carries a Gaussian per-group posterior at each Sigma node; the reported summaries are the exact moments and quantiles of the weighted mixture of those, so they carry both the within-node curvature and the Sigma uncertainty. A group effect the subspace debias selected (control$subspace_debias) is moved by the Metropolis sampler at every node instead, and is reported from those draws.
- The Laplace tier reports the conditional mode with no spread (sd and the bounds are NA), which is the only per-group quantity it forms.

The source column says per row which of these produced it: "sampled" for a posterior draw summary, "mixture" for the node mixture, "mode" for a conditional mode. A fit whose backend never forms a per-group posterior at all (the adaptive Gauss-Hermite inner marginal integrates each group out by quadrature) errors with that reason rather than returning an empty table, which would be indistinguishable from a model with no random effects. A model that genuinely has none returns a zero-row data frame.

Value

Data frame with one row per random-effect coefficient: term (the group level, and the coefficient for a random slope), estimate, sd, the 2.5% / 97.5% bounds conf.low / conf.high, and source (which construction the row came from). sd and the bounds are NA on a backend that reports a point per group (see Details).

See Also

field_effects() for a spatial field, smooth_effects() for an s(x) term.

Examples

```
set.seed(1)
df <- data.frame(x = rnorm(100), g = factor(rep(1:10, 10)))
df$y <- rpois(100, exp(0.3 * df$x))
fit <- tulpa(y ~ x + (1 | g), data = df, family = "poisson")
ranef(fit)
```

---

residuals.tulpa_fit *Residuals from a tulpa fit*

Description

Population-level residuals from the fixed-effect fitted mean: "response" is y - E[y | eta] on the response scale (trial-scaled for binomial, offset included); "pearson" additionally scales by the family standard deviation sqrt(Var(y | eta)) at the fitted linear predictor. Random effects are held at zero, matching fitted(). On a zero-inflated fit both the mean and the variance are the mixture's.

Usage

```
## S3 method for class 'tulpa_fit'
residuals(object, type = c("pearson", "response"), ...)
```

Arguments

`object`	A `tulpa_fit` object carrying `$y` and `$model_matrix`.
`type`	`"pearson"` (default) or `"response"`.
`...`	Ignored.

Value

Numeric vector of length `nobs(object)`.

`re_cov_pc_lkj_prior` *PC + LKJ hyperprior for a random-effect covariance*

Description

Construct the default weakly-informative hyperprior used by `tulpa_re_cov_nested()` for one covariance block: independent Penalized-Complexity (PC) priors on the marginal standard deviations `sigma_i` together with an LKJ prior on the correlation matrix `R` (correlated block) or no correlation (diagonal block), returned as a `log_prior_theta` function in the block's integration coordinates.

Usage

```
re_cov_pc_lkj_prior(n_coefs, prior_sigma = NULL, eta = NULL, correlated = TRUE)
```

Arguments

`n_coefs`	Number of coefficients `c` in the RE block.
`prior_sigma`	`c(U, alpha)` giving `P(sigma_i > U) = alpha`, applied independently to every marginal SD. `NULL` (the default) is `c(3, 0.01)`, the engine's prior on every field scale.
`eta`	LKJ shape. `NULL` (the default) is 2. `eta = 1` is uniform on correlation matrices; larger values favour weaker correlations. Ignored for a diagonal block.
`correlated`	`TRUE` (default) for a full covariance block (log-Cholesky coordinates, LKJ prior); `FALSE` for a diagonal / uncorrelated block (log-SD coordinates, no correlation). For `n_coefs = 1` the two coincide.

Details

For a correlated block the prior is specified on the natural scale, `p(sigma, R) = LKJ(R | eta) * prod_i PC(sigma_i)`, then pushed to the log-Cholesky coordinates `theta` of `Sigma = L L'` by the exact change-of-variables Jacobian. For a diagonal (uncorrelated) block the LKJ factor drops and `theta_i = log sigma_i` with Jacobian `sum_i theta_i`.

PC prior (Simpson et al. 2017) on each marginal SD: exponential with rate `lambda = -log(alpha) / U`, so `P(sigma_i > U) = alpha` – the `prior_sigma = c(U, alpha)` convention also used by the SPDE prior in tulpa.

LKJ prior (Lewandowski et al. 2009) on the correlation matrix: `p(R) = det(R)^(eta - 1) / c_d(eta)`. `eta = 1` is uniform over correlation matrices; `eta > 1` concentrates toward the identity. The normalizing constant is kept, so the prior is a density and a grid's evidence can be read under it.

Jacobian (correlated block): with `theta` packing `log L_ii` on the diagonal and the raw strict-lower entries of `L`, the change of variables from `(sigma, R)` to `theta` adds `sum_i (c + 2 - i) * log L_ii - c * sum_i log sigm` to `log p(sigma, R)`. (Composition of the log-diagonal map, the standard Cholesky-to-covariance Jacobian `2^c prod_i L_ii^(c+1-i)`, and the covariance-to-`(sigma, R)` Jacobian; verified against numerical differentiation in `test-re-cov-prior.R`.)

Value

A `function(theta)` returning the scalar log prior density in the block's integration coordinates, suitable for one block of the `log_prior_theta` argument of `tulpa_re_cov_nested()`.

See Also

`tulpa_re_cov_nested()`

Examples

```
# Log prior of a correlated 2 x 2 block at the origin of its log-Cholesky
# coordinates (log sd_1, log sd_2, L_21):
lp <- re_cov_pc_lkj_prior(2)
lp(c(0, 0, 0))
# A diagonal block takes one log-SD coordinate per coefficient:
re_cov_pc_lkj_prior(2, correlated = FALSE)(c(0, 0))
```

`rubins_pool` *Pool multiple imputation draws via Rubin's rules*

Description

Pools per-imputation block fits via Rubin's rules. When every draw also carries a per-coefficient skewness vector `$gamma`, the third cumulant is pooled by the law of total cumulants

$$\kappa_3 = E[\kappa_3(X|k)] + 3\,\mathrm{Cov}(\mu_k, \sigma_k^2) + \kappa_3(\mu_k),$$

giving a pooled skewness `$gamma` alongside the usual `$mean` / `$se`. If any draw is missing `$gamma`, the third-cumulant path is skipped.

kappa3 is the third cumulant of the equal-weight mixture of the draws, so gamma is standardized by that mixture's variance, mean(se^2) + mean((beta - mean(beta))^2), not by V_total: Rubin's (1 + 1/K) inflation of the between-draw variance is a correction to the standard error for finite K, not a moment of the mixture.

Usage

```
rubins_pool(draws)
```

Arguments

draws	List of K draws. Each draw is a named list of submodel results, each with beta (numeric vector), se (numeric vector, same length as beta), and optionally gamma (numeric vector, same length as beta). Every draw of a submodel must carry the same number of coefficients. A submodel absent from some draws is pooled over the draws that carry it; one carried by a single draw is skipped with a warning.

Value

A named list of pooled submodel summaries (every submodel any draw carries), each with mean, se, V_within, V_between, V_total, K (number of draws that contributed), and when applicable gamma (pooled skewness) and kappa3 (pooled third cumulant).

Examples

```
# Three imputation draws of a two-coefficient submodel.
draws <- list(
  list(m = list(beta = c(1.0, 0.50), se = c(0.2, 0.10))),
  list(m = list(beta = c(1.2, 0.45), se = c(0.2, 0.10))),
  list(m = list(beta = c(0.9, 0.55), se = c(0.2, 0.10)))
)
pooled <- rubins_pool(draws)
pooled$m$mean
pooled$m$se
```

---

sbc *Simulation-based calibration*

Description

Scores whether an inference algorithm's posterior is CALIBRATED, by reading the whole marginal CDF rather than one or two nominal levels. Draw a truth from the distribution the fit updates, simulate a data set at that truth, fit, and take the probability integral transform (PIT) of the truth under the reported posterior. Under exact inference those PIT values are exactly Uniform(0, 1), so the entire ECDF is the measurement (Talts et al. 2018).

Usage

```
sbc(object, ...)

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

## S3 method for class 'character'
sbc(
  object = c("prior_predictive", "posterior"),
  simulator = NULL,
  fitter = NULL,
  model = NULL,
  n_sim = 100L,
  quantities = NULL,
  flat_prior = character(),
  level = 0.95,
  seed = 0L,
  n_ref = NULL,
  control = list(),
  ...
)

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

## S3 method for class 'sbc'
summary(object, baseline = NULL, ...)

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

## S3 method for class 'sbc'
plot(x, arm = NULL, quantity = NULL, folded = FALSE, ...)
```

Arguments

`object`	An sbc result.
`...`	Ignored.
`simulator`, `fitter`	The prior-predictive callbacks; see the contract above. Required for `experiment = "prior_predictive"`.
`model`	The list of posterior-SBC callbacks. Required for `experiment = "posterior"`.
`n_sim`	Number of simulations.
`quantities`	Optional character vector restricting which quantities are scored. Default scores every quantity every arm reports.
`flat_prior`	Character vector of scored quantities held FIXED under a flat prior, admitted by a structural argument the caller is asserting. Prior-predictive only.

`level`	Simultaneous band level.
`seed`	Offset added to the simulation index, so simulation `s` runs the callbacks at `seed + s`.
`n_ref`	Optional; when supplied it is passed to `fitter` (or to `model$arms`) as `n_ref`, the number of reference values a rank predictive is formed against. The callback must accept it.
`control`	List of knobs: `progress` (default `FALSE`) and `rand_seed`, the pinned stream the within-atom randomizing uniforms come from (default the driver's own, so a result is reproducible and the fits are not perturbed by asking for the diagnostic).
`x`	An `sbc` result.
`baseline`	Optional arm name. When given, the summary also carries the PAIRED CRPS differences of every other arm against it, seed by seed. A negative `delta` is the arm scoring better. This is refused on an experiment where the CRPS is not a proper posterior score.
`arm`, `quantity`	Optional character vectors selecting which panels to draw. Default draws every (arm, quantity).
`folded`	Plot the folded PIT instead of the raw one.

Value

An object of class `sbc`:

`pit` one row per (simulation, arm, quantity): the truth, the raw and folded PIT, the CRPS and the predictive kind.

`report` one row per (arm, quantity): the KS distance, whether the ECDF stayed inside the simultaneous band, the exact uniformity p-value, the same three folded, and the mean CRPS with its standard error.

`bands` the calibrated simultaneous band, by sample size.

`premises` what each guard concluded.

`crps_role` the role the CRPS column has under this experiment.

The two experiments

`"prior_predictive"` ordinary SBC. `theta ~ p(theta)`, `y ~ p(y | theta)`, fit, PIT. It reports self-consistency AVERAGED over the whole generative distribution.

`"posterior"` calibration CONDITIONAL on an observed data set (Sailynoja et al. 2026, Algorithm 2). `theta' ~ pi(theta | y_obs)`, `y ~ pi(y | theta')`, and the PIT is taken under the AUGMENTED posterior `pi(theta | y, y_obs)`. That is ordinary SBC with `pi(theta | y_obs)` in the role of the prior, so the same band, the same folded read and the same proper score all carry over – and it needs no proper prior.

A fixed-truth sweep is not an SBC experiment and is not offered here: its PIT is not uniform under correct inference and its CRPS is a descriptive loss, not a proper posterior score.

What is reported

Three reads of the same PIT sample, per (arm, quantity):

raw the PIT ECDF against an exact SIMULTANEOUS band. A pointwise binomial band is not simultaneous – at n = 100, holding each order statistic at 95 percent holds all of them together at 0.4471 – so the band here is calibrated by bisection against the exact crossing probability of the uniform order statistics.

folded `2 |u - 1/2|`, also uniform, and where a symmetric over- or under-dispersion shows after cancelling in the raw ECDF.

CRPS the strictly proper score, closed form for the nested tier's own Gaussian mixture, paired seed by seed through `summary(x, baseline = )`.

Every discrete PIT (a rank, a grid axis, a draw set) is randomized within its atom, `u = F(theta^-) + V P(theta)`, so one uniform reference and one band serve every quantity. Reading `rank / n_ref` against a continuous uniform is the classic silent SBC bug.

The callback contract

For `experiment = "prior_predictive"`:

`simulator(seed)` returns a list carrying `theta`, a named numeric vector of the true values of the scored quantities, plus whatever the fitter needs. It must be a pure function of its seed.

`fitter(d)` returns a named list of ARMS, each a named list over quantities, each entry a predictive built by `sbc_mixture()` and its siblings. Several arms read off one solve per seed, so an arm-to-arm difference carries no fit-to-fit noise.

For `experiment = "posterior"`, `model` is a list of six callbacks – `data_obs`, `fit(data)`, `draw_theta(fit, seed)`, `simulate(theta, seed)`, `pool(data_obs, replicate)`, `arms(fit, data)` – plus an optional `group_ids(data)` used to verify the fresh-groups premise below. The driver hands `draw_theta` and `simulate` DIFFERENT seeds, so `set.seed(seed)` at the top of each is the correct fixture; sharing one makes the replicate's noise a function of the truth, which is not `p(y | theta')`.

Two guards

The prior-predictive experiment draws the truth from the prior, so an IMPROPER prior cannot be used – and the nested-Laplace door puts no prior on the fixed effects. A scored quantity whose truth does not move across simulations was not drawn from a proper prior, and `sbc()` errors on it before spending the fits, pointing at `experiment = "posterior"`. A location parameter whose flat prior leaves the PIT uniform by a structural argument is admitted through `flat_prior`, which is itself checked and travels on the result.

The posterior experiment rests on two premises, each of which silently invalidates the result when broken. The augmented posterior must condition on BOTH data sets, so a `pool()` returning no more than the replicate (or no more than the observed data) is refused. And the replicate must be conditionally independent of the observed data given theta, which for a hierarchical model whose group effects are integrated out means FRESH groups. Supply `group_ids` and the observable half of that is verified – the group LABELS are disjoint – and omit it and the result records the premise as unverified rather than assumed. The other half, that the simulator drew those groups' effects from the prior rather than conditionally on the observed data, is not visible from outside the callback and is not claimed.

References

Talts, Betancourt, Simpson, Vehtari & Gelman (2018). Validating Bayesian inference algorithms with simulation-based calibration. arXiv:1804.06788.

Sailynoja, Schmitt, Buerkner & Vehtari (2026). Posterior SBC: simulation-based calibration checking conditional on data. *Statistics and Computing* 36:78. doi:10.1007/s11222026108259

Grimit, Gneiting, Berrocal & Johnson (2006). The continuous ranked probability score for circular variables and its application to mesoscale forecast ensemble verification. *QJRMS* 132(621C):2925-2942.

See Also

`sbc_mixture()` for the predictive shapes a fitter reports, `diagnostics()` for the single-fit reliability band that screens what this measures, `pit_residuals()` for single-fit posterior-predictive PIT residuals (a different quantity).

Examples

```
# A conjugate normal-normal model, whose posterior is exact, so its PIT must
# be uniform: the harness scoring itself.
sim <- function(seed) {
  set.seed(seed)
  mu <- rnorm(1)
  list(y = rnorm(10L, mu, 1), theta = c(mu = mu))
}
fitter <- function(d) {
  v <- 1 / (1 + length(d$y))
  list(exact  = list(mu = sbc_normal(v * sum(d$y), sqrt(v))),
       narrow = list(mu = sbc_normal(v * sum(d$y), sqrt(v) / 2)))
}
res <- sbc("prior_predictive", simulator = sim, fitter = fitter, n_sim = 60L)
res
summary(res, baseline = "exact")
```

`sbc_predictive` *Predictive shapes an SBC fitter reports*

Description

The tagged representations `sbc()` reads. A `fitter` (or a posterior-SBC `arms`) callback returns a named list of ARMS, each a named list over quantities, and each entry is one of these – the shape the backend actually reports for that quantity. Everything downstream (the PIT, the CRPS, drawing from a predictive) dispatches on the `kind` tag, so a new backend shape is one entry in three switches rather than a parallel scorer.

Usage

```
sbc_mixture(mu, var, w = NULL)

sbc_normal(mean, sd)

sbc_discrete(support, probs)

sbc_rank(rank, n_ref)

sbc_draws(x)
```

Arguments

mu, var, w	Component means, variances (>= 0) and non-negative weights, one of each per component. w defaults to equal weights and is normalized.
mean, sd	Mean and standard deviation (>= 0) of a single Gaussian.
support, probs	Finite support and its non-negative probabilities, one per support point; normalized.
rank, n_ref	The rank in 0:n_ref of the truth among n_ref reference values, and that reference count.
x	Posterior draws.

Details

These are the extension point, not alternative front doors: sbc() is the verb, and these are the argument type it consumes.

sbc_mixture() is what an outer hyperparameter grid defines for a fixed effect – component k is N(mu_k, var_k) with weight w_k, which is exactly the mixture a nested-Laplace fit reports. sbc_normal() is the one-component case, with its own constructor so a collapsed-moment read says what it is. sbc_discrete() is a distribution on a finite support, which is what a discrete hyperparameter grid defines for its own axis. sbc_rank() is a rank of the truth among n_ref reference values, which is what a joint log-likelihood comparison against posterior draws produces – it needs no entry in the simulator's theta, since the comparison against the truth already happened when the rank was formed. sbc_draws() is for a backend reporting no analytic marginal.

The last three have ATOMS, so their PIT is randomized within the atom by sbc(); reading a rank against a continuous uniform is the classic silent SBC bug.

Value

A list carrying a kind tag and that shape's parameters.

See Also

sbc()

Examples

```
sbc_normal(0.3, 0.1)
sbc_mixture(mu = c(0, 1), var = c(1, 4), w = c(0.7, 0.3))
sbc_discrete(support = c(0.5, 1, 2), probs = c(0.2, 0.5, 0.3))
```

---

select_main_params *Select the "main" model parameters for diagnostic display*

Description

Drops per-element latent-field entries (names ending in a bracketed index, e.g. `u[12]`, `w[3]`) so plots and summaries focus on scalar coefficients and hyperparameters rather than thousands of latent values.

Usage

```
select_main_params(param_names)
```

Arguments

`param_names`	Character vector of parameter names.

Value

The subset of `param_names` that are not bracketed-index entries (or the full vector if that would be empty).

Examples

```
select_main_params(c("beta[1]", "beta[2]", "log_sigma_re", "re[1]", "re[2]"))
```

---

simulate.tulpa_fit *Simulate responses from a fitted tulpa model*

Description

Base-R alias for `posterior_predict()`: each simulation is one posterior predictive replicate at the training data. A categorical fit (`tulpa_multinomial()`, `tulpa_ordinal()`) simulates factor columns carrying the response levels.

Usage

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

Arguments

`object`	A `tulpa_fit` object.
`nsim`	Number of simulated datasets (default 1).
`seed`	Optional integer seed (RNG state is restored on exit).
`...`	Ignored.

Value

A data frame with `nsim` columns (`sim_1`, ...), one row per observation, following the `stats::simulate()` convention: its `"seed"` attribute is the value of `seed` (with the RNG kind as attribute `"kind"`) when one was given, else the state of `.Random.seed` before simulation.

`smooth_effects` *Extract fitted covariate smooths*

Description

The grid-marginalized posterior mean of each `s(x)` term's latent values, one estimate per node (bin midpoint or unique covariate value). The latent block values are read from the fit's per-grid modes, weighted by the hyperparameter grid weights.

Usage

```
smooth_effects(object, term = 1L)
```

Arguments

`object`	A `tulpa_fit` from `tulpa()` with `s(...)` term(s) in the formula.
`term`	Which smoother: index or covariate name. Default 1.

Value

A data frame with columns `x` (node location) and `estimate` (the posterior-mean smooth at that node), with the covariate name as an attribute `"var"`.

See Also

`field_effects()` for a spatial field, `ranef()` for random effects.

Examples

```
set.seed(1)
d <- data.frame(x = runif(300, -2, 2))
d$y <- rpois(300, exp(0.3 + sin(2 * d$x)))
fit <- tulpa(y ~ s(x), data = d, family = "poisson")
sm <- smooth_effects(fit)
plot(sm$x, sm$estimate, type = "l")
```

`spatial` *Areal spatially varying coefficient field*

Description

Declare one or more areal (CAR / Besag) fields over a graph from an **lme4**-style bar formula. The bar's left-hand side lists the coefficients that vary smoothly over the graph; the right-hand side names the graph-node index. Each coefficient becomes an independent CAR field, entering the linear predictor scaled by that coefficient's per-observation design value:

$$\eta_i = \ldots + \sum_c X_{ic}\, z_{g_i}^{(c)},$$

where g_i is the graph node of observation i, $z^{(c)}$ is the CAR field for design column c, and X_{ic} is that column's value at observation i. The intercept column is all ones, so `~ 1 || cell` is the ordinary spatial intercept field; a covariate column (e.g. `time`) gives a spatially varying slope on that covariate (a per-region trend).

Use it inline in a `tulpa()` model formula, the same way a random-effect bar is written:

```
y ~ time + spatial(graph = adj, formula = ~ 1 + time || cell) + (1 | site)
```

Usage

```
spatial(graph, formula, proper = FALSE, shared = NULL, by = NULL)

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

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

Arguments

`graph`	Symmetric adjacency matrix of the spatial graph (`[n_node x n_node]`, dense or sparse). One CAR field is defined over its nodes per coefficient.
`formula`	One-sided formula carrying a grouping bar, e.g. `~ 1 + time || cell`. The left-hand side is expanded with `stats::model.matrix()` into one CAR field per column (`1` = intercept field; a covariate = spatially varying slope on it; `0 +` drops the intercept). The right-hand side must be a single bare column naming the graph node. The double bar `||` builds independent fields (each its own precision); a single bar `|` builds correlated fields – a separable multivariate CAR where the per-cell coefficient vector shares a cross-covariance `Sigma` (covariance `Sigma (x) Q^-1`), so the intercept-slope correlation `rho` is shared across the graph.

proper — Logical; FALSE (default) builds intrinsic CAR (ICAR / Besag) fields with the sum-to-zero constraint (rho fixed at 1). TRUE builds proper CAR fields, each with its own precision Q = D - rho_car W and the spatial autocorrelation rho_car estimated from the data (one (sigma, rho_car) pair per field). summary() and print() report the per-field rho_car. Independent (||) only; correlated proper CAR (a single | with proper = TRUE) is a separate model.

shared — Optional shared-effect handle, passed through to the field blocks (see the model docs). Default NULL (shared).

by — Optional replicated-CAR factor: a bare column name (or a string) naming a factor in the model data. With L levels it builds one independent copy of the whole field per level – the field over the block-diagonal Kronecker graph I_L (x) Q (L disjoint copies of the graph) – with the hyperparameters shared across levels (one Sigma for |; one (sigma[, rho_car]) for ||). This is INLA's replicate = / mgcv's s(cell, by = ...) generalised to the varying-coefficient bar, and is orthogonal to the bar character: | / || sets the covariance among the coefficient columns within a field, while by sets how many independent replicates of the whole field exist. Default NULL (one field). Correlated proper CAR (| with proper = TRUE) stays out of scope with or without by.

x — A spatial() field specification, or the fit tulpa() returns for one.

... — Ignored.

Details

Each field is independent (its own precision), matching INLA's two separate f(cell, model = "besag") and f(cell.slope, time, model = "besag") fields. Nesting (a / b), interaction, or expression grouping is rejected: the grouping must be a single graph node. Add ordinary nested random effects, e.g. (1 | site), as separate terms.

Value

A tulpa_spatial_field object describing the field(s). It is expanded into one CAR block per design column at fit time, when the data is available.

See Also

spatial_car() for a single areal field passed via the spatial = argument, spatial_svc() for the coordinate-based (Gaussian-process) spatially varying coefficient.

Examples

```
# Chain graph over 10 cells
adj <- matrix(0, 10, 10)
for (i in 1:9) adj[i, i + 1] <- adj[i + 1, i] <- 1

# Spatial intercept plus a spatially varying time slope
f <- spatial(graph = adj, formula = ~ 1 + time || cell)
print(f)
```

spatial_bym2 *BYM2 spatial structure*

Description

Specify a Besag-York-Mollie 2 (BYM2) spatial random effect. BYM2 decomposes the spatial effect into a structured (ICAR) component and an unstructured (IID) component, with a mixing parameter controlling the proportion of variance attributable to spatial structure.

BYM2 is preferred over plain CAR when you want to:

- Distinguish structured vs unstructured spatial variation
- Have an interpretable spatial fraction parameter
- Use the scaling from Riebler et al. (2016)

Usage

```
spatial_bym2(
  adjacency,
  level = c("group", "obs"),
  group_var = NULL,
  shared = NULL,
  scale_factor = NULL,
  parameterization = c("standard", "collapsed")
)
```

Arguments

`adjacency`	Symmetric adjacency matrix (`[n_units x n_units]`).
`level`	Either `"group"` (one effect per level of `group_var`) or `"obs"` (one effect per row of the data; `nrow(data)` must equal `nrow(adjacency)`).
`group_var`	Name of the grouping variable in the data; required when `level = "group"`.
`shared`	Optional shared-effect handle (see model docs).
`scale_factor`	Scaling factor for the ICAR component. If NULL (default), computed from the adjacency matrix following Riebler et al. On a graph with several connected components each component is scaled to unit generalized variance separately and an isolated node (island) gets unit variance on its structured part, following Freni-Sterrantino et al. (2018). A supplied value is one scale for the whole graph.
`parameterization`	`"standard"` (default) or `"collapsed"` (deprecated).

Value

A `tulpa_spatial` object

References

Riebler, A., Sorbye, S. H., Simpson, D., & Rue, H. (2016). An intuitive Bayesian spatial model for disease mapping that accounts for scaling. Statistical Methods in Medical Research, 25(4), 1145-1165.

Freni-Sterrantino, A., Ventrucci, M., & Rue, H. (2018). A note on intrinsic conditional autoregressive models for disconnected graphs. Spatial and Spatio-temporal Epidemiology, 26, 25-34.

Examples

```
# Create adjacency matrix for 10 regions (chain structure)
adj <- matrix(0, 10, 10)
for (i in 1:9) {
  adj[i, i+1] <- adj[i+1, i] <- 1
}

# Create BYM2 spatial structure
bym2 <- spatial_bym2(adj, level = "group", group_var = "region")
print(bym2)


# Disease mapping with BYM2 spatial smoothing
set.seed(456)
n_regions <- 10
epi_data <- data.frame(
  region = factor(rep(1:n_regions, each = 4)),
  age = rnorm(n_regions * 4, 50, 10)
)
epi_data$cases <- rbinom(nrow(epi_data), size = 100, prob = 0.15)

fit <- tulpa(
  cases ~ age + spatial(region),
  spatial = spatial_bym2(adj, level = "group", group_var = "region"),
  data = epi_data,
  family = "binomial",
  n_trials = rep(100L, nrow(epi_data)),
  mode = "laplace"
)
summary(fit)
```

`spatial_car` *CAR / ICAR spatial structure*

Description

Constructs a conditional autoregressive spatial random effect from an adjacency matrix. With `proper = FALSE` (the default) this is the improper CAR / ICAR (`type = "car"`); with `proper = TRUE` it returns the same object as `spatial_car_proper()`.

Usage

```
spatial_car(
  adjacency,
  level = c("group", "obs"),
  group_var = NULL,
  proper = FALSE,
  shared = NULL,
  parameterization = c("standard", "collapsed")
)
```

Arguments

adjacency	Symmetric adjacency matrix ([n_units x n_units]).
level	Either "group" (one effect per level of group_var) or "obs" (one effect per row of the data; nrow(data) must equal nrow(adjacency)).
group_var	Name of the grouping variable in the data; required when level = "group".
proper	If TRUE, use proper CAR (type = "car_proper"); else ICAR (type = "car").
shared	Optional shared-effect handle (see model docs).
parameterization	"standard" (default) or "collapsed" (deprecated).

Value

A tulpa_spatial object with type = "car" (or "car_proper" when proper = TRUE).

See Also

spatial_car_proper(), spatial_bym2().

Examples

```
adj <- matrix(0, 10, 10)
for (i in 1:9) adj[i, i + 1] <- adj[i + 1, i] <- 1
spatial_car(adj, level = "group", group_var = "region")
```

---

spatial_car_proper *Proper CAR spatial structure*

Description

Convenience wrapper for spatial_car(..., proper = TRUE). Creates a proper conditional autoregressive (CAR) spatial random effect with the autocorrelation parameter rho estimated from the data.

Use this when you want spatial autocorrelation to be a parameter of the model rather than fixed at 1 (as in ICAR). rho ~= 0 collapses to IID, rho ~= 1 approaches ICAR.

Usage

```
spatial_car_proper(
  adjacency,
  level = c("group", "obs"),
  group_var = NULL,
  shared = NULL
)
```

Arguments

adjacency	Symmetric adjacency matrix ([n_units x n_units]).
level	Either "group" (one effect per level of group_var) or "obs" (one effect per row of the data; nrow(data) must equal nrow(adjacency)).
group_var	Name of the grouping variable in the data; required when level = "group".
shared	Optional shared-effect handle (see model docs).

Value

A tulpa_spatial object with type = "car_proper".

See Also

spatial_car() for ICAR (rho fixed at 1), spatial_bym2() for the BYM2 decomposition.

Examples

```
adj <- matrix(0, 10, 10)
for (i in 1:9) adj[i, i+1] <- adj[i+1, i] <- 1
spec <- spatial_car_proper(adj, level = "group", group_var = "site")
print(spec)
```

---

spatial_gp *Gaussian process spatial structure (NNGP)*

Description

Specify a Gaussian-process spatial random effect, approximated with a nearest-neighbour GP (NNGP) for scalability. Captures smooth spatial variation from point-referenced coordinates.

Usage

```
spatial_gp(
  coords,
  approx = c("nngp", "hsgp"),
  cov = c("exponential", "matern"),
  nu = 1.5,
  nn = 15,
  m = 6,
  c = 1.5,
  sigma_prior_U = 1,
  sigma_prior_alpha = 0.01,
  shared = NULL,
  scale_coords = TRUE,
  parameterization = c("noncentered", "centered", "collapsed")
)

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

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

Arguments

`coords` A formula (`~ lon + lat`) or character vector naming the coordinate variables in the data. With `approx = "nngp"` the coordinate DIMENSION is however many are named: two for a map, one for a transect or a depth profile, three for a depth-resolved domain. The neighbour graph and the neighbour covariance both read every column. `approx = "hsgp"` takes exactly two, and so does any sampler mode, because both store coordinates at a fixed 2-D stride.

`approx` GP approximation: `"nngp"` (default, a nearest-neighbour GP with the `cov` / `nu` / `nn` arguments) or `"hsgp"` (a Hilbert-space basis GP with `m` functions per dimension and boundary factor `c`).

`cov` Covariance function (NNGP only). One of `"exponential"` or `"matern"`.

`nu` Matern smoothness parameter, one of `1.5` or `2.5`. Used only when `cov = "matern"` (`nu = 0.5` is `cov = "exponential"`).

`nn` Number of nearest neighbours used in the NNGP approximation.

`m` Number of HSGP basis functions per dimension (`approx = "hsgp"`).

`c` HSGP boundary factor, `>= 1` (`approx = "hsgp"`).

`sigma_prior_U`, `sigma_prior_alpha` Penalized-complexity prior on the field's marginal standard deviation (`approx = "hsgp"`), calibrated so that `P(sigma > sigma_prior_U) = sigma_prior_alpha`. Defaults to `P(sigma > 1) = 0.01`. `sigma_prior_U` must be positive and `sigma_prior_alpha` must lie in `(0, 1)`.

`shared` Whether the spatial effect is shared across processes in a multi-process model. `NULL` (default) shares the effect; `FALSE` fits process-specific effects and emits a warning.

scale_coords
: Logical. Standardize coordinates before fitting (default TRUE): each column is centred and all are divided by one common factor, so distances keep their proportions and an isotropic kernel stays isotropic. spatial_range() reports the range in the original coordinate units either way.

parameterization
: Latent parameterization for the exact-NUTS field. One of "noncentered" (default; samples z ~ N(0, I) and reconstructs the field as w = f(z, sigma2, phi), avoiding the field/hyperparameter funnel), "centered" (places the NNGP density on the field directly), or "collapsed" (deprecated).

x
: A tulpa_gp or tulpa_hsgp specification.

...
: Ignored.

Value

A tulpa_gp object (also of class tulpa_spatial).

See Also

spatial_car(), spatial_bym2() for areal spatial effects.

Examples

```
# GP spatial specification from coordinate columns
spatial_gp(~ lon + lat)
spatial_gp(~ lon + lat, cov = "matern", nu = 1.5)
```

---

spatial_multiscale *Multi-scale Gaussian process spatial structure*

Description

Specify a multi-scale spatial random effect that decomposes spatial variation into local (fine-scale) and regional (broad-scale) components. Each scale has its own range and variance parameters.

This is particularly useful for large datasets (>100k observations) where spatial patterns exist at multiple scales.

Usage

```
spatial_multiscale(
  coords,
  scales = c("local", "regional"),
  approx = c("nngp", "hsgp"),
  m = 6L,
  c_boundary = 1.5,
  range_local = c(0.01, 1),
  range_regional = c(1, 10),
```

```
  cov = c("exponential", "matern"),
  nu = 1.5,
  nn_local = 10,
  nn_regional = 30,
  shared = NULL,
  scale_coords = TRUE,
  sampler = c("auto", "noncentered", "centered", "interweaved")
)

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

Arguments

coords	A one-sided formula specifying coordinate columns (e.g., ~ lon + lat), or a character vector of length 2 with column names.
scales	Character vector specifying scale names. Default: c("local", "regional").
approx	Approximation method: "nngp" (default) for Nearest Neighbor GP; "hsgp" for Hilbert Space GP (faster for smooth fields).
m	Number of HSGP basis functions per dimension (default 6). Only used when approx = "hsgp". Total basis functions will be m^2.
c_boundary	Boundary factor for HSGP domain extension (default 1.5). Only used when approx = "hsgp".
range_local	Plausible range interval for the local scale as c(lower, upper) in coordinate units. Default: c(0.01, 1) (after scaling). Under exact NUTS this is not a hard box: lower anchors a PC prior on that scale's range (P(range < lower) = 0.05, the same prior spatial_gp() uses), and the pair places the sampler's starting range at their geometric mean. The range itself is free on (0, Inf).
range_regional	Plausible range interval for the regional scale, read the same way. Default: c(1, 10) (after scaling). Keeping the two intervals separated is what identifies the scales against each other.
cov	Covariance function: "exponential" (default) or "matern".
nu	Smoothness parameter for Matern covariance, one of 1.5 or 2.5.
nn_local	Number of nearest neighbors for local scale. Default 10.
nn_regional	Number of nearest neighbors for regional scale. Default 30.
shared	Logical; if TRUE (default), spatial effects enter both all processes.
scale_coords	Logical; if TRUE (default), coordinates are centred and divided by one common factor before computing distances (see spatial_gp()).
sampler	Latent parameterization for the exact-NUTS field. "auto" (default) and "noncentered" sample z ~ N(0, I) per scale and reconstruct each field as w = f(z, sigma2, phi), avoiding the field/hyperparameter funnel; "centered" places the NNGP density on each field directly. "interweaved" alternates between parameterizations and is not implemented on the exact-NUTS path.
x	A tulpa_multiscale specification.
...	Ignored.

Details

The multi-scale model decomposes spatial variation additively:

$$\eta(s) = X\beta + w_{local}(s) + w_{regional}(s)$$

where each component follows an independent Gaussian process:

$$w_{local}(s) \sim GP(0, \sigma^2_{local} C(\phi_{local}))$$

$$w_{regional}(s) \sim GP(0, \sigma^2_{regional} C(\phi_{regional}))$$

Identifiability: With sufficient data (>500 locations), the two scales are typically well-identified when prior ranges are non-overlapping. PC priors on variance components help prevent overfitting.

Computational cost: Approximately 1.5-2x the cost of single-scale GP, as two NNGP likelihoods must be evaluated.

Value

A `tulpa_multiscale` object

See Also

`spatial_gp()` for single-scale GP, `temporal_multiscale()` for multi-scale temporal effects

Examples

```
# Create multi-scale spatial structure
ms <- spatial_multiscale(
  ~ lon + lat,
  range_local = c(0.1, 0.5),
  range_regional = c(1, 5)
)
print(ms)


set.seed(101)
n <- 25
df <- data.frame(
  lon = runif(n, 0, 10),
  lat = runif(n, 0, 10),
  depth = rnorm(n),
  temp = rnorm(n)
)
df$count <- rpois(n, exp(1 + 0.2 * df$depth))

# Both scales are sampled by exact NUTS (mode = "exact"); the field is
# returned as gp_local[i] / gp_regional[i] draws.
fit <- tulpa(
  count ~ depth + temp,
  data = df,
  family = "poisson",
```

```
  spatial = spatial_multiscale(
    ~ lon + lat,
    range_local = c(0.1, 0.5),
    range_regional = c(1, 5),
    nn_local = 5L,
    nn_regional = 8L
  ),
  mode = "exact",
  control = list(n_iter = 60L, n_warmup = 30L, seed = 1L)
)
summary(fit)
```

`spatial_range` *Extract spatial range and variance from a fitted spatial model*

Description

Summarises the posterior of spatial hyperparameters. For sampler-tier fits this reads the raw hyperparameter draws; for a nested-Laplace spatial fit it summarises the outer hyperparameter grid. Works with ICAR, BYM2, GP (NNGP), CAR, SPDE, and SVC spatial types.

Usage

```
spatial_range(object, probs = c(0.025, 0.975))
```

Arguments

`object`	A `tulpa_fit` object fitted with a spatial component.
`probs`	Quantile probabilities for the summary (default 0.025, 0.975).

Details

A range is reported in the units of the coordinates as supplied, also when the field was fitted on standardized coordinates (`scale_coords = TRUE`, the default of `spatial_gp()` and `spatial_svc()`). A `mode = "laplace"` fit conditions on the field's hyperparameters rather than inferring them, so it carries no posterior for this function to summarise.

Value

A data.frame with rows for each spatial hyperparameter and columns `mean`, `sd`, and one quantile column per entry of `probs` (named from the probability, e.g. `q2.5`, `q97.5` for the defaults).

Examples

```
set.seed(1)
S <- 12L
adj <- matrix(0, S, S)
for (i in 1:(S - 1)) adj[i, i + 1] <- adj[i + 1, i] <- 1
d <- data.frame(region = factor(rep(1:S, each = 5)), x = rnorm(S * 5))
field <- as.numeric(scale(cumsum(rnorm(S, 0, 0.5))))
d$y <- rpois(nrow(d), exp(0.3 + 0.4 * d$x + field[d$region]))
fit <- tulpa(y ~ x + spatial(region), data = d, family = "poisson",
             spatial = spatial_car(adj, level = "group", group_var = "region"))
spatial_range(fit)
```

---

spatial_rsr *Restricted spatial regression (RSR)*

Description

Apply Restricted Spatial Regression to mitigate spatial confounding. RSR orthogonalizes the spatial effect to the covariate space, preventing the spatial random effect from absorbing covariate information.

This is important when covariates are spatially smooth (e.g., climate variables, elevation) because the spatial random effect can "steal" variance from these covariates, leading to biased coefficient estimates.

Usage

```
spatial_rsr(spatial, restrict_to)

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

Arguments

`spatial`	An areal specification – `spatial_car()`, `spatial_icar()`, `spatial_bym2()` or a proper-CAR spec – or an NNGP one, `spatial_gp()`. The projection is applied by the binomial Polya-Gamma Gibbs sampler and by nested Laplace, both of which carry the field's own prior precision as an adjacency or as Vecchia factors; an HSGP basis (`spatial_gp(approx = "hsgp")`) and an SPDE mesh (`spatial_spde()`) are neither, and are refused at construction rather than accepted and then unfittable.
`restrict_to`	Formula specifying which covariates to orthogonalize against (e.g., `~ depth + temp`). The spatial effect will be constrained to be orthogonal to the column space of these covariates.
`x`	A `tulpa_rsr` specification.
`...`	Passed to the print method of the restricted field.

Details

The RSR approach (Reich et al., 2006; Hodges & Reich, 2010) modifies the spatial random effect to be orthogonal to the fixed effect design matrix:

$$w_{RSR} = (I - P_X)w$$

where $P_X = X(X'X)^{-1}X'$ is the projection matrix onto the column space of X. The projector is built at the FIELD's own resolution: one row per areal unit, or one per unique location for an NNGP field, with the restricted design averaged over the observations at each.

What RSR estimates. The restriction puts no part of the shared, smooth signal in the field, so the fixed effect takes all of it: RSR targets the MARGINAL association between the covariate and the response, where the unrestricted spatial model targets the association conditional on the field. The two differ by exactly the covariate's projection onto the field, so they are different estimands rather than a biased and an unbiased version of one (Bradley, 2024). Measured on a confounded continuous fixture (5 seeds, conditional slope 1.0, marginal 1.71): the restricted fit averaged 0.06 from the marginal value and 0.74 from the conditional one, the unrestricted fit 0.10 and 0.63.

The restriction is not free. In the geostatistical setting Hanks et al. (2015) measured POORER coverage under RSR than under the spatial model that does not restrict, and credible intervals that can be inappropriately narrow under model misspecification; Khan and Calder (2022) report the same on areal structure, with a non-spatial model competitive on coverage and higher Type-S error rates under RSR. Read an RSR interval as an interval for the marginal association under a correctly specified model, and prefer a posterior-predictive check (Hanks et al., 2015) where that is in doubt.

When to use RSR:

- Covariates are spatially smooth (environmental gradients)
- The marginal association is the quantity of interest
- Coefficients appear attenuated toward zero

When NOT to use RSR:

- Covariates are spatially uncorrelated
- Spatial effect is the primary quantity of interest
- Prediction is the main goal
- Interval coverage matters more than the point estimate

A restricted field is fitted by nested Laplace (`mode = "structured"` or `"nested_laplace"`, under any family that path takes, and alongside a random-effect term, a temporal field, smoothers or `latent()` blocks), where the field reaches the linear predictor through the projector `S P_perp` (`S` the observation-to-unit or observation-to-location incidence), or, for a binomial response, by the Polya-Gamma Gibbs sampler (`mode = "gibbs"`). `mode = "auto"` selects Gibbs for a binomial response and nested Laplace otherwise. Areal and NNGP fields take both routes.

Value

A modified spatial specification with RSR enabled

References

Reich, B. J., Hodges, J. S., & Zadnik, V. (2006). Effects of residual smoothing on the posterior of the fixed effects in disease-mapping models. Biometrics, 62(4), 1197-1206.

Hodges, J. S., & Reich, B. J. (2010). Adding spatially-correlated errors can mess up the fixed effect you love. The American Statistician, 64(4), 325-334.

Hanks, E. M., Schliep, E. M., Hooten, M. B., & Hoeting, J. A. (2015). Restricted spatial regression in practice: geostatistical models, confounding, and robustness under model misspecification. Environmetrics, 26(4), 243-254.

Khan, K., & Calder, C. A. (2022). Restricted spatial regression methods: implications for inference. Journal of the American Statistical Association, 117(537), 482-494.

Bradley, J. R. (2024). Restricted spatial regression is reasonable statistical practice: clarifications, interpretations, and new developments. arXiv:2408.05106.

See Also

spatial_gp(), spatial_car()

Examples

```
# Create RSR spatial structure on an areal field
W <- matrix(0, 4, 4)
for (i in 1:3) W[i, i + 1] <- W[i + 1, i] <- 1
rsr <- spatial_rsr(
  spatial_car(W, level = "obs"),
  restrict_to = ~ depth + temp
)
print(rsr)


# Areal binomial data on a chain of regions, covariate spatially confounded
set.seed(404)
n_regions <- 12
W <- matrix(0, n_regions, n_regions)
for (i in 1:(n_regions - 1)) W[i, i + 1] <- W[i + 1, i] <- 1
df <- data.frame(region = factor(rep(1:n_regions, each = 5)))
df$x <- as.integer(df$region) / 4 + rnorm(nrow(df), 0, 0.5)
df$y <- rbinom(nrow(df), 20, plogis(-0.5 + 0.6 * df$x))

# RSR orthogonalises the spatial field to x, protecting its coefficient
fit <- tulpa(
  y ~ x + spatial(region),
  data = df,
  family = "binomial",
  n_trials = rep(20L, nrow(df)),
  spatial = spatial_rsr(spatial_car(W, level = "obs"), restrict_to = ~ x),
  mode = "auto",
  control = list(n_iter = 500L, warmup = 250L)
)
summary(fit)
```

```
# The same modifier on a continuous field: one observation per location,
# the projector built at the unique coordinates.
set.seed(7)
n <- 80
pts <- data.frame(lon = runif(n), lat = runif(n))
pts$x <- as.numeric(scale(pts$lon + pts$lat + rnorm(n, 0, 0.3)))
pts$y <- rbinom(n, 25, plogis(-0.2 + pts$x))
fit_gp <- tulpa(
  y ~ x,
  data = pts,
  family = "binomial",
  n_trials = rep(25L, n),
  spatial = spatial_rsr(spatial_gp(~ lon + lat), restrict_to = ~ x),
  mode = "gibbs",
  control = list(n_iter = 500L, warmup = 250L)
)
summary(fit_gp)
```

`spatial_spde` *SPDE spatial field (Matern via triangular mesh)*

Description

Specify a continuous Matern spatial field using the SPDE approach (Lindgren, Rue & Lindstrom 2011). Builds a triangular mesh from observation coordinates, computes FEM matrices, and passes them to tulpa's SPDE Laplace engine with CHOLMOD sparse solver.

Usage

```
spatial_spde(
  coords,
  data = NULL,
  mesh = NULL,
  boundary = NULL,
  max_edge = NULL,
  cutoff = 0,
  nu = 1,
  prior_range = NULL,
  prior_sigma = NULL
)
```

Arguments

`coords`	A formula `~ x + y` or a two-column matrix of coordinates.
`data`	Optional data.frame for formula evaluation.

mesh	A pre-built tulpa_mesh object. If NULL (default), a mesh is built automatically from coords.
boundary	Optional boundary: a two-column matrix of polygon vertices, an sf polygon, or NULL for convex hull with extension.
max_edge	Maximum edge length for mesh refinement. A single value or c(inner, outer).
cutoff	Minimum distance between mesh vertices. Default 0.
nu	Matern smoothness parameter. A positive number. Integer nu (1, 2, 3, ...) gives an exact FEM construction (operator order alpha = nu + 1). Fractional nu (e.g. 0.5, 1.5) uses the operator-based rational SPDE approximation with BRASIL best-rational coefficients (Bolin & Kirchner 2020; Hofreither 2021); supported by the Laplace fitter fit_spde() (single-point and nested over range/sigma). NUTS and analytic marginal SEs remain integer-only. Default 1.
prior_range	PC prior on the spatial range, c(U, alpha) with P(range < U) = alpha. NULL (the default) anchors it on the data: U is a fifth of the diagonal of the coordinates' bounding box and alpha = 0.5, so U is the prior median.
prior_sigma	PC prior on the marginal standard deviation, c(U, alpha) with P(sigma > U) = alpha. NULL (the default) is c(3, 0.01), the engine's prior on every field scale.

Value

A tulpa_spatial object with type "spde".

Examples

```
set.seed(42)
coords <- cbind(runif(50), runif(50))
spec <- spatial_spde(coords)
print(spec)
```

---

spatial_spde_custom *SPDE spatial field from custom matrices*

Description

Specify a continuous Matern spatial field using externally-provided FEM matrices. Use this with meshes from fmesher, rSPDE, or any other source.

Usage

```
spatial_spde_custom(
  C,
  G,
  A,
  nu = 1,
  prior_range = NULL,
  prior_sigma = NULL,
  coords = NULL
)
```

Arguments

C	Mass matrix (n_mesh x n_mesh sparse matrix, e.g. from `fmesher::fm_fem()$c0`).
G	Stiffness matrix (n_mesh x n_mesh sparse matrix, e.g. from `fmesher::fm_fem()$g1`).
A	Projection matrix (n_obs x n_mesh sparse matrix, e.g. from `fmesher::fm_basis()`).
nu	Matern smoothness parameter. A positive number; integer values give the exact FEM construction, fractional values the BRASIL rational SPDE approximation (supported by `fit_spde()`;). Default 1.
prior_range	PC prior on the spatial range, `c(U, alpha)` with P(range < U) = alpha. `NULL` (the default) anchors it on `coords` as `spatial_spde()` does, and needs them.
prior_sigma	PC prior on the marginal standard deviation, `c(U, alpha)` with P(sigma > U) = alpha. `NULL` (the default) is `c(3, 0.01)`.
coords	Observation coordinates (an `n_obs x 2` matrix), read only to anchor the default range prior. Not needed when `prior_range` is given.

Value

A `tulpa_spatial` object with type `"spde"`.

Examples

```
# FEM matrices of a 6-node 1-D chain mesh, observed at the nodes:
n <- 6L
C <- Matrix::Diagonal(n)
G <- Matrix::bandSparse(n, k = c(-1, 0, 1),
                        diagonals = list(rep(-1, n - 1), c(1, rep(2, n - 2), 1),
                                         rep(-1, n - 1)))
A <- Matrix::Diagonal(n)
sp <- spatial_spde_custom(C, G, A, prior_range = c(1, 0.5),
                          prior_sigma = c(1, 0.01))
sp$type
```

---

spatial_svc *Spatially varying coefficient structure*

Description

Specify a spatially varying coefficient (SVC): one or more fixed-effect coefficients are allowed to vary smoothly over space, with the variation governed by a Gaussian process (NNGP or HSGP approximation).

Usage

```
spatial_svc(
  coords,
  terms = 1,
  cov = c("exponential", "matern"),
  nn = 15,
  shared = NULL,
  scale_coords = TRUE,
  approx = c("nngp", "hsgp"),
  m = 6,
  c_boundary = 1.5,
  parameterization = c("noncentered", "centered")
)

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

Arguments

`coords` A formula (`~ lon + lat`) or character vector of length 2 naming the two coordinate variables in the data.

`terms` Which coefficients vary over space. A formula, an integer vector of design-matrix column indices, or a character vector of term names. Default `1` (the intercept).

`cov` Covariance function. One of `"exponential"` or `"matern"`.

`nn` Number of nearest neighbours used in the NNGP approximation (`approx = "nngp"`).

`shared` Whether the effect is shared across processes in a multi-process model. `NULL` (default) shares it; `FALSE` fits process-specific effects and emits a warning.

`scale_coords` Logical. Standardize coordinates before fitting (default `TRUE`), by one common factor as in `spatial_gp()`. Ranges and the coordinates `svc()` reports are in the original units either way.

`approx` Spatial approximation. `"nngp"` (nearest-neighbour GP) or `"hsgp"` (Hilbert-space GP).

`m` Number of basis functions per dimension for the HSGP approximation (`approx = "hsgp"`).

`c_boundary` Boundary-extension factor for the HSGP domain (`approx = "hsgp"`).

`parameterization` Latent parameterization for the exact-NUTS field (`approx = "nngp"` only; HSGP is already non-centered by construction). `"noncentered"` (default) samples `z_j ~ N(0, I)` per term and reconstructs each field as `w_j = f(z_j, sigma2_j, phi_j)`, removing the field/hyperparameter funnel that otherwise attenuates the field's amplitude when it is weakly identified. `"centered"` places the NNGP density on each term's field directly; it is marginally cheaper on a well-identified response, but on a weakly identified one it recovers only about a third of the field's spread.

x	A tulpa_svc specification.
...	Ignored.

Value

A tulpa_svc object (also of class tulpa_spatial).

See Also

spatial_gp() for a spatial random effect (rather than a varying coefficient).

Examples

```
# Intercept that varies smoothly over space
spatial_svc(~ lon + lat)
```

---

spatiotemporal *Spatiotemporal interaction specifications for tulpa*

Description

Functions to specify spatiotemporal interaction effects for tulpa models. These capture dependencies that arise when spatial patterns vary over time, or when temporal trends differ across space.

Specify a spatiotemporal interaction effect for tulpa models. The interaction captures structured or unstructured deviation from the additive spatial + temporal model.

No tulpa backend fits an interaction term, so this constructor errors. The additive space-time model is fitted by tulpa(spatial = , temporal =) and by fit_st_nested().

Usage

```
spatiotemporal(
  spatial,
  temporal,
  type = c("I", "II", "III", "IV", "iid", "separable"),
  shared = NULL
)
```

Arguments

spatial	A spatial specification from spatial_car(), spatial_bym2(), or spatial_gp().
temporal	A temporal specification from temporal_rw1(), temporal_rw2(), temporal_ar1(), or temporal_gp().
type	Interaction type: • "I" or "iid": Unstructured interaction (IID) • "II": Structured time at each location

- `"III"`: Structured space at each time point
- `"IV"`: Fully structured (Kronecker product of spatial and temporal)
- `"separable"`: Separable covariance (Kronecker product)

`shared` Logical; if TRUE (default), spatiotemporal effect enters both all processes. Set to FALSE for process-specific effects (triggers warning about potential confounding).

Details

Spatiotemporal interactions extend the basic additive model:

$$\eta_{st} = X\beta + f_s(space) + f_t(time)$$

to include interactions:

$$\eta_{st} = X\beta + f_s(space) + f_t(time) + \delta_{st}$$

where δ_{st} captures space-time interactions.

Interaction Types (following Knorr-Held, 2000):

- **Type I**: Unstructured interaction - IID $\delta_{st} \sim N(0, \sigma^2)$
- **Type II**: Structured time, unstructured space - temporal structure at each location
- **Type III**: Structured space, unstructured time - spatial structure at each time
- **Type IV**: Structured space AND time - full Kronecker interaction

Separable Models:

- **Separable**: Covariance is Kronecker product $C_{st} = C_s \otimes C_t$
- **Non-separable**: GP with joint space-time metric

Type I (IID)

Independent random effect for each space-time combination:

$$\delta_{st} \overset{iid}{\sim} N(0, \sigma_\delta^2)$$

This is the simplest form, requiring S*T parameters but capturing no structured interaction.

Type II (Temporal structure per location)

Each location has its own temporal random effect:

$$\delta_{\cdot t}^{(s)} \sim RW(\sigma^2)$$

This captures location-specific temporal trends but assumes independence across locations.

Type III (Spatial structure per time point)

Each time point has its own spatial random effect:

$$\delta_{s\cdot}^{(t)} \sim ICAR(\tau)$$

This captures time-specific spatial patterns but assumes independence across time points.

Type IV (Full structure)

Kronecker product of spatial and temporal precision matrices:

$$Q_\delta = Q_s \otimes Q_t$$

This is the most constrained model, assuming the interaction has the same structure as the marginal effects.

Separable

For GP-based effects, assumes separable covariance:

$$C(\mathbf{s}_1, t_1; \mathbf{s}_2, t_2) = C_s(\mathbf{s}_1, \mathbf{s}_2) \cdot C_t(t_1, t_2)$$

Value

Nothing: the call always signals an error.

References

Knorr-Held, L. (2000). Bayesian modelling of inseparable space-time variation in disease risk. Statistics in Medicine, 19(17-18), 2555-2567.

See Also

spatial_car(), spatial_gp(), temporal_rw1(), temporal_ar1()

Examples

```
adj <- matrix(0, 4, 4)
for (i in 1:3) adj[i, i + 1] <- adj[i + 1, i] <- 1
# No backend fits an interaction term, so the constructor refuses; the
# additive model is tulpa(spatial = , temporal = ).
try(spatiotemporal(spatial_car(adj, group_var = "region"),
                   temporal_rw1("year"), type = "IV"))
```

---

spatiotemporal_effects

Extract spatiotemporal effects from fitted model

Description

Extract posterior distributions of spatiotemporal interaction effects from a fitted tulpa model.

Usage

```
spatiotemporal_effects(
  object,
  format = c("array", "long", "summary"),
  probs = c(0.025, 0.5, 0.975),
  ...
)

## Default S3 method:
spatiotemporal_effects(
  object,
  format = c("array", "long", "summary"),
  probs = c(0.025, 0.5, 0.975),
  ...
)

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

Arguments

object	A fit carrying a `$spatiotemporal` Knorr-Held interaction block (`n_spatial`, `n_times`) and its draws in `$.internal$spatiotemporal_draws`. `tulpa()` itself has no `spatiotemporal =` argument; such a block comes from a model package's own fitter (see `spatiotemporal()` for what the engine itself fits), and the default method reads any fit, of any class, that stores it this way.
format	Output format: `"array"` (default, S x T x draws), `"long"` (data frame with s, t, draw, value columns), or `"summary"` (posterior summaries).
probs	Quantiles to compute if `format = "summary"`.
...	Ignored
x	A `tulpa_st_summary` table.

Value

For `format = "array"`, an S x T x draws numeric array carrying `n_spatial`, `n_times` and `n_draws` attributes; for `"long"`, a data frame with columns `s`, `t`, `draw` and `value`; for `"summary"`, a `tulpa_st_summary` data frame with columns `s`, `t`, `mean`, one column per entry of `probs` and `sd`, which has `print()` and `plot()` methods.

Examples

```
# A Knorr-Held interaction block is attached by a model package's own
# fitter (see spatiotemporal() for what the engine fits). The accessor reads
# its layout and an n_draws x (S * T) draw matrix stored s-fastest: column
# (t - 1) * S + s holds region s at time t. Here both are simulated for
# S = 3 regions over T = 4 times.
set.seed(1)
S <- 3L; n_t <- 4L
```

```
st_mean <- outer(c(-0.5, 0, 0.5), seq(-0.3, 0.3, length.out = n_t))
draws <- matrix(rnorm(100 * S * n_t, mean = rep(c(st_mean), each = 100),
                      sd = 0.1), nrow = 100)
fit <- structure(
  list(spatiotemporal = list(n_spatial = S, n_times = n_t),
       .internal = list(spatiotemporal_draws = draws)),
  class = "tulpa_fit")

st_effects <- spatiotemporal_effects(fit, format = "summary")
head(st_effects)
dim(spatiotemporal_effects(fit, format = "array"))
```

---

spatiotemporal_gp *Non-separable spatiotemporal GP*

Description

Specify a non-separable Gaussian Process for spatiotemporal effects. Unlike separable models where the covariance factors as $C_s \otimes C_t$, non-separable models allow for direct space-time interaction in the covariance.

No tulpa backend fits a joint space-time covariance, so this constructor errors. A spatial GP alongside a temporal field is fitted directly by `fit_st_nested()` (`spatial_type = "hsgp"` or `"nngp"`); `tulpa()`'s formula front door does not yet route a continuous spatial field together with a temporal field (it does for an areal one – icar/bym2/car_proper – via `tulpa(spatial = , temporal = )`; gcol33/tulpa#812).

Usage

```
spatiotemporal_gp(
  coords,
  time_var,
  cov_space = c("exponential", "matern", "gaussian", "spherical"),
  cov_time = c("exponential", "matern", "gaussian"),
  nonsep_type = c("product", "sum", "gneiting", "cressie_huang"),
  nn = 15,
  shared = NULL
)
```

Arguments

`coords`	A one-sided formula specifying coordinate columns (e.g., `~ lon + lat`), or a character vector of length 2.
`time_var`	Name of the time variable in data.
`cov_space`	Spatial covariance: `"exponential"` (default), `"matern"`, `"gaussian"`, or `"spherical"`.
`cov_time`	Temporal covariance: `"exponential"` (default), `"matern"`, or `"gaussian"`.
`nonsep_type`	Non-separability type:

- "product": $C_{st} = C_s \cdot C_t$ (separable, for reference)
- "sum": $C_{st} = C_s + C_t$
- "gneiting": Gneiting (2002) non-separable class
- "cressie_huang": Cressie-Huang (1999) non-separable class

nn — Number of nearest neighbors for NNGP approximation. Default 15.

shared — Logical; if TRUE (default), effect enters both processes.

Details

The non-separable covariance functions allow for more flexible space-time dependence:

Gneiting class:

$$C(h,u) = \frac{\sigma^2}{(a|u|^{2\alpha}+1)^{\tau}} \exp\left(-\frac{c\|h\|^{2\gamma}}{(a|u|^{2\alpha}+1)^{\beta\gamma}}\right)$$

where h is spatial lag, u is temporal lag, and parameters control the space-time interaction.

Cressie-Huang class: Constructed via Fourier transform methods to ensure positive definiteness.

Value

Nothing: the call always signals an error.

References

Gneiting, T. (2002). Nonseparable, stationary covariance functions for space-time data. Journal of the American Statistical Association, 97(458), 590-600.

Cressie, N., & Huang, H. C. (1999). Classes of nonseparable, spatio-temporal stationary covariance functions. Journal of the American Statistical Association, 94(448), 1330-1340.

Examples

```
# No backend fits a joint space-time covariance, so the constructor refuses:
try(spatiotemporal_gp(~ lon + lat, "year", nonsep_type = "gneiting"))
```

---

svc — *Extract spatially-varying coefficients from a fitted model*

Description

Extract posterior distributions of spatially-varying coefficients (SVCs) from a fitted tulpa model with SVC specification.

Usage

```
svc(object, terms = NULL, summary = FALSE, probs = c(0.025, 0.5, 0.975), ...)

## S3 method for class 'tulpa_fit'
svc(object, terms = NULL, summary = FALSE, probs = c(0.025, 0.5, 0.975), ...)

## S3 method for class 'tulpa_svc_posterior'
summary(object, probs = c(0.025, 0.5, 0.975), ...)

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

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

Arguments

`object`	A `tulpa_fit` object fitted with `svc` argument
`terms`	Which SVC terms to extract. If NULL (default), extracts all.
`summary`	Logical; if TRUE, return summary statistics instead of full posterior draws.
`probs`	Quantiles to compute if `summary = TRUE`.
`...`	Ignored
`x`	A `tulpa_svc_posterior` or `tulpa_svc_summary` object.

Value

A `tulpa_svc_posterior` object containing:

- `draws`: Array of posterior draws (draws x locations x terms)
- `coords`: Coordinate matrix
- `term_names`: Names of SVC terms

`summary()` on a `tulpa_svc_posterior` returns a `tulpa_svc_summary` data frame with one row per location and term, holding the observation index, term name, coordinates, and the posterior mean, SD and requested quantiles of each spatially varying coefficient.

See Also

`spatial_svc()`, `plot.tulpa_svc_posterior()`

Examples

```
set.seed(303)
n <- 25L
df <- data.frame(lon = runif(n), lat = runif(n), x = rnorm(n))
bsurf <- 0.9 * sin(2.8 * df$lon) + 0.7 * cos(2.2 * df$lat)
df$count <- rpois(n, exp(0.2 + (0.8 + bsurf) * df$x))
```

```
# The varying slope on `x` is a spatial field; SVC is exact-mode only.
fit <- tulpa(
  count ~ x,
  data = df,
  family = "poisson",
  spatial = spatial_svc(~ lon + lat, terms = ~ x - 1, nn = 5L),
  mode = "exact",
  control = list(n_iter = 80L, n_warmup = 40L, seed = 1L)
)

svc_post <- svc(fit)
summary(svc_post)
```

`temporal_ar` *AR(p) temporal latent field (general-order autoregressive)*

Description

A stationary autoregressive process of order `p`, $w_t = \sum_{j=1}^{p} \phi_j w_{t-j} + \varepsilon_t$, as a user-defined GMRF latent block for a model formula via `latent(temporal_ar(...))` – the general-order extension of `temporal_ar2()`, sharing the same construction: the precision is the exact stationary AR(p) GMRF (the banded inverse of the Yule-Walker Toeplitz autocovariance), and stationarity is enforced by the partial-autocorrelation parameterization (Levinson-Durbin map of `p` unconstrained `atanh_psi` hyperparameters), so the hyperparameter integration never leaves the stationary region.

The block's hyperparameters are `log_tau` (log innovation precision) and `atanh_psi1..atanh_psi<p>`; the AR coefficients are recovered through `psi_j = tanh(atanh_psi_j)` and the Levinson-Durbin recursion. Note the outer integration cost grows with the hyperparameter count `p + 1`; orders beyond 3-4 are better served by a sampler tier.

Usage

```
temporal_ar(
  time_idx,
  p = 2L,
  n_times = NULL,
  prior_tau_sd = 2,
  prior_psi_sd = 1.5,
  name = NULL
)

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

Arguments

`time_idx`	Integer vector (1-based) of the time point for each observation.
`p`	Autoregressive order (>= 1).
`n_times`	Number of distinct time points; defaults to `max(time_idx)`.
`prior_tau_sd, prior_psi_sd`	Prior SDs for `log_tau` and the `atanh_psi` hyperparameters (weakly-informative Gaussian). Defaults 2 / 1.5.
`name`	Optional block label (default `ar<p>`).
`x`	A temporal AR(p) block.
`...`	Passed to the `tgmrf` print method.

Value

A `tgmrf` / `tulpa_latent_block` object.

See Also

`temporal_ar2()` (the `p = 2` shorthand), `temporal_ar1()` for the first-order formula-integrated process, `tgmrf()` for the general user-defined GMRF interface.

Examples

```
set.seed(1)
Tt <- 150L
w <- numeric(Tt); w[1:3] <- rnorm(3)
for (t in 4:Tt) w[t] <- 0.4 * w[t-1] + 0.2 * w[t-2] - 0.2 * w[t-3] + rnorm(1, 0, 0.4)
d <- data.frame(t = seq_len(Tt), y = w + rnorm(Tt, 0, 0.3))
fit <- tulpa(y ~ latent(temporal_ar(d$t, p = 3)), data = d,
             family = "gaussian", mode = "nested_laplace")
```

`temporal_ar1` *AR1 temporal structure (first-order autoregressive)*

Description

Specify a first-order autoregressive temporal random effect. AR1 models temporal correlation where each time point depends on the previous one: `phi[t] = rho * phi[t-1] + epsilon[t]`.

Unlike RW1/RW2, AR1 is a proper (stationary) model with an estimated autocorrelation parameter rho.

Usage

```
temporal_ar1(time_var, group_var = NULL, shared = NULL, rho_prior = NULL)
```

Arguments

`time_var`	A formula (`~ time`) or single character string naming the time variable in the data.
`group_var`	Optional formula (`~ g`) or character string naming a grouping variable. When supplied, a separate random walk is fit per group; `NULL` (default) fits a single walk shared across all observations.
`shared`	Whether the temporal effect is shared across processes in a multi-process model. `NULL` (default) shares the effect; `FALSE` fits process-specific effects and emits a warning about unshared confounding.
`rho_prior`	Prior for the AR(1) autocorrelation. Default `NULL` is a Uniform(-1, 1) prior. Supply a `prior_beta(alpha, beta)` to place a Beta(alpha, beta) prior on `u = (rho + 1)/2` for a more informative prior (e.g. `prior_beta(5, 2)` favours positive autocorrelation). Honoured on both the exact-sampler and nested-Laplace paths.

Details

The AR1 process has marginal variance sigma^2 / (1 - rho^2) and correlation between time points t and s of rho^|t-s|.

The precision matrix is tridiagonal and full rank, so no constraints are needed.

Value

A `tulpa_temporal` object

Examples

```
# Create temporal AR1 specification
temporal_ar1("year")
temporal_ar1("year", group = "site")


# AR1 temporal correlation
set.seed(127)
df <- data.frame(
  year = rep(1:20, each = 3),
  x = rnorm(60)
)
f <- as.numeric(arima.sim(list(ar = 0.7), 20, sd = 0.4))
df$count <- rpois(60, exp(1 + 0.3 * df$x + f[df$year]))

fit <- tulpa(
  count ~ x,
  data = df,
  family = "poisson",
  temporal = temporal_ar1("year"),
  mode = "auto"
)
summary(fit)
```

temporal_ar2 *AR(2) temporal latent field (second-order autoregressive)*

Description

A stationary second-order autoregressive process $w_t = \phi_1 w_{t-1} + \phi_2 w_{t-2} + \varepsilon_t$ as a user-defined GMRF latent block, for use in a model formula via `latent(temporal_ar2(...))`. It reuses tulpa's nested-Laplace / NUTS machinery (no dedicated C++ kernel). The precision is the exact stationary AR(2) GMRF; stationarity is enforced by the PACF parameterization, so the hyperparameter grid never leaves the stationarity region.

The block's hyperparameters are `log_tau` (log innovation precision) and `atanh_psi1`, `atanh_psi2` (unconstrained partial autocorrelations); the AR coefficients are recovered as `phi2 = tanh(atanh_psi2)`, `phi1 = tanh(atanh_psi1) (1 - phi2)`.

Usage

```
temporal_ar2(
  time_idx,
  n_times = NULL,
  prior_tau_sd = 2,
  prior_psi_sd = 1.5,
  name = "ar2"
)
```

Arguments

`time_idx`	Integer vector (1-based) of the time point for each observation.
`n_times`	Number of distinct time points; defaults to `max(time_idx)`.
`prior_tau_sd, prior_psi_sd`	Prior SDs for `log_tau` and the two `atanh_psi` hyperparameters (weakly-informative Gaussian). Defaults 2 / 1.5.
`name`	Optional block label.

Value

A `tgmrf` / `tulpa_latent_block` object.

See Also

`temporal_ar1()` for the first-order (formula-integrated) process; `tgmrf()` for the general user-defined GMRF interface.

Examples

```
set.seed(1)
Tt <- 120L
w <- numeric(Tt); w[1:2] <- rnorm(2)
for (t in 3:Tt) w[t] <- 0.5 * w[t-1] + 0.3 * w[t-2] + rnorm(1, 0, 0.4)
d <- data.frame(t = seq_len(Tt), y = w + rnorm(Tt, 0, 0.3))
# `phi` is the gaussian residual VARIANCE, here the 0.3 the series was
# simulated with. Left unsupplied the fit conditions on 1 and says so.
fit <- tulpa(y ~ latent(temporal_ar2(d$t)), data = d, family = "gaussian",
             phi = 0.3^2, mode = "nested_laplace")
```

---

temporal_corr *Extract temporal correlation parameters from a fitted model*

Description

Returns posterior summary for temporal hyperparameters (tau, rho for AR1, sigma/lengthscale for temporal GP).

Usage

```
temporal_corr(object, probs = c(0.025, 0.975))
```

Arguments

`object`	A `tulpa_fit` object fitted with a temporal component.
`probs`	Quantile probabilities (default 0.025, 0.975).

Value

A data.frame with rows for each temporal hyperparameter.

Examples

```
set.seed(127)
df <- data.frame(year = rep(1:20, each = 3), x = rnorm(60))
f <- as.numeric(arima.sim(list(ar = 0.7), 20, sd = 0.4))
df$count <- rpois(60, exp(1 + 0.3 * df$x + f[df$year]))
fit <- tulpa(count ~ x, data = df, family = "poisson",
             temporal = temporal_ar1("year"))
temporal_corr(fit)
```

temporal_gp *Gaussian process temporal structure*

Description

Specify a Gaussian Process (GP) temporal random effect for irregularly-spaced or continuous time points. Unlike RW1/RW2/AR1 which assume equally-spaced observations, GP temporal effects model correlation as a function of time distance.

This is particularly useful for:

- Irregularly-spaced time series
- Continuous time (e.g., exact timestamps)
- Smooth temporal trends with uncertainty

Usage

```
temporal_gp(
  time_var,
  cov = c("exponential", "matern", "gaussian", "periodic"),
  nu = 1.5,
  period = NULL,
  group_var = NULL,
  shared = NULL,
  scale_coords = TRUE,
  parameterization = c("noncentered", "centered")
)

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

Arguments

`time_var` Name of the time variable in data. Can be a formula (e.g., ~ `year`) or a character string (e.g., `"year"`). Should be numeric (continuous time) or convertible to numeric.

`cov` Covariance function: `"exponential"` (default, rough), `"matern"` (tunable smoothness), `"gaussian"` (very smooth), or `"periodic"` (for seasonal patterns).

`nu` Smoothness parameter for Matern covariance, one of:

- 0.5: equivalent to exponential (rough)
- 1.5: once differentiable (moderate smoothness)
- 2.5: twice differentiable (smooth) These are the smoothnesses with a closed form; anything between them needs a Bessel function of the second kind and is rejected. Ignored for non-Matern covariance functions.

`period` Period for periodic covariance (e.g., 12 for monthly, 365 for daily data with annual cycle). Only used when `cov = "periodic"`.

`group_var` Optional name of grouping variable for panel data. If provided, separate GPs are estimated for each group.

`shared` Logical; if TRUE (default), temporal effect enters both all processes.

`scale_coords` Logical; if TRUE (default), time values are scaled to unit variance before computing distances. Either way the lengthscale's support is placed relative to the spread of the times, and `temporal_corr()` reports the lengthscale in the original time units.

`parameterization` Parameterization for GP effects: `"noncentered"` (default) stores z ~ N(0,1) and scales by covariance (better for weakly-informed effects); `"centered"` stores effects directly (better for strongly-informed effects).

`x` A `tulpa_temporal_gp` specification.

`...` Ignored.

Details

The GP temporal model adds a time-correlated random effect:

$$\eta(t) = X\beta + f(t)$$

where $f(t)$ follows a Gaussian process:

$$f(t) \sim GP(0, \sigma^2 C(|t - t'|; \phi))$$

The correlation function $C(d; \phi)$ depends on time distance d:

- **Exponential**: $C(d) = \exp(-d/\phi)$ - continuous but not differentiable
- **Matern**: Smooth with tunable roughness via ν
- **Gaussian**: $C(d) = \exp(-(d/\phi)^2)$ - infinitely differentiable
- **Periodic**: $C(d) = \exp(-2\sin^2(\pi d/p)/\phi^2)$ - for seasonal data

Implementation: the exponential kernel (equivalently Matern with `nu = 0.5`) is an Ornstein-Uhlenbeck process, so its joint density factorizes into a first-order Markov chain and evaluates in O(T) with no matrix. The other kernels have no finite-dimensional state-space form and are evaluated from a dense T x T Cholesky, where T is the number of distinct time points.

The field is fit on the sampler path: its two hyperparameters (`log_sigma2_temporal_gp`, `logit_phi_temporal_gp`) are sampled jointly with the field and the fixed effects, so pass a sampler `mode` such as `mode = "hmc"`. There is no nested-Laplace kernel for it.

Value

A `tulpa_temporal_gp` object

See Also

`temporal_rw1()`, `temporal_ar1()` for equally-spaced temporal effects, `spatial_gp()` for spatial GP effects

Examples

```
# Create GP temporal specification
temporal_gp("timestamp")
temporal_gp("day", cov = "matern", nu = 1.5)
temporal_gp("month", cov = "periodic", period = 12)


# Irregularly-spaced time series
set.seed(140)
times <- sort(runif(40, 0, 10))
df <- data.frame(
  time = times,
  x = rnorm(40),
  y = rbinom(40, 1, 0.5)
)

fit <- tulpa(
  y ~ x,
  data = df,
  family = "binomial",
  temporal = temporal_gp("time"),
  mode = "hmc",
  control = list(n_iter = 200, warmup = 100, n_chains = 1)
)
```

---

temporal_multiscale *Multi-scale temporal structure*

Description

Specify a temporal random effect that decomposes variation into separate scales: a smooth trend, an optional seasonal cycle, and a short-term component. Each scale uses its own prior, letting slow and fast dynamics be modelled jointly.

Usage

```
temporal_multiscale(
  time_var,
  trend = c("rw2", "rw1", "none"),
  seasonal = NULL,
  short_term = c("ar1", "iid", "none"),
  group_var = NULL,
  shared = NULL
)

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

Arguments

time_var	Single character string naming the time variable in the data.
trend	Prior for the smooth long-term trend. One of "rw2", "rw1", or "none".
seasonal	Optional integer period (>= 2) of a seasonal cycle, e.g. 12 for monthly data with an annual cycle. NULL (default) omits the seasonal component.
short_term	Prior for the short-term component. One of "ar1", "iid", or "none".
group_var	Optional character string naming a grouping variable for group-specific temporal effects.
shared	Whether the effect is shared across processes in a multi-process model. NULL (default) shares it; FALSE fits process-specific effects and emits a warning.
x	A tulpa_temporal_multiscale specification.
...	Ignored.

Details

At least one of trend, seasonal, or short_term must be active.

Value

A tulpa_temporal_multiscale object.

See Also

temporal_rw1(), temporal_rw2(), temporal_ar1() for single-scale temporal priors.

Examples

```
# Trend + annual seasonal cycle + AR1 short-term component on monthly data
temporal_multiscale("month", trend = "rw2", seasonal = 12, short_term = "ar1")
```

---

temporal_rtr *Restricted temporal regression (RTR)*

Description

The temporal analogue of spatial_rsr(): constrain a temporal random effect to be orthogonal to a set of covariates, so a temporally smooth covariate does not have its fixed-effect coefficient attenuated by a confounded temporal field, $u_{RTR} = (I - P_X)u$.

No tulpa backend applies that projection to a temporal field, so this constructor errors rather than returning a specification that would fit as the unrestricted temporal model. spatial_rsr() is the wired spatial analogue.

Usage

```
temporal_rtr(temporal, restrict_to)
```

Arguments

`temporal`	A `tulpa_temporal` specification (e.g. `temporal_rw1()`, `temporal_ar1()`).
`restrict_to`	A one-sided formula giving the covariate space the temporal effect is made orthogonal to, e.g. `~ x`.

Value

Nothing: the call always signals an error.

See Also

`spatial_rsr()`, `temporal_rw1()`, `temporal_ar1()`

Examples

```
# No backend fits a restricted temporal field, so the constructor refuses:
try(temporal_rtr(temporal_rw1("year"), ~ x))
```

---

temporal_rw1 — *RW1 temporal structure (first-order random walk)*

Description

Specify a first-order random walk temporal random effect. RW1 penalizes first differences, so adjacent time points are smoothed toward each other: `phi[t] - phi[t-1] ~ N(0, sigma^2)`.

Usage

```
temporal_rw1(time_var, group_var = NULL, cyclic = FALSE, shared = NULL)
```

Arguments

`time_var`	A formula (`~ time`) or single character string naming the time variable in the data.
`group_var`	Optional formula (`~ g`) or character string naming a grouping variable. When supplied, a separate random walk is fit per group; `NULL` (default) fits a single walk shared across all observations.
`cyclic`	Logical. If `TRUE`, the random walk wraps around so the last time point is a neighbour of the first (cyclic boundary, e.g. month of year). Default `FALSE`.
`shared`	Whether the temporal effect is shared across processes in a multi-process model. `NULL` (default) shares the effect; `FALSE` fits process-specific effects and emits a warning about unshared confounding.

Details

The precision matrix is rank `T - 1` (one constraint needed). RW1 is the least smooth of the random-walk priors; for smoother trends see `temporal_rw2()`, and for a stationary alternative see `temporal_ar1()`.

Value

A `tulpa_temporal` object.

See Also

`temporal_rw2()`, `temporal_ar1()` for other temporal priors.

Examples

```
# Create temporal RW1 specification
temporal_rw1("year")
temporal_rw1("month", cyclic = TRUE)
```

---

temporal_rw2 — *RW2 temporal structure (second-order random walk)*

Description

Specify a second-order random walk temporal random effect. RW2 penalizes deviations from linearity, resulting in smoother trends than RW1: `phi[t] - 2*phi[t-1] + phi[t-2] ~ N(0, sigma^2)`.

Usage

```
temporal_rw2(time_var, group_var = NULL, cyclic = FALSE, shared = NULL)
```

Arguments

`time_var`	A formula (`~ time`) or single character string naming the time variable in the data.
`group_var`	Optional formula (`~ g`) or character string naming a grouping variable. When supplied, a separate random walk is fit per group; `NULL` (default) fits a single walk shared across all observations.
`cyclic`	Logical. If `TRUE`, the random walk wraps around so the last time point is a neighbour of the first (cyclic boundary, e.g. month of year). Default `FALSE`.
`shared`	Whether the temporal effect is shared across processes in a multi-process model. `NULL` (default) shares the effect; `FALSE` fits process-specific effects and emits a warning about unshared confounding.

Details

RW2 produces smoother curves than RW1 because it penalizes the second derivative (curvature) rather than the first derivative (slope). It requires at least 3 time points.

The precision matrix is rank T-2 (two constraints needed).

Value

A `tulpa_temporal` object

Examples

```
# Create temporal RW2 specification
temporal_rw2("year")


# Smooth temporal trend
set.seed(126)
df <- data.frame(
  year = rep(1:20, each = 4),
  x = rnorm(80)
)
trend <- sin(seq(0, 2, length.out = 20))
df$count <- rpois(80, exp(1 + 0.3 * df$x + trend[df$year]))

fit <- tulpa(
  count ~ x,
  data = df,
  family = "poisson",
  temporal = temporal_rw2("year"),
  mode = "auto"
)
summary(fit)
```

---

temporal_tvc *Time-varying coefficient structure*

Description

Specify a time-varying coefficient (TVC): one or more fixed-effect coefficients are allowed to evolve over time, with the evolution governed by a temporal prior (`rw1`, `rw2`, `ar1` or `gp`).

`rw1`, `rw2` and `ar1` read the time index as a position on a grid, so consecutive instants are one step apart whatever the data says. `gp` is the continuous-time structure: the coefficient is a Gaussian process over the distinct time VALUES, which is what irregular spacing needs. It is distinct from `temporal_gp()`, a GP over time entering the linear predictor additively (`eta_i += f(t_i)`); here the GP IS a coefficient (`eta_i += x_i w(t_i)`).

Usage

```
temporal_tvc(
  time_var,
  terms = 1,
  structure = c("rw1", "rw2", "ar1", "gp"),
```

```
  cov = c("exponential", "matern", "gaussian", "periodic"),
  nu = 1.5,
  period = NULL,
  group_var = NULL,
  shared = NULL,
  sigma_prior_U = 1,
  sigma_prior_alpha = 0.01,
  scale_coords = TRUE
)

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

Arguments

`time_var` Single character string naming the time variable in the data. `structure = "gp"` needs it numeric: a factor states an ordering with no spacing for the kernel to measure.

`terms` Which coefficients vary over time. A formula, an integer vector of design-matrix column indices, or a character vector of term names. Default `1` (the intercept).

`structure` Temporal prior governing how the coefficients evolve. One of `"rw1"`, `"rw2"`, `"ar1"` or `"gp"`.

`cov, nu, period` Covariance kernel for `structure = "gp"`, ignored otherwise. `cov` is one of `"exponential"` (the default), `"matern"`, `"gaussian"` or `"periodic"`; `nu` is the Matern smoothness, closed-form at `0.5`, `1.5` and `2.5` only (`0.5` IS the exponential kernel); `period` is the periodic kernel's period. Exponential and Matern `nu = 0.5` evaluate in `O(T)` through the Ornstein-Uhlenbeck factorization; the rest take a dense `T x T` Cholesky per coefficient per gradient evaluation.

`group_var` Optional character string naming a grouping variable for group-specific time-varying coefficients.

`shared` Whether the effect is shared across processes in a multi-process model. `NULL` (default) shares it; `FALSE` fits process-specific effects and emits a warning.

`sigma_prior_U, sigma_prior_alpha` Penalized-complexity prior on each varying coefficient's marginal standard deviation, calibrated so that `P(sigma > sigma_prior_U) = sigma_prior_alpha`. Defaults to `P(sigma > 1) = 0.01`. `sigma_prior_U` must be positive and `sigma_prior_alpha` must lie in `(0, 1)`. Read on every structure: it is the same anchor pair whether the field samples a log-precision (`rw1` / `rw2` / `ar1`) or a log-variance (`gp`).

`scale_coords` Logical, `structure = "gp"` only: standardize the time values before fitting (default `TRUE`), which puts the lengthscale on the same universal support `temporal_gp()` uses. `period` is stated in the raw time units and makes the same trip.

`x` A `tulpa_tvc` specification.

`...` Ignored.

Value

A `tulpa_tvc` object.

See Also

temporal_rw1(), temporal_rw2(), temporal_ar1() for the underlying temporal priors; temporal_gp() for a GP over time that is not a varying coefficient.

Examples

```
# Intercept that drifts as a first-order random walk over year
temporal_tvc("year", structure = "rw1")

# A slope evolving as a continuous-time GP over irregularly-spaced visits
temporal_tvc("day", terms = ~ x - 1, structure = "gp", cov = "matern")
```

---

test_dispersion *Test for over- or underdispersion*

Description

Compares the variance of observed data to the variance expected under the fitted model (via simulation). Ratio > 1 = overdispersion; < 1 = underdispersion. Equivalent to DHARMa::testDispersion().

Usage

```
test_dispersion(object, ...)

## Default S3 method:
test_dispersion(
  object,
  observed = NULL,
  nsim = 250L,
  seed = 123L,
  alternative = c("two.sided", "greater", "less"),
  ...
)
```

Arguments

object	A fitted model with simulate() method
...	Passed to methods.
observed	Observed response vector (optional – extracted from fit)
nsim	Number of simulations (default 250)
seed	Random seed (default 123)
alternative	"two.sided", "greater", or "less"

Value

An htest object with dispersion ratio and p-value

Examples

```
set.seed(1)
d <- data.frame(x = rnorm(80))
d$y <- rpois(80, exp(0.3 + 0.5 * d$x))
fit <- tulpa(y ~ x, data = d, family = "poisson", mode = "laplace")
test_dispersion(fit, nsim = 50)
```

---

test_outliers *Test for outliers (simulation envelope)*

Description

Counts how many observations fall outside the min-to-max range of all simulations. Under a correct model, the expected number is approximately `2 * N / (nsim + 1)`. Equivalent to `DHARMa::testOutliers()`.

Usage

```
test_outliers(object, ...)

## Default S3 method:
test_outliers(object, observed = NULL, nsim = 250L, seed = 123L, ...)
```

Arguments

`object`	A fitted model with `simulate()` method
`...`	Passed to methods.
`observed`	Observed response vector (optional)
`nsim`	Number of simulations (default 250)
`seed`	Random seed (default 123)

Value

An `htest` object (binomial test)

Examples

```
set.seed(1)
d <- data.frame(x = rnorm(80))
d$y <- rpois(80, exp(0.3 + 0.5 * d$x))
fit <- tulpa(y ~ x, data = d, family = "poisson", mode = "laplace")
test_outliers(fit, nsim = 50)
```

`test_uniformity` *Test uniformity of PIT residuals*

Description

If the model is correct, PIT residuals should follow Uniform(0, 1). Applies a Kolmogorov-Smirnov test against that null. Equivalent to `DHARMa::testUniformity()`.

Usage

```
test_uniformity(
  object,
  observed = NULL,
  nsim = 250L,
  seed = 123L,
  plot = FALSE
)
```

Arguments

`object`	A fitted model, a numeric vector of PIT residuals, or a matrix of simulations
`observed`	Observed data (if object is simulations matrix)
`nsim`	Number of simulations (default 250)
`seed`	Random seed (default 123)
`plot`	If TRUE, draws a QQ plot

Value

An `htest` object (KS test result)

Examples

```
set.seed(1)
test_uniformity(runif(50))   # PIT residuals supplied directly

set.seed(1)
d <- data.frame(x = rnorm(80))
d$y <- rpois(80, exp(0.3 + 0.5 * d$x))
fit <- tulpa(y ~ x, data = d, family = "poisson", mode = "laplace")
test_uniformity(fit, nsim = 50)
```

test_zero_inflation *Test for zero inflation*

Description

Compares the number of zeros in observed data to the distribution expected under the fitted model. Equivalent to `DHARMa::testZeroInflation()`.

Usage

```
test_zero_inflation(object, ...)

## Default S3 method:
test_zero_inflation(object, observed = NULL, nsim = 250L, seed = 123L, ...)
```

Arguments

`object`	A fitted model with `simulate()` method
`...`	Passed to methods.
`observed`	Observed response vector (optional)
`nsim`	Number of simulations (default 250)
`seed`	Random seed (default 123)

Value

An `htest` object with zero-inflation ratio and p-value

Examples

```
set.seed(1)
d <- data.frame(x = rnorm(80))
d$y <- rpois(80, exp(0.3 + 0.5 * d$x))
fit <- tulpa(y ~ x, data = d, family = "poisson", mode = "laplace")
test_zero_inflation(fit, nsim = 50)
```

tgmrf *User-defined GMRF latent block*

Description

`tgmrf()` lets a user define a Gaussian Markov random field as a latent block in a tulpa model by supplying two R closures: a precision-matrix factory `Q(theta)` and a log-prior on theta `prior(theta)`. The resulting object plugs into a formula via `latent(tgmrf(...))` and is consumed by every tulpa inference tier (Laplace, EM+Laplace, VI, nested_laplace+CCD, NUTS, IMH-Laplace).

This is the latent-side dual of `LikelihoodSpec`. `LikelihoodSpec` lets a model package own the observation model; `tgmrf()` lets a script own a single latent block.

Usage

```
tgmrf(
  Q,
  prior,
  init,
  mu = NULL,
  graph = NULL,
  bounds = NULL,
  obs_idx = NULL,
  name = NULL
)

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

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

Arguments

Q	A function `function(theta)` returning a sparse precision matrix of class `Matrix::sparseMatrix` (typically `dgCMatrix`). The matrix must be square and symmetric. Called once at registration with `theta = init` to infer `n_latent` and capture the sparsity pattern.
prior	A function `function(theta)` returning a finite numeric scalar – the log-prior density at `theta`.
init	Numeric vector of starting values for `theta`. Names, if present, become the canonical theta names.
mu	Optional function `function(theta)` returning a numeric vector of length `n_latent`. Default `NULL` is equivalent to a zero mean.
graph	Optional sparse matrix specifying the upper bound on `Q`'s sparsity pattern. If supplied, the registration check verifies that the nonzero pattern of `Q(init)` is a subset.
bounds	Optional list with components `lower` and `upper`, each a numeric vector of length `length(init)`. Used by `nested_laplace + CCD` to build the outer grid; unused by NUTS / VI.
obs_idx	Optional integer vector mapping each observation to a latent slot in `[1, n_latent]`. If `NULL` (default), the fit-time driver assumes `N == n_latent` and uses `seq_len(N)` – i.e. one observation per latent slot, in row order.
name	Optional character; cosmetic label used by `print()` / `summary()`.
x	A `tgmrf` block.
...	Passed from `summary()` to `print()`.
object	A `tgmrf` block.

Details

For a Gaussian latent block `z ~ N(mu(theta), Q(theta)^{-1})`:

$$\log p(z \mid \theta) = \tfrac{1}{2}\log\det Q(\theta) - \tfrac{1}{2}(z-\mu(\theta))^\top Q(\theta)(z-\mu(\theta)) + \text{const}$$

the gradient `-Q(z - mu)` and Hessian `-Q` wrt `z` are closed form, so the user never writes gradient code. The Laplace inner step needs only `Q(theta)` and `mu(theta)` at numeric `theta`. NUTS and VI additionally use a forward finite-difference gradient on theta, costing `dim(theta)` extra `Q` calls per outer step – cheap for the typical `dim(theta) <= 5`.

Value

An object of class `c("tgmrf", "tulpa_latent_block")` with components:

- `Q`, `prior`, `mu` – the user closures (or `NULL` for `mu`).
- `init`, `theta_names`, `theta_dim` – hyperparameter metadata.
- `n_latent` – inferred from `Q(init)`.
- `pattern` – captured sparsity pattern as a `dgCMatrix` of 1s.
- `graph` – user-supplied pattern, validated against `pattern` if given.
- `bounds`, `name` – passed through.

`summary()` prints the block, its log prior at `init` and its bounds, and returns `object` invisibly.

See Also

`latent()` for the formula slot that consumes a `tgmrf` block.

Examples

```
# Periodic AR(1) block, wrap-around tridiagonal precision.
periodic_ar1 <- function(n) {
  tgmrf(
    Q = function(theta) {
      sigma <- exp(theta[1]); rho <- tanh(theta[2])
      d <- rep((1 + rho^2) / sigma^2, n)
      o <- rep(-rho / sigma^2, n)
      M <- Matrix::bandSparse(
        n, k = c(-1, 0, 1), diagonals = list(o, d, o)
      )
      M[1, n] <- M[n, 1] <- -rho / sigma^2
      methods::as(M, "CsparseMatrix")
    },
    prior = function(theta) {
      stats::dnorm(theta[1], 0, 1, log = TRUE) +
        stats::dnorm(theta[2], 0, 1, log = TRUE)
    },
    init = c(log_sigma = 0, atanh_rho = 0),
    name = "periodic_ar1"
  )
```

```
}

blk <- periodic_ar1(20)
print(blk)
```

---

tgmrf_cpp *User-defined GMRF latent block, compiled C++ backend*

Description

tgmrf_cpp() is the compiled-C++ analogue of tgmrf(). Where tgmrf() takes R closures Q(theta) / prior(theta), tgmrf_cpp() takes a user-written .cpp file that defines the same kernels as templated C++ (so the same source compiles against every AD type tulpa uses). The returned object has the same S3 class as tgmrf() – c("tgmrf", "tulpa_latent_block") – so every downstream consumer (formula parser, inference layers, S3 methods) treats the two paths identically. Only the dispatch on Q(theta) differs: registry-stored function pointer (tgmrf_cpp()) vs Rcpp::Function callback (tgmrf()).

The user .cpp file must include <tulpa/tgmrf.h> and call the TULPA_REGISTER_TGMRF(id, Q_fn, mu_fn, log_prior_fn) macro once. See inst/examples/tgmrf_periodic_ar1.cpp for a worked example.

Usage

```
tgmrf_cpp(
  cpp_file,
  id,
  init,
  mu = NULL,
  graph = NULL,
  bounds = NULL,
  obs_idx = NULL,
  name = NULL,
  cache_dir = tulpa_cache_dir(),
  rebuild = FALSE
)
```

Arguments

cpp_file	Absolute path to the user's .cpp file. The file is compiled via Rcpp::sourceCpp() with caching keyed on digest::sha256(file_contents) + TULPA_ABI_VERSION so repeated calls with an unchanged source skip the rebuild.
id	Character: the stable id used as the registry key. Must match the first argument of TULPA_REGISTER_TGMRF in the user's .cpp.

`init`, `mu`, `graph`, `bounds`, `obs_idx`, `name`
: Same arguments as `tgmrf()`. `init` is required and supplies the canonical theta names. `mu` is currently ignored on the C++ path – the mu kernel registered by `TULPA_REGISTER_TGMRF` is the source of truth; pass the argument here only to keep call sites symmetric with `tgmrf()`.

`cache_dir`
: Directory for Rcpp's `sourceCpp` cache. Defaults to a per-user location under `tools::R_user_dir("tulpa", "cache")`.

`rebuild`
: Force a recompile even if the cached DLL is up to date. Default `FALSE`.

Value

An object of class `c("tgmrf", "tulpa_latent_block")` with the same fields as `tgmrf()` plus `backend = "cpp"` and `cpp_id = id`. The `Q` / `prior` fields are R wrapper closures around the registered C++ kernels, so callers (e.g. NUTS-over-theta) that hold the object can still call `block$Q(theta)` directly.

See Also

`tgmrf()` for the R-closure path; `latent()` for the formula slot that consumes a `tgmrf` block.

Examples

```
# Compile the periodic AR(1) block shipped with the package (80 nodes,
# theta = (log sigma, atanh rho)) into a temporary cache, then evaluate
# its registered precision kernel at the initial theta.
cpp_file <- system.file("examples", "tgmrf_periodic_ar1.cpp",
                        package = "tulpa")
blk <- tgmrf_cpp(cpp_file, id = "tgmrf_periodic_ar1",
                 init = c(log_sigma = 0, atanh_rho = atanh(0.5)),
                 name = "periodic_ar1",
                 cache_dir = file.path(tempdir(), "tulpa-cache"))
blk$n_latent
Q <- blk$Q(c(0, atanh(0.5)))
dim(Q)
Q[1:3, 1:3]
```

`tidy.tulpa_fit` *Tidy fixed-effect table (broom-compatible)*

Description

Tidy fixed-effect table (broom-compatible)

Usage

```
## S3 method for class 'tulpa_fit'
tidy(x, conf.level = 0.95, ...)
```

Arguments

`x`	A `tulpa_fit` object.
`conf.level`	Interval level (default 0.95).
`...`	Ignored.

Value

Data frame: term, estimate, std.error, conf.low, conf.high.

Examples

```
set.seed(1)
df <- data.frame(x = rnorm(100), g = factor(rep(1:10, 10)))
df$y <- rpois(100, exp(0.3 * df$x))
fit <- tulpa(y ~ x + (1 | g), data = df, family = "poisson")
tidy(fit)
```

`tulpa` *Fit a tulpa model*

Description

Single entry point for fitting a Bayesian hierarchical model. `tulpa()` parses the formula, builds the model matrices, selects an inference backend through the tier/mode system (see `inference_mode_info()`), assembles the arguments that backend needs, and dispatches.

The fit conditions on the random-effect standard deviations `sigma_re` (and, for non-Gaussian dispersion, `phi`): both the Laplace (Tier 2) and the sampler (Tier 1) paths target the posterior given these. Integrating over the hyperparameters is the role of the nested-Laplace / EM layer.

Usage

```
tulpa(
  formula,
  data,
  family = "gaussian",
  mode = "auto",
  sigma_re = NULL,
  n_trials = NULL,
  weights = NULL,
  phi = 1,
  estimate_phi = FALSE,
  phi2 = NULL,
  beta_prior = NULL,
  re_prior = NULL,
  hyperprior = c("proper", "flat"),
```

```
  ziformula = NULL,
  zi_prior = NULL,
  warm_start = NULL,
  spatial = NULL,
  temporal = NULL,
  control = list(),
  ...
)
```

Arguments

`formula` A model formula. Fixed effects, `(1 | g)` / `(1 + x | g)` random effects, and `offset(...)` terms are recognised. As in `stats::lm()` and lme4, aliased fixed-effect columns of a rank-deficient design are dropped with a warning, and unused levels of a grouping factor are dropped. A binomial response may be logical (`TRUE` a success).

`data` A data frame. Incomplete rows (an `NA` in the response, a predictor or a grouping variable) are refused, not dropped.

`family` Character family name: one of `family_names()` (`"binomial"`, `"poisson"`, `"neg_binomial_2"`, `"gaussian"`, `"beta"`, ...), or a categorical response family – `"multinomial"` (baseline-category logit via `tulpa_multinomial()`), `"ordinal"` (cumulative logit via `tulpa_ordinal()`), or `"ordinal_probit"` (cumulative probit). Categorical families take fixed-effect models only.

`mode` Inference mode or backend. `"auto"` (default) picks the most reliable Tier 1/Tier 2 method expected to finish; a tier (`"exact"`, `"structured"`) or a backend name (`"laplace"`, `"mala"`, ...) forces it. `"eb"` estimates the random-effect covariance(s) by empirical Bayes instead of conditioning on `sigma_re` (see `tulpa_eb()`); it is opt-in by name, because its intervals are conditional on that estimate rather than marginal over it. The maximized likelihood integrates the fixed effects out rather than profiling them, so with `hyperprior = "flat"` the estimate is restricted (REML-type) – it matches `glmmTMB(..., REML = TRUE)`, not the ML fit of `lme4::glmer()`.

`sigma_re` Random-effect SDs to condition on: length 1 (recycled) or one per RE term. Defaults to 1 per term with a warning. Ignored by every backend that DETERMINES the RE scale itself – by integrating it (`re_cov_nested`, `re_cov_gibbs`), sampling it (`gibbs` and the ModelData samplers, which carry `log_sigma_re` in the latent vector) or maximizing over it (`eb`, `agq`) – and supplying it there warns.

`n_trials` Binomial denominators (length `nrow(data)`), or `NULL`.

`weights` Optional observation weights (non-negative numeric vector, length `nrow(data)`): each observation's log-likelihood contribution is scaled by its weight (prior / frequency weights, e.g. survey weights or aggregated-data counts – a weight of 2 is equivalent to duplicating the row). Supported on the non-spatial Laplace path (`mode = "laplace"`), the nested-Laplace random-effect covariance integrator (`re_cov_nested`, which `mode = "auto"` / `"structured"` pick for a weighted random-effect model, so its scale is still integrated) and the log-posterior samplers (`mala`, `imh_laplace`, `pathfinder`); other backends reject weights loudly.

phi
: Dispersion passed to the family, held fixed. One convention at every door: for gaussian / lognormal this is the residual VARIANCE (the SD is sqrt(phi)), for neg_binomial_2 the size, gamma the shape, beta the precision, t the scale; binomial and poisson ignore it. The compiled kernels parameterize the two variance families by the residual SD and are handed sqrt(phi) at the boundary.

  Defaulted, it conditions at 1 and says so with a warning, since for a family that reads a dispersion that is a modelling choice rather than a neutral value. fit$phi_estimated records whether the value on the fit was estimated or conditioned on.

estimate_phi
: Estimate the dispersion from the data instead of conditioning on phi, which then supplies the starting value. log(phi) joins the empirical-Bayes maximization as one further coordinate carrying the exact derivative of the Laplace log-marginal, so the estimate is ML-II: the hyperprior covers the random-effect covariances only and the dispersion enters unpenalized. That marginal integrates the fixed effects out, so the estimate is the restricted (REML-type) one – for a gaussian response, the REML residual variance. fit$phi is the estimate and fit$phi_estimated distinguishes it from a conditioned value.

  Available under mode = "eb", and for the families whose dispersion derivative is registered (see tulpa_eb()). Any other mode errors rather than fitting at the starting value under a name that says otherwise.

phi2
: Optional second dispersion: the Student-t degrees of freedom (family = "t"; default 4 when NULL) or the Tweedie variance power (family = "tweedie", required – a defaulted power would be a statistical decision the caller never made). Supported on the non-spatial Laplace path, the random-effect covariance paths (mode = "eb" and the nested Sigma integrator, which thread it into their inner Laplace solve), the log-posterior samplers, and the ModelData samplers. Backends without a phi2 channel refuse it rather than fit at the family's default. estimate_phi covers phi alone; phi2 is always conditioned on.

beta_prior
: Optional list(mean, sd) Gaussian prior on the fixed effects. NULL takes the engine default, prior_normal(0, 2.5), on every backend that carries a fixed-effect prior – the prior is a modelling statement, so the backend mode = "auto" selects does not change it. The nested-Laplace and SPDE paths hold their own field-conditional prior and reject a supplied beta_prior. The resolved prior is reported on the fit as $beta_prior.

re_prior
: Optional list() of random-effect / variance-component hyperpriors (statistical, so they live in the signature rather than in control). Recognised entries, each consumed by the backend that needs it: prior_sigma (PC-prior anchor c(U, alpha) on a free RE covariance SD, used when hyperprior = "proper"), eta (LKJ concentration for a correlated RE covariance, same condition), prior_df / prior_scale (inverse-Wishart on the RE covariance, control$re_cov = "gibbs"), prior_sigma_scale (half-Cauchy scale on the RE SD for mode = "gibbs"), and sigma_re_scale (half-Cauchy scale on the RE / BYM2 SD for the ModelData samplers). An entry the resolved backend does not read is an error naming the backends that do.

hyperprior
: The outer hyperparameter prior, "proper" (default) or "flat", forwarded to every route that integrates or maximizes over hyperparameters: the nested-Laplace path (see tulpa_nested_laplace()), the SPDE path (fit_spde()), and the

random-effect covariance routes (mode = "laplace" with random slopes, tulpa_re_cov_nested(); mode = "eb", tulpa_eb()). "proper" is each route's normalised default (the PC prior on a scale, the PC range prior, PC + LKJ over a free covariance); "flat" folds none of the engine's own, so the fit reports no evidence. A density the call states applies under either. Every other backend carries priors of its own, and "flat" there is an error rather than a choice that silently does nothing.

ziformula
: Optional one-sided formula for the zero-inflation probability, e.g. ~ 1 for a constant structural-zero rate or ~ x to model it. The response becomes a mixture: with probability plogis(X_zi beta_zi) the observation is a structural zero, otherwise it is drawn from family. Available for the count families with a compiled zero-inflated kernel; paired with truncated_poisson or truncated_neg_binomial_2 it is the hurdle model, since the base P(Y = 0) is then 0 and the mixture degenerates to the two-part likelihood. Backends that do not carry the mixture refuse it rather than fit the model without it.

zi_prior
: Optional list(sd) Gaussian prior on the zero-inflation coefficients, beta_zi ~ N(0, sd^2); NULL (default) uses 2.5. One scalar SD applies to the whole block, and the mean is fixed at 0, because that is what the compiled kernels carry. The prior is what identifies the logit where a level contributes no zeros – there the likelihood is monotone in that coefficient and alone would send it to -Inf. sd = Inf removes the penalty. Ignored without ziformula.

warm_start
: Optional starting point for the NUTS sampler, from a cheaper fit of the same model: "eb" or "laplace" fits one first, or pass an existing fit from either mode. The sampler then starts at that mode with an inverse mass read off its curvature, instead of at the origin with a structural one. Chains after the first are dispersed around the mode at the fit's own scale, so between-chain spread – which rhat() compares against – is not collapsed by the shared starting point. Only the NUTS/HMC backends take one; the rest error rather than ignore it. Not available under a spatial, temporal or GP field, whose hyperparameters neither source fit estimates.

  The variance-component slots take an adapting mass by default, because a plug-in fit estimates no curvature for them. Passing a fit from tulpa_eb() with marginal = TRUE supplies one: its theta_cov gives each log_sigma_re slot a posterior variance to start from. This applies to uncorrelated terms, whose hyperparameter coordinates are the log standard deviations the sampler holds; a correlated term stays adapting, since its log-Cholesky coordinates are not the sampler's.

spatial
: Optional spatial-field spec. How it is addressed depends on the field family:

  - **Areal** ("icar", "car", "bym2", "car_proper"): a list with type and adjacency, paired with a spatial(col) term in formula naming the per-observation unit column. Term and spec must be supplied together.
  - **Continuous** (spatial_gp(~ lon + lat) for an NNGP field, spatial_gp(~ lon + lat, approx = 'hsgp') for a Hilbert-space GP, spatial_spde(~ lon + lat, data) for a Matern SPDE field): the spec object carries the coordinate columns (the SPDE spec also carries the mesh + FEM matrices), so **no** spatial(col) term is used – observations are mapped to locations from their coordinates.

  The mode selects how the spatial hyperparameter is handled:

- mode = "nested_laplace", "structured", and "auto" (when not the binomial Gibbs case below) **integrate** the hyperparameter – the designed Tier 2 path, mirroring latent(...) blocks. Areal icar/car/ bym2/car_proper and continuous gp/nngp/hsgp go through tulpa_nested_laplace(); SPDE is redirected to fit_spde(), which integrates (range, sigma) with its own CCD / grid design.
- mode = "laplace" **conditions** on a fixed hyperparameter via tulpa_laplace() (the cheap explicit fit).
- mode = "gibbs" routes the areal icar/bym2 cases through the binomial Polya-Gamma samplers (Tier 1 exact); mode = "auto" picks this for a binomial icar/bym2 field.

temporal

Optional temporal field spec, integrated by nested Laplace for the discrete walks and sampled for the continuous ones: temporal_rw1(), temporal_rw2(), temporal_ar1() (nested Laplace); temporal_gp() and temporal_multiscale(), which are sampler-path only – their hyperparameters are sampled jointly with the field, so mode = "auto" routes them to the exact ModelData sampler. A plain field routes the single-block temporal kernel; a group_var panel spec fits a separate walk per group sharing one hyperparameter; combined with an areal spatial field it forms an additive space-time joint prior.

control

Optional list of backend tuning arguments. Each backend accepts its own set, checked at the door: n_iter / warmup / epsilon (mala), n_draws (pathfinder), n_chains / max_treedepth / adapt_delta / mass_matrix (hmc), the outer-grid knobs ?tulpa_nested_laplace documents (n_per_axis, prune, screen_iters, diagnose_k, within_cell, checkpoint, the progress* family, ...), and re_cov ("nested" / "gibbs" / "aghq"), which selects the RE-covariance integrator on any random-effect model.

n_iter does not count the same thing on every sampler, and warmup (alias n_warmup) is validated against it before any sampler runs:

- On hmc, ess, sghmc, sgld, gibbs, mala and imh_laplace, n_iter is the TOTAL number of iterations, warmup included, so the fit keeps n_iter - warmup draws per chain and warmup < n_iter is required (n_iter = 2000, warmup = 1000 keeps 1000).
- On re_cov_gibbs (control$re_cov = "gibbs") and mclmc, n_iter is the number of KEPT iterations and warmup runs on top of it, so any warmup >= 0 is valid (n_iter = 1000, warmup = 1000 keeps 1000).
- smc and vi read neither (n_particles, vi_max_iter instead).

The default warmup is n_iter %/% 2 on the ModelData samplers and 1000 on gibbs / re_cov_gibbs.

One statistical knob lives here rather than in the signature: marginal (mode = "eb" only) turns on the marginal-Laplace covariance correction, which widens the reported intervals to account for the hyperparameter uncertainty EB conditions on. It is a formal argument of tulpa_eb() and is forwarded from control by tulpa() alone, so it has no meaning on any other backend and is refused there. See tulpa_eb().

...

Reserved for future statistical arguments. Nothing is read from it today, so any entry errors: a stray name here is a misspelled argument or a tuning knob that belongs in control.

Value

A `tulpa_fit` object carrying the backend's output plus `inference_mode`, `inference_tier`, `backend`, `selection_reason`, `formula`, and `family`. Two field-name conventions to know when reaching into the object directly (the generic accessors handle both): on nested-Laplace fits `$weights` is the hyperparameter GRID weights; user observation weights are stored as `$obs_weights`. `$draws` is a draws matrix on engine fits, while model-package fits may carry a list (`$y_rep`, `$log_lik`) under the same name.

Coverage

- **No random effects** and **random intercepts** (`(1 | g)`) are supported on the design path (`mode = "laplace"`) and the sampler path (`mode = "mala"`, `"pathfinder"`, `"imh_laplace"`, and the ModelData kernels `"hmc"` / `"sghmc"` / `"sgld"` / `"mclmc"` / `"smc"` / `"vi"` / `"ess"`).
- **Random slopes** are supported on the Laplace (Tier 2) path: there is no scalar `sigma_re` to condition on, so the RE covariance `Sigma` is integrated rather than fixed. This covers correlated terms (`(1 + x | g)`, a full `Sigma`), uncorrelated terms (`(1 + x || g)`, a diagonal `Sigma`), and several terms together (`(1 + x | g) + (1 | h)`) – each term becomes a covariance block, and any accompanying `(1 | g)` term is integrated as a 1x1 block (nothing is silently conditioned at `sigma_re = 1`). `mode = "laplace"` routes to the nested-Laplace `Sigma` integrator (`tulpa_re_cov_nested()`, CCD design, PC + LKJ hyperprior by default – see `hyperprior`); `control$re_cov = "gibbs"` switches to the exact Metropolis-within-Gibbs debias (`tulpa_re_cov_gibbs()`), and `control$re_cov = "aghq"` keeps the nested integrator but replaces the inner joint-Laplace marginal with adaptive Gauss-Hermite quadrature (`control$n_quad`, default 9 there; see `n_quad` in `tulpa_re_cov_nested()`). Both also run on the sampler path (`mode = "mala"` / `"pathfinder"`).
- `mode = "gibbs"` (Polya-Gamma) fits a single random-intercept model for `family = "binomial"` or `"neg_binomial_2"`, and **samples** the RE sd rather than conditioning on `sigma_re`; tune it via `re_prior$prior_sigma_scale` and a mean-zero `beta_prior`.
- **Latent prior blocks** (`latent(tgmrf(...))`) route to the nested-Laplace path (Tier 2), which integrates over the block hyperparameters. `mode = "auto"` and `"structured"` select it automatically when latent blocks are present; `mode = "nested_laplace"` forces it. Several random-intercept `(1 | g)` terms may accompany the blocks; the one-term restriction belongs to the Polya-Gamma spatial Gibbs sampler (`mode = "gibbs"`), which updates one RE block alongside the field. Joint multi-arm nested models cannot be expressed by a single-response formula – call `tulpa_nested_laplace_joint()` directly.
- The ModelData sampler kernels (`"hmc"`, `"ess"`, `"sghmc"`, `"sgld"`, `"mclmc"`, `"smc"`, `"vi"`) thread the full latent vector – fixed effects, random effects (all forms), areal spatial (`icar` / `bym2`), and temporal (`rw1` / `rw2` / `ar1`) – through one ModelData builder and sample the variance components jointly with the field. `ess` carries random effects but declines a structured spatial / temporal block (its isotropic Gaussian-prior block cannot represent the graph precision); continuous-coordinate fields (`gp` / `nngp` / `hsgp` / `spde`), `car_proper`, and exotic latent blocks stay on the dedicated nested-Laplace / SPDE / Polya-Gamma paths.

References

Rue, Martino & Chopin (2009). Approximate Bayesian inference for latent Gaussian models by using integrated nested Laplace approximations. *JRSS-B* 71(2):319-392. Hoffman & Gelman

(2014). The No-U-Turn Sampler: adaptively setting path lengths in Hamiltonian Monte Carlo. *JMLR* 15(47):1593-1623.

See Also

inference_mode_info(), tulpa_laplace(), mala(), pathfinder()

Examples

```
set.seed(1)
n <- 200L
g <- sample(letters[1:12], n, replace = TRUE)
d <- data.frame(
  y = rbinom(n, 1, plogis(-0.3 + 0.6 * rnorm(n))),
  x = rnorm(n),
  g = g
)
# Random-intercept logistic GLMM, Laplace tier.
fit <- tulpa(y ~ x + (1 | g), data = d, family = "binomial", mode = "laplace")
coef(fit)
summary(fit)
```

---

tulpa_bar_field_replicate

Replicate an areal graph across the levels of a factor (replicated CAR)

Description

Build the block-diagonal Kronecker graph `I_L (x) Q` and the level-offset node index for a replicated areal field: one independent copy of the graph per level of a `by` factor, all sharing one precision. This is the graph-side counterpart to tulpa_bar_field_specs() – that helper expands the coefficient columns and is graph-agnostic, while replication needs the graph, so it is a sibling rather than a new argument. A downstream package composes the two (column expansion x replication) from the one implementation rather than re-deriving the Kronecker remap.

Usage

```
tulpa_bar_field_replicate(adjacency, node, by)
```

Arguments

`adjacency`	Symmetric adjacency matrix of the base graph (`[n_node x n_node]`, dense or sparse).
`node`	Integer vector of 1-based graph-node indices, one per observation (the resolved bar right-hand side).
`by`	A vector of the same length as `node` giving each observation's replication level; coerced to a factor. With `L` distinct levels the field is replicated `L` times.

Value

A list:

`adjacency` the `[L*n_node x L*n_node]` block-diagonal Kronecker adjacency `I_L (x) Q` (the base graph for `L == 1`).

`index` integer vector, one per observation: the node index offset into its level's copy (`node + (level - 1) * n_node`).

`n_levels` the number of replication levels `L`.

`n_nodes` the base graph node count `n_node`.

`levels` the factor levels of `by`, in replicate order.

See Also

`tulpa_bar_field_specs()` for the coefficient-column expansion, `spatial()` for the inline areal field constructor whose `by =` argument this powers.

Examples

```
adj <- matrix(0, 4, 4)
for (i in 1:3) adj[i, i + 1] <- adj[i + 1, i] <- 1
node <- rep(1:4, times = 2)
lev  <- rep(c("a", "b"), each = 4)
rep_info <- tulpa_bar_field_replicate(adj, node, lev)
dim(rep_info$adjacency)   # 8 x 8 (I_2 (x) Q)
rep_info$index            # level b nodes offset by 4
```

`tulpa_bar_field_specs` *Expand a varying-coefficient bar into per-column field specs*

Description

Expand the left-hand side of an **lme4**-style varying-coefficient bar (`~ 1 + w || node`) against a data frame into one descriptor per design matrix column. This is the same expansion `spatial()` and the inline `temporal()` field constructor use internally to turn a bar into one CAR / temporal field per design column, exposed so a downstream package can reuse the one implementation rather than re-parsing the bar grammar.

The bar's right-hand side names the node index (the graph node for a spatial field, the time index for a temporal field); it is not expanded but is returned as the `node` attribute. The left-hand side is expanded with `stats::model.matrix()`: the intercept column (`1`) is the unweighted (all-ones) field, and each covariate column is a varying coefficient whose per-observation weight is that column's design value (`0 +` drops the intercept).

Usage

```
tulpa_bar_field_specs(formula, data)
```

Arguments

`formula` A one-sided formula carrying a single grouping bar, e.g. `~ 1 + w || cell`. Use `tulpa_is_spatial_bar()` to test first. The right-hand side must be a single bare column naming the node index; nesting (`a / b`), interaction (`a:b`), or expression grouping is rejected.

`data` A data frame whose columns the bar left-hand side and the node index refer to. The per-column weight vectors are evaluated against it, so they have `nrow(data)` entries.

Value

A list with one element per design-matrix column, each a list:

`column_name` character; `"Intercept"` for the intercept column, otherwise the `model.matrix()` column name (e.g. `"w"`).

`weight` a numeric vector of length `nrow(data)` for a covariate column (the per-observation design value scaling that field), or `NULL` for the intercept column (which is the all-ones, unweighted field).

`is_intercept` logical; `TRUE` for the intercept column.

The list carries two attributes: `node` (character, the node-index column named by the bar right-hand side) and `correlated` (logical, `TRUE` for a single `|`, `FALSE` for a double `||`).

See Also

`tulpa_is_spatial_bar()` for the recognizer, `spatial()` for the inline areal field constructor that consumes this expansion.

Examples

```
d <- data.frame(cell = rep(1:5, each = 4), w = rnorm(20))

# Intercept plus a varying slope on w
specs <- tulpa_bar_field_specs(~ 1 + w || cell, d)
length(specs)                 # 2
specs[[1]]$column_name        # "Intercept"
is.null(specs[[1]]$weight)    # TRUE (unweighted field)
specs[[2]]$column_name        # "w"
identical(specs[[2]]$weight, d$w)  # TRUE
attr(specs, "node")           # "cell"
```

tulpa_batched_pareto_k

Outer Pareto-k-hat for a batched-target grid-integrated fit

Description

Outer Pareto-k-hat reliability diagnostic for a grid-integrated nested fit whose inner marginal comes from a BATCHED re-fit rather than a per-sample closure: fits a Gaussian proposal at `(theta_hat, L_scale)`, draws from it, evaluates the caller's batched log-target, applies a radius cap with heavy-tail escalation, then reports `pareto_k` / `is_ess` via the shared PSIS/GPD core. This is more than plain PSIS smoothing over an already-computed log-ratio vector – see `tulpa_psis()` for that – because it also builds the proposal, draws, and evaluates the target.

Usage

```
tulpa_batched_pareto_k(
  theta_hat,
  L_scale,
  log_target_batched,
  n_samples = .nl_diag("k_samples"),
  radius_cap = Inf,
  return_draws = FALSE,
  tail_points = NULL,
  Z = NULL
)
```

Arguments

`theta_hat, L_scale`
: Mean and Cholesky scale of the Gaussian proposal `N(theta_hat, L_scale L_scale')`, in the integrator's own coordinate space (e.g. `(log range, log sigma)` for an SPDE field).

`log_target_batched`
: Function taking an `S x d` matrix of proposal draws and returning the integrator's unnormalized log posterior at each row, in that same coordinate space. Any change-of-variables Jacobian is the caller's responsibility.

`n_samples`
: Number of draws from the proposal.

`radius_cap`
: Whitened-radius cap on evaluated draws (`Inf` keeps every draw); draws past the cap are folded back in under heavy-tail escalation rather than dropped.

`return_draws`
: If `TRUE`, include the draws in the returned list.

`tail_points`
: Optional override for the PSIS tail size; `NULL` uses the automatic rule.

`Z`
: Optional pre-drawn `S x d` whitened sample; `NULL` draws it here.

Value

A list with `pareto_k`, `is_ess` and `n_eval` (the number of draws whose log importance ratio was finite). When the diagnostic ran to completion it also carries `lr`, those `n_eval` finite log importance ratios in draw order (what `pareto_k` was fitted on, so `tulpa_psis(lr, tail_points)$pareto_k` reproduces it), and with `return_draws = TRUE` the evaluated draws `U` (`n_eval x d`) and their smoothed `log_weights`. When it declined, `pareto_k` and `is_ess` are `NA` and `declined` names the reason (e.g. `"draws_too_few"`).

See Also

`tulpa_psis()`

Examples

```
set.seed(1)
# A Gaussian target a little wider than the standard-normal proposal:
lt <- function(U) -0.5 * rowSums(sweep(U, 2L, c(1.2, 0.8), "/")^2)
k <- tulpa_batched_pareto_k(theta_hat = c(0, 0), L_scale = diag(2),
                            log_target_batched = lt, n_samples = 500)
k$pareto_k
```

`tulpa_cache_clear` *Remove compiled blocks from the tgmrf_cpp() cache*

Description

Deletes cached compilation output under `tulpa_cache_dir()`. The cache holds build artefacts only – a removed entry is rebuilt by the next `tgmrf_cpp()` call on the same source – so it can be emptied at any time.

Usage

```
tulpa_cache_clear(older_than = 0)
```

Arguments

`older_than`	Remove only entries whose last modification is more than this many days ago. `0` (the default) removes every entry.

Value

The number of entries removed, invisibly.

See Also

`tulpa_cache_dir()` for the location.

Examples

```
# Deletes the caller's own cached builds, so it runs only in an
# interactive session.
if (interactive()) {
  tulpa_cache_clear(older_than = 30)  # keep the last 30 days
  tulpa_cache_clear()                 # empty the cache
}
```

---

tulpa_cache_dir *Default cache directory for* `tgmrf_cpp()`*-compiled DLLs*

Description

Reports the per-user directory under `tools::R_user_dir("tulpa", "cache")` where `tgmrf_cpp()` keeps the objects it compiles. Reporting the path does not create it: the directory appears the first time a compile needs it, and `tulpa_cache_clear()` removes what it holds.

Usage

```
tulpa_cache_dir(create = FALSE)
```

Arguments

`create`	Create the directory if it is missing. `FALSE` (the default) only reports the path.

Value

Absolute path (character) to the cache directory.

See Also

`tulpa_cache_clear()` to remove cached objects.

Examples

```
# Where compiled tgmrf_cpp() blocks are cached. Reporting creates nothing.
tulpa_cache_dir()
```

tulpa_check_control *Validate a* `control = list()` *surface against its canonical key set*

Description

Front-door fitters across the `tulpa*` ecosystem carry their perf / numerical / tuning knobs in a single `control` list (design principle 6). Without a name check a misspelled knob is a silent no-op – `control$adaptve_grid` fits the default and reports nothing. This validates the names a caller supplied against the whitelist of what the target fitter actually reads, and errors listing the allowed set.

Usage

```
tulpa_check_control(control, allowed, where)
```

Arguments

`control`	The `control` list to validate. `NULL` and empty lists pass.
`allowed`	Character vector of accepted knob names.
`where`	Name of the calling fitter, used in the error message.

Details

Consumer packages (`tulpaRatio`, `tulpaObs`) call this with their own key registry rather than reimplementing the check.

Value

`invisible(NULL)`, called for the side effect of erroring on an unknown or unnamed knob.

Examples

```
tulpa_check_control(list(max_iter = 50), c("max_iter", "tol"), "my_fit")
try(tulpa_check_control(list(max_itr = 50), c("max_iter", "tol"), "my_fit"))
```

tulpa_criteria *Model criteria from a pointwise log-likelihood*

Description

Turn an `[n_draws x n_obs]` pointwise log-likelihood into the standard Bayesian goodness-of-fit currency: WAIC, DIC, conditional predictive ordinates (CPO) and their sum LPML, PSIS-LOO, all from one matrix. PSIS-LOO reuses the native `tulpa_psis()` smoothing, so the CPO and LOO numbers are the same computation (`CPO_i = exp(elpd_loo_i)`, `LPML = elpd_loo`). The input may be a matrix or a streaming `tulpa_loglik()` so EVA-scale fits are processed in observation blocks.

Usage

```
tulpa_criteria(
  log_lik,
  criteria = c("waic", "loo", "cpo", "lpml", "dic"),
  loglik_at_mean = NULL,
  group = NULL,
  chunk_size = NULL,
  pointwise = FALSE
)

## S3 method for class 'tulpa_criteria'
print(x, digits = 1, ...)
```

Arguments

`log_lik`	An `[n_draws x n_obs]` numeric matrix of pointwise log-likelihoods, or a `tulpa_loglik()` streaming wrapper.
`criteria`	Which criteria to compute. Any of `"waic"`, `"loo"`, `"cpo"`, `"lpml"`, `"dic"`. `"loo"`, `"cpo"`, and `"lpml"` share the single PSIS pass; `"dic"` additionally needs `loglik_at_mean`.
`loglik_at_mean`	Optional length-`n_obs` vector of pointwise log-likelihoods evaluated at the posterior mean of the parameters, supplied by the caller (the model package knows the parameterization). Required for DIC's plug-in deviance; without it the DIC fields are `NA`.
`group`	Optional length-`n_obs` grouping (an integer / factor / character vector). The LOO unit is **one column of** `log_lik`: with `group = NULL` (the default) every column is its own fold (leave-one-row-out, e.g. per plot / per visit) and the result is byte-identical to the ungrouped call. When supplied, the per-draw pointwise log-likelihoods are summed within group to a `[n_draws x n_groups]` matrix **before** PSIS, so each fold is a whole group (leave-one-group-out cross-validation, LOGO-CV). Use it to switch the estimand from per-row to per-group LOO – e.g. on a cell-compressed hierarchical fit, leave out a whole cell rather than one of its rows. WAIC's variance term, `lppd`, `elpd_loo`, `cpo` and `pareto_k` are all computed on the grouped matrix. DIC is a plug-in deviance over all observations and is unaffected by `group`.
`chunk_size`	Number of observation columns to process per block. The default streams the whole matrix at once when materialized, else picks a block sized to a few million entries.
`pointwise`	If `TRUE`, also return the per-observation vectors (`elpd_waic`, `p_waic`, `elpd_loo`, `pareto_k`, `cpo`) for plotting / stacking.
`x`	A `tulpa_criteria` result.
`digits`	Decimal places printed.
`...`	Ignored.

Details

p_waic is the well-known positively-biased variance estimator at low draw counts; the result records n_draws and the count of observations with p_waic_i > 0.4 (the loo heuristic for an unreliable WAIC), and the PSIS-LOO elpd_loo is the more stable figure to report when that count is non-trivial.

The LOO unit is whatever **one column of** log_lik holds. If the consumer built the matrix with one column per row (plot / visit), the default is per-row LOO; if a column already carries a whole group's compressed likelihood, leaving it out drops that group and pareto_k can blow up by construction. The group argument makes the unit explicit: supply it to aggregate columns into folds and report leave-one-group-out CV (LOGO-CV) instead, a different and deliberate estimand.

Value

A tulpa_criteria object: a list with the requested scalar scores (each estimate paired with its standard error where defined), n_draws / n_obs, the PSIS pareto_k summary, and – when pointwise = TRUE – a pointwise data frame.

References

Vehtari, Gelman & Gabry (2017). Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. *Statistics and Computing* 27(5):1413-1432. Watanabe (2010). Spiegelhalter et al. (2002). Geisser & Eddy (1979).

See Also

tulpa_psis() for the smoothing core, tulpa_pit() for the probability-integral-transform companion, compare_models() for model comparison.

Examples

```
# A draws x observations log-likelihood matrix (here built directly;
# in practice extracted from a fitted model's posterior draws).
set.seed(1)
y  <- rnorm(40)
mu <- matrix(rnorm(200 * 40, sd = 0.2), 200, 40)
ll <- dnorm(matrix(y, 200, 40, byrow = TRUE), mean = mu, log = TRUE)
tulpa_criteria(ll)
tulpa_criteria(ll, criteria = "waic", pointwise = TRUE)$pointwise[1:3, ]
```

---

tulpa_diagnostics *Simulation-based diagnostics for tulpa models*

Description

Model-checking tools based on posterior predictive simulation. These are native R implementations equivalent to DHARMa's test suite, with no external dependencies. They work with any model that provides simulate(), fitted(), and residuals() methods.

Value

The diagnostic functions documented in this family return their individual results (a test-statistic object, a data frame of residuals, or a `check_model` summary); see each function's own help page.

---

tulpa_draws_array *Posterior draws as a 3D array*

Description

Assembles a fitted model's posterior draws into an `[iteration, chain, parameter]` array – the layout expected by `bayesplot` and analogous to `posterior::as_draws_array()`. Multiple chains are recognised from a 3D draws array, a `$chain_id` row map, or an `$n_chains` count over chain-major rows; a single pooled chain yields a one-chain array.

Usage

```
tulpa_draws_array(fit)
```

Arguments

`fit`	A `tulpa_fit` (or subclass) carrying posterior `$draws`.

Details

As with `posterior_sample()`, a fit that carries no draws returns `NULL` with a message naming its backend and the representation it carries instead.

Value

A numeric array with dimensions `[n_iter, n_chain, n_param]` and the parameter names on the third dimension, or `NULL` if the fit carries no draws. Chains of unequal length are truncated to the shortest.

See Also

`posterior_sample()`, `tulpa_posterior_draws()`, `diagnostics()`

Examples

```
set.seed(1)
d <- data.frame(x = rnorm(60))
d$y <- rpois(60, exp(0.5 + 0.3 * d$x))
fit <- tulpa(y ~ x, data = d, family = "poisson", mode = "hmc",
             control = list(n_iter = 400L, warmup = 200L, n_chains = 2L,
                            seed = 1L))
dim(tulpa_draws_array(fit))   # [iteration, chain, parameter]
```

tulpa_eb *Empirical-Bayes random-effect covariances*

Description

Estimate one or more random-effect covariances `Sigma` by maximizing the Laplace marginal likelihood over them (plus the hyperprior), then report the fixed effects conditional on the maximizer. This is the plug-in ("ML-II" / empirical-Bayes) counterpart of `tulpa_re_cov_nested()`, which integrates over `Sigma` instead of fixing it at the maximizer.

Usage

```
tulpa_eb(
  y,
  n_trials = NULL,
  X,
  re_terms,
  family = "binomial",
  phi = 1,
  phi2 = NULL,
  prior_sigma = NULL,
  eta = NULL,
  hyperprior = c("proper", "flat"),
  log_prior_theta = NULL,
  beta_prior = NULL,
  offset = NULL,
  n_quad = 1L,
  marginal = FALSE,
  estimate_phi = FALSE,
  X_zi = NULL,
  zi_prior_sd = 2.5,
  control = list()
)
```

Arguments

`y`, `n_trials`, `X`, `family`, `phi`
: Passed to `tulpa_laplace()` for the inner solve. `n_trials = NULL` defaults to 1 (binary / single-trial).

`re_terms`
: Either a single random-effect term or a list of them; see `tulpa_re_cov_nested()` for the per-term fields.

`phi2`
: Optional second dispersion, threaded into every inner `tulpa_laplace()` solve: the Student-t degrees of freedom (`family = "t"`, default 4 when `NULL`) or the Tweedie variance power (`family = "tweedie"`, required – a defaulted power would be a statistical decision the caller never made). A `phi2` supplied for any other family errors rather than being ignored. It is conditioned on, never estimated: `estimate_phi` covers `phi` alone.

prior_sigma, eta

Hyperparameters of the PC + LKJ prior used when hyperprior = "proper" (see re_cov_pc_lkj_prior()); NULL (the defaults) is c(3, 0.01) and 2. Ignored when hyperprior = "flat" or log_prior_theta is supplied. When active, the prior is part of the maximized objective, so it regularizes the estimate: with few groups it is what keeps a block off the sigma = 0 boundary.

hyperprior

"proper" (default) or "flat". "flat" maximizes with log_prior_theta the zero function – an unpenalized restricted (REML-type) estimate, beta integrated out rather than profiled (see Details), which can reach the sigma = 0 boundary on small designs (see the "lower end of the search bracket" warning). "proper" builds the PC + LKJ prior from prior_sigma / eta, regularizing the estimate away from that boundary. Ignored when log_prior_theta is supplied. Must match hyperprior on the paired tulpa_re_cov_nested() call for the two to share theta_hat.

log_prior_theta

Optional function(theta) returning a scalar log prior density on the full stacked parameter vector, overriding hyperprior entirely. Default NULL, which defers to hyperprior.

beta_prior

Optional Gaussian prior on the fixed effects, threaded into every inner tulpa_laplace() solve (list(mean, sd)).

offset

Optional observation-level offset on the linear predictor (length length(y)), e.g. log(exposure) for a rate model. Not supported with n_quad > 1, which errors rather than dropping it.

n_quad

Quadrature order for the inner marginal. 1 (default) uses the joint-field Laplace inner solve. > 1 refines it with n_quad-point adaptive Gauss-Hermite quadrature, which requires a single shared grouping factor; see tulpa_re_cov_nested().

marginal

Report fixed-effect intervals that carry the hyperparameter uncertainty, instead of the intervals conditional on theta_hat. The posterior for theta is taken as Gaussian around the maximizer with covariance solve(H_theta), the inner mode is linearized in theta, and the law of total variance adds J solve(H_theta) J' to the conditional covariance, where J = d mode / d theta. Both H_theta and J come from one central-difference stencil over the outer objective, costing 1 + 2k^2 further inner solves for k hyperparameter coordinates (k is 1 for a scalar (1 | g) block and 3 for a correlated (1 + x | g) one). Default FALSE. Widens intervals; never narrows them. When the variance components themselves are the target, or the correction's two approximations look strained (a strongly skewed variance-component marginal), integrate with tulpa_re_cov_nested() instead.

estimate_phi

Estimate the family's dispersion alongside the random-effect covariances, instead of conditioning on phi. When TRUE the supplied phi is the starting value and fit$phi is the estimate, with fit$phi_estimated distinguishing the two cases. log(phi) joins the maximization as one further coordinate, carrying the exact derivative of the Laplace log-marginal with respect to it, so the cost is one more coordinate for BFGS and not a second optimization.

The dispersion enters unpenalized – the hyperprior covers the covariance coordinates only – so this is the ML-II estimate of phi, not a MAP under an undeclared prior. Like the covariances it is read off the likelihood marginal over beta, so

it is the restricted (REML-type) estimate: for a gaussian response, the REML residual variance.

Available for every family carrying a dispersion, which is every front-door family except `poisson`, `binomial` and `truncated_poisson` – those have no free dispersion at all, so estimating one is a category error rather than a missing feature, and it is refused. Needs `n_quad = 1`.

Alongside `X_zi` both mixture kinds are covered. A hurdle (a zero-truncated base) has zero branch `log(pi)`, which carries no dispersion, so the base family's registered derivative is already the mixture's. Genuine zero inflation has zero branch `log(pi + (1 - pi) P(Y = 0))`, which depends on `phi` through `P(Y = 0)` and couples it to both linear predictors; that branch is supplied for `neg_binomial_2`, the only untruncated mixture family here with a free dispersion. Other untruncated bases are refused rather than handed the base derivative under a model it does not describe.

`X_zi` Optional zero-inflation design matrix (`length(y)` rows), making the model a two-process mixture: each observation is a structural zero with probability `plogis(X_zi beta_zi)` and otherwise follows `family`. Paired with a zero-truncated family it is the hurdle model. The random effects enter the count predictor only, and the maximization is over the same covariance coordinates – the mixture changes the inner solve, not the outer objective's parameters. The ZI coefficients are reported alongside the count ones in `coef()` / `vcov()`, so the fixed block is `ncol(X) + ncol(X_zi)` wide. Needs `n_quad = 1`: the adaptive Gauss-Hermite inner marginal runs through a single-predictor oracle.

`zi_prior_sd` Prior SD on `beta_zi`, keeping the logit identified where a level carries no zeros (the likelihood alone would send it to `-Inf`). Ignored when `X_zi` is `NULL`.

`control` A named list of numerical knobs: `max_iter`, `tol`, `n_threads` (inner-solve controls, see `tulpa_laplace()`), and `outer_maxit` (iteration budget for the maximization over `Sigma`, default 500; applies to the Nelder-Mead simplex used from two parameters up, since the one-parameter case is bracketed by Brent). Exhausting the budget warns. `outer_reltol` sets that maximization's convergence tolerance (default `1e-10` for the gradient-driven methods, `1e-8` for the simplex, which cannot resolve as finely); it is converted to L-BFGS-B's `factr` on the bounded path, so one request means the same thing whichever method runs. `sigma_init` supplies the starting random-effect SD – a scalar, or one per coefficient across all blocks – replacing the method-of-moments guess taken from a pilot fit at `Sigma = I`. Worth setting when the true scale is far from 1, where that pilot starts the search on a flat stretch, and when a run should be reproducible from its inputs rather than from a pilot fit. It is diagonal: it sets each coefficient's scale and leaves any correlation to be fitted. Two further knobs tune `marginal = TRUE` and are inert without it: `marginal_step` (the stencil step in `theta` space, default `1e-3`) and `marginal_richardson` (default `FALSE`; evaluate the stencil at `step` and `step / 2` and extrapolate, turning the `O(step^2)` truncation error into `O(step^4)` at twice the solves – worth it only when the inner solver's own noise sits well below the truncation error, i.e. a tight `tol`).

Details

Blocks, coordinates and the default hyperprior are exactly those of `tulpa_re_cov_nested()` – a correlated `(1 + x | g)` term is a full `Sigma = L L'` in log-Cholesky coordinates, an uncorrelated `(1 + x || g)` term is diagonal in log-SD coordinates, and a scalar `(1 | g)` term is the degenerate one-coefficient block. Both functions call the same outer objective and the same optimizer, so `tulpa_eb()$theta_hat` and `tulpa_re_cov_nested()$theta_hat` are the same estimate on the same data – which requires `hyperprior` to default the same way on both: `"proper"`, the PC + LKJ prior (see `re_cov_pc_lkj_prior()`) at the anchor every scale axis of the engine defaults to, which at small G keeps a block off the `sigma = 0` boundary. `hyperprior = "flat"` maximizes the marginal likelihood alone.

The marginal likelihood is marginal over the fixed effects too: the inner Laplace approximation integrates `beta` out together with the random effects (under `beta_prior`, or the inner solve's vague `N(0, 100^2)` default) rather than profiling it. `hyperprior = "flat"` is therefore a RESTRICTED, REML-type estimate, not maximum likelihood: for a gaussian response it is REML, and for a non-gaussian one it agrees with `glmmTMB(..., REML = TRUE)`, not with the ML fit of `lme4::glmer()` or `glmmTMB(..., REML = FALSE)`. On a binomial random intercept with 40 groups of 6 the three read RE SDs of 0.7931 (`tulpa_eb`), 0.7931 (glmmTMB REML) and 0.7594 (ML); the fixed effects at `theta_hat` match the REML fit's as well. The restriction is what removes ML's downward bias in the variance components when `beta` is estimated from the same data, and it carries over to `n_quad > 1` and to `estimate_phi = TRUE`. Validating against `glmer()` should expect exactly that gap.

The reported fixed-effect covariance is the conditional one at `theta_hat` (`solve(H_beta)`). It does not include the hyperparameter uncertainty that `tulpa_re_cov_nested()` integrates over, so EB intervals are narrower – increasingly so as the number of groups falls. Use the nested integrator when the variance components themselves, or calibrated fixed-effect intervals, are the target; use EB when the point estimate is, or as a fast starting fit.

Value

A `tulpa_fit` with:

- `mode`, `H_beta`: the fixed-effect (and, on the `n_quad = 1` path, random- effect) mode and the fixed-effect precision at `theta_hat`, driving `coef`/`confint`/`vcov`/`summary`.
- `map`: the `Sigma` / `sigma` / `rho` summary at `theta_hat` (a single list for one block, a named list of them for several).
- `Sigma`: the estimated covariance (a matrix for one block, a named list of matrices for several).
- `theta_hat`, `log_marginal`, `layout`, `n_blocks`, `n_coefs`.
- `converged`: whether the inner Newton solve at `theta_hat` converged.
- `outer_convergence`: `optim`'s code for the maximization over `Sigma` (`0` on success). A non-zero value also warns.
- With `marginal = TRUE` and a correction that formed: `cov_marginal` (the widened fixed-effect covariance, which `vcov` / `summary` / `confint` then report), `cov_conditional` (the `solve(H_beta)` they would otherwise have reported, kept so the two are comparable on one fit), `H_theta` and `theta_cov` (the outer Hessian at `theta_hat` and its inverse, named by `theta_names`), and the `marginal_step` / `marginal_richardson` actually used. All absent when the correction was not requested or could not be formed, so `is.null(fit$cov_marginal)` tests whether the reported intervals are marginal. A requested correction that fails warns and leaves the conditional covariance in place.

References

Casella (1985). An introduction to empirical Bayes data analysis. *The American Statistician* 39(2):83-87. Rue, Martino & Chopin (2009). Approximate Bayesian inference for latent Gaussian models by using integrated nested Laplace approximations. *JRSS-B* 71(2):319-392.

See Also

`tulpa_re_cov_nested()` to integrate over `Sigma` rather than fix it; `tulpa_laplace()` for the inner solve.

Examples

```
set.seed(1)
G <- 30L; per <- 10L; n <- G * per
grp <- rep(seq_len(G), each = per); x <- rnorm(n)
b <- rnorm(G, 0, 0.8)
y <- rpois(n, exp(0.3 + 0.5 * x + b[grp]))
re_term <- list(idx = grp, n_groups = G, n_coefs = 1L)
fit <- tulpa_eb(y, NULL, cbind(1, x), re_term, family = "poisson")
fit$map$sigma          # empirical-Bayes RE standard deviation
```

`tulpa_em_laplace` *Fit a latent-variable model via EM + Laplace approximation*

Description

Generic EM engine. Model packages provide callbacks for the E-step (latent variable posterior) and the M-step encoding (how to assemble submodel data from weights). Each M-step submodel block is fit independently via `tulpa_laplace()`; the engine reads `family` and `offset` per block so heterogeneous-family mixtures (e.g. a binomial zero submodel + poisson positive submodel for a hurdle model) work without engine changes.

Usage

```
tulpa_em_laplace(
  e_step,
  m_step_encode,
  spatial = NULL,
  re_list = list(),
  max_iter = 50L,
  tol = 1e-04,
  damping = 0.3,
  correction = c("auto", "mi", "gibbs", "none"),
  n_imputations = 20L,
  n_gibbs = 10L,
  draw_z = NULL,
  m_step_extra = NULL,
```

```
  beta_prior = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

`e_step` Callback: `function(fits, ...) -> list(weights = numeric, ...)`. Called once per EM iteration with the current per-submodel `tulpa_laplace()` fits. Must return a list whose `weights` element is the latent-variable posterior used by the next M-step.

`m_step_encode` Callback: `function(weights, ...) -> list of blocks`. Each block is itself a list with the following fields:

- `y` (numeric, required) – response.
- `X` (matrix, required) – fixed-effects design with `nrow(X) == length(y)`.
- `family` (character scalar, required) – one of `"binomial"`, `"poisson"`, `"gaussian"`, `"negbin"` / `"neg_binomial_2"`, `"gamma"`, `"beta"`. Forwarded to `tulpa_laplace()` for that block.
- `n_trials` (numeric or `NULL`, optional; absent or `NULL` defaults to 1) – binomial trial counts.
- `offset` (numeric or `NULL`, optional) – observation-level offset, length-matched to `y` when non-`NULL`.
- `phi` (numeric scalar, optional) – dispersion forwarded to `tulpa_laplace()` (used by `negbin`, `gamma`).
- `weights` (numeric, optional; length `length(y)`) – per-observation likelihood weight, scaling that row's log-density, score and Fisher information. This is the channel a soft (fractional) latent label travels on; see the section below.
- `re_list`, `spatial` (optional) – forwarded as-is.
- `control` (list, optional) – on a block with a `prior`, entries merged over the driver's own `tulpa_nested_laplace()` control (`n_threads`, `keep_grid_hessians = TRUE`, `diagnose_k = FALSE`); an M-step fit is one iterate of the EM objective, so its outer Pareto-k diagnostic is off unless the block sets `diagnose_k = TRUE`.

`spatial, re_list` Not consumed at the driver level – latent structure is per-submodel, set as a `spatial` / `re_list` field on each block `m_step_encode()` returns. Supplying either here is an error (rather than a silent no-op).

`max_iter` Maximum EM iterations.

`tol` Convergence tolerance on max relative parameter change.

`damping` EM damping factor in `[0, 1)`. With `damping = d`, the E-step weights are smoothed between iterations as `(1 - d) * new + d * prev` (`d = 0` is no damping); the M-step then refits on the smoothed weights, so the parameter update is damped indirectly through the weights rather than by mixing successive parameter vectors.

correction	Post-EM correction. "none" returns the EM point estimate only. "mi" draws n_imputations independent hard z from the converged posterior weights P(z|y, theta_hat), refits each block on the hard z, and pools via rubins_pool(). "gibbs" runs a warm-started z|theta -> theta|z Markov chain of length n_gibbs starting from the EM fits, also pooled via rubins_pool(). "auto" resolves to "none".
n_imputations	Number of MI draws (default 20L). Used when correction = "mi".
n_gibbs	Length of the Gibbs chain (default 10L). Used when correction = "gibbs".
draw_z	Optional function function(weights) -> hard_z that turns the E-step's continuous weights into a hard latent draw. Used only by correction %in% c("mi", "gibbs"). The default treats weights as a numeric vector of Bernoulli probabilities and draws per-observation. Multi-class / matrix-valued latent structures must supply their own callback.
m_step_extra	Optional function(fits, weights, ...) -> fits. Fired once per M-step in every phase (EM iterations, MI draws, Gibbs steps). Receives the freshly assembled list of tulpa_laplace() results (fits), the continuous E-step weights P(z|y, theta) (NOT the hard z draw used by MI/Gibbs to encode the block), and any extra arguments forwarded through .... Returns a list with the same length and names as the input, possibly with mutated dispersion / shape / precision fields (e.g. fits[[k]]$phi). Use this to update non-eta parameters that fall out of the Laplace M-step (NB overdispersion, Gamma shape, Beta precision, Gaussian sigma). When NULL (default), behavior is unchanged.
beta_prior	Optional Gaussian prior on the fixed effects, applied to every block fit via tulpa_laplace() (i.e. blocks without a prior field). NULL (default) keeps the weak built-in prior. Otherwise a list with sd (required) and optional mean; see tulpa_laplace(). The same prior flows into the MI / Gibbs correction refits, so penalized corrections come for free. A block may override the default by setting its own beta_prior field in m_step_encode (e.g. different priors for the occupancy and detection submodels). Use scalar mean / sd here when blocks differ in width; per-coefficient vectors belong on the block.
verbose	Print per-iteration progress.
...	Forwarded to e_step, m_step_encode, and m_step_extra.

Details

This is an engine block, not a front door: model packages call it programmatically, so its tuning knobs (max_iter, tol, damping) sit in the signature rather than in a control list.

Value

A list with:

- fits – named list of tulpa_laplace() results, one per block.
- weights – final E-step weights.
- n_iter – number of EM iterations actually run.
- converged – logical.

- `history` – `data.frame(iter, delta)` of max relative parameter change per iteration.
- `correction` – the resolved correction mode (`"none"`, `"mi"`, or `"gibbs"`).
- `pooled` – present when `correction %in% c("mi", "gibbs")`. Named list of pooled per-submodel summaries from `rubins_pool()`.
- `draws` – present when `correction %in% c("mi", "gibbs")`. List of per-draw fits with `beta` / `se` attached.

Soft latent labels go in `weights`, not in `y`

The M-step maximizes the expected complete-data log-likelihood. For a Bernoulli latent `z_i` carrying E-step posterior weight `w_i` that is

$$Q = \sum_i [\, w_i \log p_i + (1 - w_i) \log(1 - p_i) \,],$$

a **weighted Bernoulli** log-likelihood, which carries no binomial coefficient. Encode it as two rows per unit – `y = 1` at weight `w_i` and `y = 0` at weight `1 - w_i`:

```
list(y       = rep(c(1, 0), each = n),
     X       = rbind(X, X),
     weights = c(w, 1 - w),
     family  = "binomial")
```

A fractional `y` on a binomial block is refused, and is not the same objective: it asks for the exact binomial density at a non-integer response, whose normalizer `lchoose(n, y)` is not zero there and depends on `w`. On the weighted encoding the block's `log_marginal` is the Laplace marginal of `Q`, an EM objective that increases across iterations.

Examples

```
# Zero-inflated Poisson via EM: the E-step scores the posterior probability
# that each zero is non-structural. Those are soft labels, so the occupancy
# arm is the weighted Bernoulli above -- two rows per unit -- and the
# abundance arm weights each count by the same w.
set.seed(1)
n <- 200
z <- rbinom(n, 1, 0.7)
y <- rpois(n, 4) * z
X <- cbind(1, rnorm(n))

e_step <- function(fits, ...) {
  if (!length(fits)) return(list(weights = pmax(as.numeric(y > 0), 0.5)))
  psi <- plogis(drop(X %*% fits$occ$mode))
  lam <- exp(drop(X %*% fits$abund$mode))
  w <- ifelse(y > 0, 1, psi * exp(-lam) / (psi * exp(-lam) + (1 - psi)))
  list(weights = w)
}
m_step_encode <- function(weights, ...) {
  list(
    occ   = list(y = rep(c(1, 0), each = n), X = rbind(X, X),
```

```
                  weights = c(weights, 1 - weights), family = "binomial"),
    abund = list(y = y, X = X, family = "poisson", weights = weights)
  )
}

res <- tulpa_em_laplace(e_step, m_step_encode, verbose = FALSE)
res$converged
plogis(res$fits$occ$mode[1])    # P(non-structural), truth 0.7
exp(res$fits$abund$mode[1])     # abundance mean, truth 4
```

---

tulpa_em_mc *Generic Monte-Carlo EM driver*

Description

Same M-step plumbing as `tulpa_em_laplace()`: every iteration calls `m_step_encode(weights, ...)` to assemble per-submodel blocks and fits each block via `tulpa_laplace()`. The difference is the E-step: instead of computing closed-form weights, an `e_step_sample` callback returns `n_mc` weight draws per iteration. Each draw is run through the M-step independently and the resulting parameter estimates are pooled via `rubins_pool()`.

Convergence criterion is the max relative change in *pooled* M-step parameter estimates between iterations. To increase Monte-Carlo accuracy as iterations progress (Booth-Hobert ascent-based MCEM), set `n_mc_growth` > 1.

This is an engine block, not a front door: model packages call it programmatically, so its tuning knobs (`max_iter`, `tol`, `n_mc`) sit in the signature rather than in a `control` list.

Usage

```
tulpa_em_mc(
  e_step_sample,
  m_step_encode,
  n_mc = 10L,
  n_mc_growth = 1,
  n_mc_max = 200L,
  max_iter = 30L,
  tol = 0.001,
  verbose = TRUE,
  ...
)
```

Arguments

`e_step_sample`	Function `function(fits, n_mc, ...) -> list`. Must return a list of length `n_mc`, each element a weights object of the same shape `m_step_encode` consumes (typically a numeric vector of length `n_obs`, or a matrix `n_obs x K` for K-class latent variables). On the first iteration `fits` is `list()` – the callback should return draws from the prior.

m_step_encode	Function function(weights, ...) -> list of blocks. Identical to the contract used by tulpa_em_laplace(): each block is a list with required fields y, X, family, and optional n_trials, offset, phi, re_list, spatial, weights. See ?tulpa_em_laplace for the full spec.
n_mc	Initial number of Monte-Carlo draws per iteration (default 10L).
n_mc_growth	Multiplicative growth of n_mc per iteration (default 1.0 = constant). Set > 1 for ascent-based MCEM that ramps up MC accuracy near convergence.
n_mc_max	Cap on n_mc (default 200L).
max_iter	Maximum EM iterations (default 30L).
tol	Convergence tolerance on max relative change in pooled parameter estimates (default 1e-3). Looser than tulpa_em_laplace default because Monte-Carlo noise floors the achievable precision.
verbose	Print per-iteration progress (default TRUE).
...	Forwarded to e_step_sample and m_step_encode.

Value

A list with:

- pooled – named list of pooled per-submodel summaries (mean, se, V_within, V_between, V_total); see rubins_pool().
- fits – list of per-MC-draw fit lists from the *final* iteration, each indexed by submodel name.
- n_iter – iterations actually run.
- n_mc_final – n_mc value used in the last iteration.
- converged – logical.
- history – data.frame(iter, n_mc, delta).

Tier

Inherits the tier of the inner M-step (Laplace => Tier 2). The Monte-Carlo E-step *itself* is exact in the limit n_mc -> infinity, so as a *full pipeline* MCEM is asymptotically Tier 1 if e_step_sample is exact.

References

Wei, G. C. G., & Tanner, M. A. (1990). A Monte Carlo implementation of the EM algorithm and the poor man's data augmentation algorithms. *Journal of the American Statistical Association*, 85(411), 699-704.

Booth, J. G., & Hobert, J. P. (1999). Maximizing generalized linear mixed model likelihoods with an automated Monte Carlo EM algorithm. *JRSS B*, 61(1), 265-285.

See Also

tulpa_em_laplace() for the closed-form-weights variant, rubins_pool() for the pooling rule.

Examples

```
# Two-component Gaussian mixture with unit variances: the latent class z is
# sampled in the E-step, and each class mean is an intercept-only Gaussian
# block in the M-step.
set.seed(1)
z <- rbinom(100, 1, 0.4)
y <- ifelse(z == 0, rnorm(100, -1), rnorm(100, 2))

e_step_sample <- function(fits, n_mc, ...) {
  p1 <- if (length(fits) == 0L) {
    ifelse(y > median(y), 0.8, 0.2)
  } else {
    mu0 <- fits[[1]]$comp0$mode[1]
    mu1 <- fits[[1]]$comp1$mode[1]
    l0 <- dnorm(y, mu0, log = TRUE) + log(0.6)
    l1 <- dnorm(y, mu1, log = TRUE) + log(0.4)
    1 / (1 + exp(l0 - l1))
  }
  lapply(seq_len(n_mc), function(s) rbinom(length(y), 1, p1))
}
m_step_encode <- function(weights, ...) {
  list(comp0 = list(y = y[weights == 0], X = matrix(1, sum(weights == 0), 1),
                    family = "gaussian"),
       comp1 = list(y = y[weights == 1], X = matrix(1, sum(weights == 1), 1),
                    family = "gaussian"))
}

fit <- tulpa_em_mc(e_step_sample, m_step_encode, n_mc = 5L, max_iter = 5L,
                   tol = 5e-3, verbose = FALSE)
sapply(fit$pooled, function(p) p$mean)
fit$history
```

`tulpa_ep` *Expectation-Propagation fit for a GLM*

Description

Fits a generalized linear model with a Gaussian coefficient prior by Expectation Propagation: the posterior is approximated by a Gaussian whose per-observation sites match the moments of the tilted distribution (via Gauss-Hermite quadrature). EP is exact for a Gaussian likelihood and typically more accurate than Laplace on skewed likelihoods, since it matches marginal moments rather than the mode curvature.

Usage

```
tulpa_ep(
  formula,
  data,
```

```
  family = "binomial",
  phi = 1,
  phi2 = NULL,
  n_trials = NULL,
  beta_prior = .tulpa_default_beta_prior("ep"),
  control = list()
)
```

Arguments

formula	Model formula.
data	A data frame.
family	Character family name (see family_names()).
phi	Dispersion passed to the family, held fixed. One convention at every door: for gaussian / lognormal this is the residual VARIANCE (the SD is sqrt(phi)), for neg_binomial_2 the size, gamma the shape, beta the precision, t the scale; binomial and poisson ignore it. The compiled kernels parameterize the two variance families by the residual SD and are handed sqrt(phi) at the boundary. Defaulted, it conditions at 1 and says so with a warning, since for a family that reads a dispersion that is a modelling choice rather than a neutral value. fit$phi_estimated records whether the value on the fit was estimated or conditioned on.
phi2	Optional second dispersion (Student-t degrees of freedom for family = "t"; default 4 when NULL).
n_trials	Binomial denominators (length nrow(data)), or NULL (= 1).
beta_prior	Fixed-effect prior as list(mean, sd): a mean-zero (mean = 0) Gaussian on every coefficient with SD sd (default the engine default, prior_normal(0, 2.5)). EP's site parameterisation assumes a mean-zero coefficient prior, so a non-zero mean errors – use a sampler (mode = "mala") for a shifted prior.
control	List: max_sweeps (default 50), tol (default 1e-6), damping (default 0.8), n_quad (Gauss-Hermite nodes, default 20), n_draws (default 2000), seed. An offset(...) term in formula is honoured: it shifts each observation's linear predictor before the likelihood is evaluated, and is not itself part of the fitted posterior.

Value

A tulpa_fit with means (posterior mean) and cov (posterior covariance) of the EP Gaussian, which is the posterior coef(), vcov(), summary.tulpa_fit() and confint() report; draws sampled from that Gaussian; log_marginal (the EP approximation); converged (the sweep tolerance was met AND every site's adaptive quadrature mode search converged AND no site's tilted variance floored – a site that is not well resolved by the quadrature turns this FALSE even when the outer sweep loop itself settled); n_site_not_converged, n_site_floored.

References

Minka (2001). Expectation Propagation for approximate Bayesian inference. UAI. Rasmussen & Williams (2006). Gaussian Processes for Machine Learning, Algorithm 3.5.

See Also

tulpa() (Laplace / sampler tiers), pathfinder() (VI).

Examples

```
set.seed(1)
d <- data.frame(x = rnorm(200))
d$y <- rbinom(200, 1, plogis(-0.3 + 0.8 * d$x))
fit <- tulpa_ep(y ~ x, data = d, family = "binomial")
coef(fit)
```

---

tulpa_family *Construct a minimal tulpa_family for simulation*

Description

Lightweight constructor for a tulpa_family object that exposes the contract required by prior_predict() and tulpa_simulate(). Model packages (tulpaRatio, tulpaObs) register richer families that also link to C++ likelihoods; this helper is for tests and simple custom families that only need simulation.

Usage

```
tulpa_family(
  name,
  simulate_fn,
  process_names = "y",
  extra_params = list(),
  link_inv = NULL
)

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

Arguments

name	Family name (character, length 1).
simulate_fn	function(eta, params, n_obs, ...) returning a numeric vector of length n_obs (single-process) or a list of such vectors keyed by process_names (multi-process). eta is a list of linear predictors, one per process.
process_names	Character vector. Defaults to "y" (single-process).
extra_params	Named list of tulpa_prior objects for likelihood- specific scalar parameters (e.g., dispersion phi). Drawn at each prior-predictive iteration. Defaults to empty.

link_inv	List of inverse-link functions per process; defaults to identity for every process. tulpa passes the raw linear predictor to simulate_fn, so most families implement the link inside simulate_fn (e.g., mu = exp(eta) for Poisson). The link_inv slot exists for families that prefer to keep the inverse-link separate.
x	A tulpa_family.
...	Ignored.

Value

A tulpa_family object.

Examples

```
fam <- tulpa_family(
  name = "poisson",
  simulate_fn = function(eta, params, n_obs, ...) {
    rpois(n_obs, exp(eta[[1]]))
  }
)
```

---

tulpa_fit_summary *Build the summary of a tulpa fit around a coefficient table*

Description

The constructor summary.tulpa_fit() returns through: everything but the fixed-effect table is read off the fit the same way for every model. A model package whose fit stores its coefficients in its own layout (several arms, no draws) computes that table and hands it here, so its summary() is the same object, printed and read the same way.

Usage

```
tulpa_fit_summary(object, coefficients, level = 0.95)
```

Arguments

object	A tulpa_fit.
coefficients	Data frame, one row per coefficient (row names are the coefficient names), with columns estimate, std.error and the two interval bounds, in that order.
level	Coverage of the interval columns.

Value

A summary.tulpa_fit object, as described on summary.tulpa_fit().

See Also

summary.tulpa_fit()

Examples

```
set.seed(1)
df <- data.frame(x = rnorm(60))
df$y <- rpois(60, exp(0.5 + 0.3 * df$x))
fit <- tulpa(y ~ x, data = df, family = "poisson", mode = "laplace")
tab <- coef(summary(fit))
identical(coef(tulpa_fit_summary(fit, tab)), tab)
```

---

tulpa_formula — *Formula parsing for tulpa models*

Description

Parses mixed-model formulas by walking the formula's abstract syntax tree. R formulas are already parse trees – we do structural recursion to find random effect terms (`|` nodes) and separate them from fixed effects.

Value

The formula helpers documented in this family return parsed-formula structures (lists describing the fixed effects, random-effect terms, and latent blocks); see each function's own help page.

---

tulpa_gaussian — *Fit a Gaussian linear model via tulpa's generic engine*

Description

Proof-of-concept function demonstrating the tulpa generic interface: the Gaussian likelihood is supplied as a `LikelihoodSpec` and fitted by the production NUTS sampler every other ModelData backend runs (dual-averaged step size, adapted diagonal mass matrix, reverse-mode AD gradient checked against finite differences before sampling). Fits `y ~ Normal(X * beta, sigma)`.

Usage

```
tulpa_gaussian(
  formula,
  data,
  beta_prior = .tulpa_default_beta_prior("gaussian"),
  control = list()
)
```

Arguments

formula	A formula (e.g., y ~ x1 + x2)
data	A data frame
beta_prior	Fixed-effect prior as `list(mean, sd)`: a mean-zero (`mean = 0`) Gaussian on every coefficient with SD `sd` (default the engine default, `prior_normal(0, 2.5)`).
control	List of sampler knobs: `iter` (total iterations, default 2000), `warmup` (default 1000), `max_treedepth` (default 10), `adapt_delta` (target acceptance statistic, default 0.8), `seed` (`NULL` draws from the session RNG).

Value

A `tulpa_fit` list with `draws` (post-warmup draws of `beta[j]` and `log_sigma`), `means`, `sigma` (`exp()` of the posterior mean of `log_sigma`), `accept_rate` (the mean NUTS acceptance statistic over the post-warmup iterations), the per-iteration `accept_prob`, `divergent`, `treedepth` and `log_prob`, the adapted step size `epsilon`, and `formula`, `N`, `p`.

Examples

```
set.seed(1)
d <- data.frame(x = rnorm(40))
d$y <- 1 + 2 * d$x + rnorm(40, 0, 0.6)
fit <- tulpa_gaussian(y ~ x, d, control = list(iter = 1000, warmup = 500,
                                               seed = 1))
fit$means
fit$accept_rate
```

---

tulpa_gibbs *Fit via Polya-Gamma Gibbs sampler*

Description

Public API for PG Gibbs sampling. Used by model packages for binomial and negative binomial GLMMs.

Usage

```
tulpa_gibbs(
  y,
  n_trials,
  X,
  group,
  n_groups,
  family = "binomial",
  beta_prior = .tulpa_default_beta_prior("gibbs"),
  prior_sigma_scale = 2.5,
  spatial = NULL,
```

```
  temporal = NULL,
  control = list()
)
```

Arguments

y	Response vector
n_trials	Trial sizes (binomial)
X	Design matrix
group	Integer vector of group indices (1-based)
n_groups	Number of groups
family	Character: "binomial" or "neg_binomial_2"
beta_prior	Fixed-effect prior as `list(mean, sd)`: a mean-zero (`mean = 0`) Gaussian on every coefficient with SD `sd` (default the engine default, `prior_normal(0, 2.5)`). The Polya-Gamma sampler uses a mean-zero prior, so a non-zero `mean` errors.
prior_sigma_scale	Prior scale for RE sigma (statistical; default 2.5).
spatial	Optional spatial spec. When supplied the fit routes to the matching spatial Polya-Gamma Gibbs sampler via `dispatch_gibbs_spatial()`; `group`/`n_groups` are the iid random-effect block carried alongside the field. The full areal + continuous family is available for `family = "binomial"`; `family = "neg_binomial_2"` is backed by the areal ICAR negbin sampler only. Supported `types`: • areal – `"icar"`, `"bym2"`, `"rsr"`: a list with `type`, `adjacency` and a 1-based `spatial_idx` per observation (e.g. `list(type = "icar", adjacency = W, spatial_idx = unit)`). `"rsr"` reuses `spatial$rsr_projection` if present, else builds the unit-level projector from the design. • continuous – `"gp"`/`"nngp"` (a validated `spatial_gp()` spec) and `"multiscale_gp"` (a validated `spatial_multiscale()` spec). These samplers carry no observation->location map, so they require one observation per unique location in coordinate order.
temporal	Optional temporal spec: a validated `temporal_multiscale()` object. Routes to the multiscale temporal Polya-Gamma sampler via `dispatch_gibbs_temporal()` (binomial only; RW1 trend + cyclic seasonal + AR1/IID short-term). Cannot be combined with `spatial`.
control	A named list of numerical / tuning knobs (statistical arguments stay in the signature above): `n_iter` (default 2000), `warmup` (default 1000), `thin` (default 1, applied on every route including the spatial and temporal ones; the run keeps `ceiling((n_iter - warmup) / thin)` draws), `seed` (`NULL` draws from the session RNG; the Polya-Gamma kernels use R's RNG, so a seed makes the fit reproducible), `verbose` (default FALSE), `n_threads` (default 1).

Details

For `family = "neg_binomial_2"` the Polya-Gamma weights are drawn at the exact real shape `PG(y + r, eta)` and the dispersion `r` is updated by a random-walk Metropolis-Hastings step on `log(r)`

whose stationary support is bounded to `r` in `[0.1, 500]`; data favouring a dispersion outside that range pile up at the boundary.

Every sampler that moves a latent effect's level through the first coefficient – the negative-binomial kernels, and the binomial kernels carrying a `spatial` or `temporal` field – leaves `eta` unchanged only when the first column of `X` is an all-ones intercept, and those routes error on a design without one. An intrinsic field (ICAR, the structured BYM2 part, RW1) is reported centred with its level in the intercept, and its sweep carries the intercept's prior through the field mean; a proper one (the negative-binomial iid block, an NNGP field) has the level it shares with the intercept drawn from its full conditional, which both priors define.

Value

A `tulpa_fit` holding one MCMC chain: `draws`, the `[n_saved x n_param]` matrix of retained draws, with `chain_id`, `n_chains`, `means` (column means) and `param_names`. Columns follow the parameter naming of the ModelData samplers: the fixed effects (the column names of `X`, else `beta[j]`), `log_sigma_re` and `re[g]` for the random-intercept block, then the route's own blocks – `log_phi` (the negative-binomial size); `log_tau_spatial` and `phi_spatial[u]` (ICAR, and RSR's projected field); `log_sigma_spatial`, `logit_rho_bym2`, `phi_spatial[u]` and `theta_spatial[u]` (BYM2); `log_sigma2_gp`, `log_phi_gp` and `gp_w[u]` (GP); the `_local` / `_regional` counterparts with `gp_local[u]` / `gp_regional[u]` (multiscale GP); `log_sigma2_trend`, `trend[t]`, `log_sigma2_seasonal`, `seasonal[s]`, `log_sigma2_short`, `short_term[t]` and, for an AR1 short-term component, `logit_rho_short` (temporal). Scales are stored on the log scale and correlations on the logit scale, as the samplers store them; a component the model does not carry has no column.

Examples

```
set.seed(1)
G <- 20L; npg <- 15L; n <- G * npg
grp <- rep(seq_len(G), each = npg)
X <- cbind(1, rnorm(n))
b <- rnorm(G, 0, 0.6)
y <- rbinom(n, 1, plogis(X %*% c(-0.2, 0.5) + b[grp]))

fit <- tulpa_gibbs(y, rep(1L, n), X, grp, G, family = "binomial",
                   control = list(n_iter = 500L, warmup = 250L))
coef(fit)
diagnostics(fit)
```

`tulpa_grid_axis` *Read a default nested-Laplace outer-grid axis*

Description

Materialises one of tulpa's own default outer-grid axes – the values a `*_grid` argument (e.g. `sigma_grid`, `range_grid`) takes when a caller does not supply one. Geometric axes are returned as `exp(seq(log(lo), log(hi), length.out = n))`; axes declared as explicit nodes are returned as-is. A default is a decision tulpa makes internally and can change between releases (gcol33/tulpa#633), so a caller that

needs to know the current default – rather than restate it – should read it here instead of duplicating tulpa's own grid table.

Usage

```
tulpa_grid_axis(key, n = NULL)
```

Arguments

key	Name of a default axis, e.g. "field_sd", "copy_alpha", "bym2_rho", "gp_var", "gp_lengthscale". Unknown keys error.
n	Optional resolution override: same lo / hi bounds (and the same prepend atom, if any) at a different node count. NULL uses the axis's own declared n. Errors for an axis declared as explicit nodes (no resolution to vary) or for a data-dependent axis (its shape depends on arguments the table does not hold).

Value

Numeric vector of grid nodes.

Examples

```
tulpa_grid_axis("field_sd")
tulpa_grid_axis("copy_alpha", n = 9)
```

---

tulpa_grid_log_quad *Per-cell log quadrature weight of an outer grid*

Description

The per-cell log quadrature weight (prior mass) of a nested-Laplace outer grid: every path that turns a fit's log_marginal into posterior cell weights goes through this one rule, so the prior mass a cell carries is decided in one place. Rebuilds axis specs from the grid's own columns when specs is not supplied. Intended for reconstructing a fit's outer-grid posterior weights (with tulpa_theta_matrix() and tulpa_normalise_weights_safe()) when the fit doesn't already carry fit$log_quad.

Usage

```
tulpa_grid_log_quad(
  theta_grid,
  specs = NULL,
  copy_slab = "exponential",
  close_domain = TRUE,
  folded_axes = NULL,
  refining = NULL
)
```

Arguments

theta_grid	A named [n_cells x n_axes] matrix (see tulpa_theta_matrix()).
specs	Optional pre-built per-axis spec list; NULL rebuilds it from theta_grid's columns via tulpa_joint_axis_specs_from_grid().
copy_slab	"exponential" or "flat"; the copy-scale axis's continuum measure (see ?tulpa_joint_axis_specs_f
close_domain	Whether an unbounded axis's outer cells are closed at the grid's own edge rather than left open to infinity.
folded_axes	Optional names of axes folded onto [0, Inf) (e.g. a correlation axis reflected at 0).
refining	Optional refinement tag vector (see tulpa_hyper_slice_home()); when supplied, specs are rebuilt from the grid's declared cells only.

Value

Numeric vector of per-cell log quadrature weights, length nrow(theta_grid), or NULL if theta_grid has no axis names.

See Also

tulpa_theta_matrix(), tulpa_normalise_weights_safe(), tulpa_joint_axis_specs_from_grid()

Examples

```
set.seed(1)
S <- 30L                                  # spatial units in a chain
nb <- lapply(seq_len(S), function(s) setdiff(c(s - 1L, s + 1L), c(0L, S + 1L)))
nn <- lengths(nb)
field <- as.numeric(scale(cumsum(rnorm(S, 0, 0.4))))
idx <- rep(seq_len(S), each = 6L); n <- length(idx); x <- rnorm(n)
y <- rbinom(n, 1L, plogis(-0.2 + 0.6 * x + field[idx]))
prior <- list(type = "icar", n_spatial_units = S, spatial_idx = idx,
              adj_row_ptr = c(0L, cumsum(nn)), adj_col_idx = unlist(nb) - 1L,
              n_neighbors = nn, tau_grid = c(0.5, 1, 2, 4, 8))
fit <- tulpa_nested_laplace(y, rep(1L, n), cbind(1, x), prior = prior,
                            family = "binomial",
                            control = list(progress = FALSE))
tg <- tulpa_theta_matrix(fit)
lq <- tulpa_grid_log_quad(tg)
# The fit's own cell weights, rebuilt from its log marginals:
tulpa_normalise_weights_safe(fit$log_marginal, log_quad = lq)
```

tulpa_hyper_check_copy_slab

Validate/default a copy-scale slab measure choice

Description

Validates a `copy_slab` argument – `"exponential"` (the default) or `"flat"`, the two continuum measures tulpa supports for a copy scale's non-atom mass – defaulting `NULL` to `"exponential"` and erroring on anything else. Intended so a consumer package's own `copy_slab` argument stays in sync with tulpa's own accepted choices rather than restating them.

Usage

```
tulpa_hyper_check_copy_slab(x)
```

Arguments

`x`	A `copy_slab` value: `NULL`, `"exponential"`, or `"flat"`.

Value

`x`, defaulted to `"exponential"` when `NULL`.

See Also

`tulpa_hyper_copy_slab_density()`

Examples

```
tulpa_hyper_check_copy_slab("exponential")
try(tulpa_hyper_check_copy_slab("uniform"))
```

tulpa_hyper_copy_slab_density

The copy-scale axis's PC-prior density

Description

The proper density tulpa uses for the copy scale's continuum: an exponential, the penalized-complexity prior for a scale parameter with its base model at zero (Simpson et al. 2017). The rate is set by putting 5% of the prior mass above `upper`, so a caller that reads this off a fit's own declared grid gets the exact rate the outer integration used – rather than restating a PC-prior rate that could silently drift from it.

Usage

```
tulpa_hyper_copy_slab_density(upper)
```

Arguments

upper	The largest declared node of the copy-scale axis.

Value

A function `log p(x) = log(lambda) - lambda * x`, or `NULL` if `upper` is not a finite positive number.

See Also

`tulpa_hyper_check_copy_slab()`, `tulpa_joint_axis_specs_from_grid()`

Examples

```
lp <- tulpa_hyper_copy_slab_density(upper = 2)
lp(c(0.1, 1, 2))
```

---

tulpa_hyper_draws *Hyperparameter draws from a nested-Laplace fit*

Description

The hyperparameter half of a draw from the outer-grid mixture: one row per draw, one column per outer-grid axis. A draw's coordinate on each axis is sampled from the within-cell density its cell carries under the fit's own within-cell read, so the columns are a continuous marginal rather than the handful of grid-node values `fit$theta_grid[cells, ]` returns.

An axis carrying a declared POINT MASS – the copy scale's `alpha = 0`, the "no coupling" model whose prior probability is stated rather than read off a node count – keeps it: draws in that cell are exactly the level, in the proportion the fit reports as `fit$copy_atom$posterior_mass`, and the continuum above it is continuized on its own partition. The level is a model and not a cell representative, so spreading it over a box would both put mass where the axis has none and leave none on the level itself.

Reading the node coordinate directly is what `tulpa_posterior_draws()` consumers used to do, and on a 5- to 15-node axis it makes every draw an atom: a truth between two nodes, or past the outermost one, has no draw that can land near it. The continuization is the cell-conditional of the SAME construction the fit reports its interval from (`box_uniform` by default, `chord` when the fit asked for it or its support does not admit the box read), so the draws reproduce the fit's own `theta_ci_lo` / `theta_median` / `theta_ci_hi` to Monte Carlo error.

Usage

```
tulpa_hyper_draws(fit, cells = NULL, n = 1000, within = NULL)
```

Arguments

`fit`	A nested-Laplace fit (`tulpa_nested_laplace()` or `tulpa_nested_laplace_joint()`).
`cells`	Integer vector of outer-grid cell indices, one per draw – the `"cells"` attribute of a `tulpa_posterior_draws()` matrix, so the hyperparameter row and the latent row of a draw come from the same cell. `NULL` (default) allocates `n` fresh draws across the cells by weight.
`n`	Number of draws when `cells` is `NULL` (default 1000); ignored otherwise.
`within`	Within-cell construction, `"box_uniform"`, `"log_quadratic"` or `"chord"`. Under `"log_quadratic"` a draw in a cell is the point its row's reconstructed CDF reaches inside the share of that CDF the cell's mass owns, so the draws in one cell stay around it. `NULL` (default) takes the one the fit was read with (`fit$within_cell_requested`).

Value

A numeric matrix `[n x n_axes]` with the outer grid's axis names as columns, or `NULL` when the fit carries no outer-grid axis. Carries `attr(., "cells")` – the cell each row's coordinates were drawn in – plus `attr(., "within_cell")` and `attr(., "within_cell_declined")`, the per-axis construction that ran and why a requested one did not, and `attr(., "within_cell_copula")`, the Gaussian-copula correlation matrix that ties the axes' within-cell draws together so the draws keep the grid's own correlation between axes (the identity where the grid's axes are uncorrelated). Each axis's marginal is the same whatever the copula.

See Also

`tulpa_posterior_draws()`, `tulpa_nested_laplace()`

Examples

```
set.seed(1)
S <- 30L                                    # spatial units in a chain
nb <- lapply(seq_len(S), function(s) setdiff(c(s - 1L, s + 1L), c(0L, S + 1L)))
nn <- lengths(nb)
field <- as.numeric(scale(cumsum(rnorm(S, 0, 0.4))))
idx <- rep(seq_len(S), each = 6L); n <- length(idx); x <- rnorm(n)
y <- rbinom(n, 1L, plogis(-0.2 + 0.6 * x + field[idx]))
prior <- list(type = "icar", n_spatial_units = S, spatial_idx = idx,
              adj_row_ptr = c(0L, cumsum(nn)), adj_col_idx = unlist(nb) - 1L,
              n_neighbors = nn, tau_grid = c(0.5, 1, 2, 4, 8))
fit <- tulpa_nested_laplace(y, rep(1L, n), cbind(1, x), prior = prior,
                            family = "binomial",
                            control = list(progress = FALSE))
th <- tulpa_hyper_draws(fit, n = 200)
quantile(th[, "tau"], c(0.025, 0.5, 0.975))
```

tulpa_hyper_grid *Outer hyperparameter-grid integration with a user-supplied inner fit*

Description

Generic driver for nested-Laplace-style outer integration over a small hyperparameter block. The user supplies per-axis specs (values + optional log-prior + log-scale / bounds / refinable metadata) and an `inner_fit` callback that, at every hyperparameter cell, returns the inner log marginal and optionally the fixed-effect posterior mode + marginal covariance. The driver builds the Cartesian outer grid, normalises the log-marginals to integration weights, reports per-axis posterior moments and weighted quantiles, and (when the inner fit supplies them) law-of-total-covariance fixed-effect posterior + mixture draws.

This factors the per-family outer-grid plumbing in `tulpa_nested_laplace()` / `tulpa_nested_laplace_joint()` / `tulpa_re_cov_nested()` into one callback-driven entry point: downstream consumers (occupancy / N-mixture / cover hurdle families in tulpaObs, custom user families) drop in their own per-cell inner fit and get the outer integration for free.

Usage

```
tulpa_hyper_grid(
  hyper_specs,
  inner_fit,
  combine = c("law_of_total_cov", "weighted_mean_only", "none"),
  n_draws = 2000L,
  seed = NULL,
  beta_names = NULL,
  control = list()
)
```

Arguments

`hyper_specs` A list of axis specs. Each spec is either a `hyper_axis_spec()` object or a plain list with the same fields (the driver auto-wraps); see `hyper_axis_spec()`. The outer grid is the Cartesian product of the per-axis grids; the joint log-prior is the sum of per-axis `log_prior` contributions (axes with `log_prior = NULL` are flat / improper).

`inner_fit` `function(hypers)` returning a list with:

- `log_marginal` – scalar; the inner-solve log marginal at this cell. Non-finite values are mapped to `-Inf` (the cell gets zero weight).
- `beta_mean` – numeric vector of fixed-effect posterior mean at the cell. Required when `combine != "none"`.
- `beta_cov` – `p x p` numeric matrix of fixed-effect marginal covariance at the cell. Required when `combine = "law_of_total_cov"`. `hypers` is a named numeric vector with the current cell's axis values (names match `vapply(hyper_specs, [[, character(1), "name")`). Errors thrown by `inner_fit` are caught and treated as a failed cell (`log_marginal = -Inf`, no beta contribution).

`combine` How to pool per-cell fixed-effect posteriors into a single posterior over the betas. One of:

- `"law_of_total_cov"` (default) – compute `E[Cov(beta | theta)] + Cov(E[beta | theta])` from the per-cell `(beta_mean, beta_cov)`; synthesise `n_draws` posterior draws by mixture sampling. Requires `beta_mean` and `beta_cov` per cell.
- `"weighted_mean_only"` – pool only the per-cell `beta_mean` into the weighted posterior mean. `beta_cov` is ignored; no draws.
- `"none"` – do not assemble a fixed-effect posterior. Only the hyperparameter posterior is returned. `beta_mean` / `beta_cov` from `inner_fit` are ignored if supplied.

`n_draws` Number of posterior draws of the fixed effects to synthesise from the cell mixture (default 2000). Used only when `combine = "law_of_total_cov"`. `0` disables draw synthesis (the law-of-total-cov mean and covariance are still returned).

`seed` Optional integer seed for the draw synthesis.

`beta_names` Optional character vector naming the fixed-effect coordinates. When `NULL` (default) the names are taken from the first successful cell's `beta_mean` (or `beta1`, `beta2`, ... if it is unnamed).

`control` Optional list of refinement / tuning knobs:

- `adaptive_grid` (`FALSE`) – run the boundary / interior refinement pass on every axis whose spec has `refinable = TRUE`. New cells are appended along the refining axis paired with the boundary modal cell's other-axis values. Each new cell is measured by the part of its row's base cells it takes over, so the grid integrates the same prior measure with a finer resolution where the mass is.
- `adaptive_grid_edge_thresh` (`0.02`) – per-axis trigger threshold.
- `adaptive_grid_max_passes` (`1L`) – cap on refinement passes.
- `var_of_means_consistency` (`FALSE`) – run a post-integration consistency pass: for refinable axes whose marginal has collapsed onto too few nodes to carry a spread, bisect the gaps between adjacent nodes that carry the axis's mass, each new point a level laid in every row of the other axes that holds the posterior, and repeat until the axis reaches `var_of_means_min_ess` or has taken `.nl_diag("axis_refine_nodes")` new levels. One kernel call per round.
- `var_of_means_min_ess` (`.nl_diag("axis_sd_ess")`) – the quadrature effective sample size an axis marginal has to reach for the pass to leave it alone. Read off the weights, so the trigger is not one SD estimator compared against the other.

Value

A list of class `c("tulpa_fit", "list")` with:

- `theta_grid` – numeric matrix `[n_cells x n_axes]` of outer-grid hyperparameter values; columns named after the axes.

- theta_names – character vector of axis names.
- log_marginal – numeric [n_cells]; per-cell log integrand inner$log_marginal + log_prior(theta_cell). Non-finite / failed cells are -Inf.
- log_prior – numeric [n_cells]; per-cell log-prior contribution (the sum across axes of axis$log_prior(theta_cell[axis])). 0 when all axes have log_prior = NULL.
- weights – numeric [n_cells] summing to 1 (or all NA with a warning when no cell carries finite mass).
- theta_mean, theta_sd – named numeric vectors; weighted posterior mean and SD per axis. Each SD comes from the estimator its axis's own resolution calls for – the weighted spread of the axis marginal where it is resolved, the 3-point parabola at the modal node where the marginal has collapsed onto too few nodes to have a spread.
- theta_sd_source, theta_sd_ess, theta_sd_stencil_declined – per axis, which estimator produced the SD, the quadrature effective sample size that decided it, and (where the parabola was wanted and could not be formed) why it declined.
- theta_median, theta_ci_lo, theta_ci_hi – named numeric vectors; weighted-quantile median and 2.5 / 97.5\ (the recommended summary for right-skewed scale-like axes).
- beta, beta_cov, draws – fixed-effect posterior, present per combine:
  - combine = "law_of_total_cov": weighted mean, total covariance (E[V] + V[E]), [n_draws x p] mixture draws.
  - combine = "weighted_mean_only": weighted mean only; beta_cov and draws are NULL.
  - combine = "none": all three are NULL.
- means, param_names, process_info, n_samples, n_params, N – the tulpa_fit accessor surface, populated when a fixed-effect posterior is assembled.
- hyper_specs – echoed list of hyper_axis_spec objects (the normalised form).
- combine, n_failed, n_grid, refining_axis – diagnostic fields.
- adaptive_grid_info, var_of_means_consistency_info – present when the corresponding refinement pass fired.

See Also

hyper_axis_spec() for the axis-spec constructor; tulpa_nested_laplace_joint() for the family-specific outer-grid driver that this helper generalises.

Examples

```
# Integrate a Gaussian linear model's residual SD: inner_fit(hypers) returns
# the per-cell log marginal and the conditional fixed-effect posterior, and
# the axis spec names the grid. tulpa_nested_laplace() is the packaged driver
# built on this.
set.seed(1)
n <- 40L; X <- cbind(1, rnorm(n)); y <- as.numeric(X %*% c(0.5, -1) + rnorm(n))
XtX <- crossprod(X); Xty <- crossprod(X, y)
inner_fit <- function(hypers) {
  sigma <- as.numeric(hypers["sigma"])
  V <- chol2inv(chol(XtX / sigma^2 + diag(1e-4, 2)))
  mu <- as.numeric(V %*% (Xty / sigma^2))
```

```
  list(log_marginal = -n * log(sigma) - 0.5 * sum((y - X %*% mu)^2) / sigma^2 +
         0.5 * as.numeric(determinant(V)$modulus),
       beta_mean = stats::setNames(mu, c("b0", "b1")), beta_cov = V)
}
specs <- list(hyper_axis_spec("sigma", grid = exp(seq(log(0.5), log(2),
                                                      length.out = 9)),
                              log_scale = TRUE, bounds = c(0, Inf)))
res <- tulpa_hyper_grid(specs, inner_fit, n_draws = 500L, seed = 1L)
res$theta_median
res$beta
```

---

tulpa_hyper_grid_supports

Per-axis integrated support of an outer grid

Description

The natural-scale support of every axis in `specs` that carries one, as a named list of intervals – the region each axis's outer-grid measure actually integrates, including levels a refinement pass added. Intended for recovering an axis's integrated span from a settled grid, e.g. so a sampled-hyperparameter prior can be derived from what the outer integration used rather than restated alongside it.

Usage

```
tulpa_hyper_grid_supports(theta_grid, specs, refining = NULL)
```

Arguments

`theta_grid`	A named `[n_cells x n_axes]` matrix (see `tulpa_theta_matrix()`).
`specs`	Per-axis spec list, e.g. from `tulpa_joint_axis_specs_from_grid()`.
`refining`	Optional per-cell refinement tag vector (see `tulpa_hyper_slice_home()`).

Value

A named list of natural-scale `c(lo, hi)` intervals, one per axis in `specs` that carries a declared coordinate, or `NULL` if `theta_grid` or `specs` is `NULL`.

See Also

`tulpa_joint_axis_specs_from_grid()`, `tulpa_hyper_slice_home()`

Examples

```
set.seed(1)
S <- 30L                                  # spatial units in a chain
nb <- lapply(seq_len(S), function(s) setdiff(c(s - 1L, s + 1L), c(0L, S + 1L)))
nn <- lengths(nb)
field <- as.numeric(scale(cumsum(rnorm(S, 0, 0.4))))
```

```
idx <- rep(seq_len(S), each = 6L); n <- length(idx); x <- rnorm(n)
y <- rbinom(n, 1L, plogis(-0.2 + 0.6 * x + field[idx]))
prior <- list(type = "icar", n_spatial_units = S, spatial_idx = idx,
              adj_row_ptr = c(0L, cumsum(nn)), adj_col_idx = unlist(nb) - 1L,
              n_neighbors = nn, tau_grid = c(0.5, 1, 2, 4, 8))
fit <- tulpa_nested_laplace(y, rep(1L, n), cbind(1, x), prior = prior,
                            family = "binomial",
                            control = list(progress = FALSE))
tg <- tulpa_theta_matrix(fit)
tulpa_hyper_grid_supports(tg, tulpa_joint_axis_specs_from_grid(tg))
```

`tulpa_hyper_slice_home`

Per-cell refinement tag of an outer grid

Description

The axis whose refinement pass added each cell of a nested-Laplace outer grid, `""` for a cell of the grid as declared. A pass adds levels to an axis, laid in every row of the others that holds the posterior; the tag tells the declared levels apart wherever a reader needs them alone, e.g. rebuilding axis specs from a grid's declared nodes (see `tulpa_joint_axis_specs_from_grid()`), whose prior must not move with the nodes a pass added.

Usage

```
tulpa_hyper_slice_home(refining, n)
```

Arguments

`refining`	The `refining` tag vector stored on a fit (`NULL` for a grid with no refinement).
`n`	Number of grid cells; `refining`, if not `NULL`, must have this length.

Value

A character vector of length `n`: `""` for a declared cell, the axis name for a cell a refinement pass added.

See Also

`tulpa_hyper_grid_supports()`, `tulpa_joint_axis_specs_from_grid()`

Examples

```
tulpa_hyper_slice_home(NULL, 3)
tulpa_hyper_slice_home(c("", "", "sigma"), 3)
```

`tulpa_integrator` *Select the symplectic integrator for HMC and NUTS*

Description

Get or set the symplectic splitting integrator that the exact-MCMC tier (HMC / NUTS) uses to build its trajectory proposals. The integrator is backed by the SIMP library, which supplies leapfrog and the higher-order Yoshida members from one triple-jump composition.

Usage

```
tulpa_integrator(name, mts_substeps = 4L)
```

Arguments

`name`	Integrator name: `"leapfrog"` (default), `"minerror2"`, `"adaptive2"`, `"adaptive3"`, `"mts"`, `"yoshida4"`, `"yoshida6"`, or `"yoshida8"`. Omit to query the current selection without changing it.
`mts_substeps`	Number of inner prior-force substeps per trajectory step for the `"mts"` integrator (default 4). Ignored by the other integrators.

Details

The default, `"leapfrog"`, reproduces tulpa's historical trajectory step exactly.

`"minerror2"` is a two-stage order-two scheme whose coefficient is tuned to cancel the leading energy error on a Gaussian target. Since mass adaptation drives a posterior toward an isotropic Gaussian, it conserves energy well near the adapted optimum and adapts to larger step sizes, at two gradient evaluations per step. It is the recommended choice when leapfrog's step size is limited by energy error on a near-Gaussian posterior.

`"yoshida4"` is an order-4 scheme that also samples reliably (three gradient evaluations per step). `"yoshida6"` and `"yoshida8"` are available but experimental for NUTS: high-order composition integrators have a sharp step-size stability threshold that the dual-averaging adaptation pushes against, so `"yoshida6"` needs a high `adapt_delta` (0.95 or more) and `"yoshida8"` often fails to adapt. For sampling, prefer `"leapfrog"`, `"minerror2"`, or `"yoshida4"`.

`"adaptive2"` and `"adaptive3"` are step-adapted versions of the two- and three-stage schemes. Rather than fixing the coefficient in advance, each NUTS chain resolves it at the end of warmup for its own operating point: the coefficient that minimizes the worst-case energy error over the band of dimensionless steps the chain actually takes, read off from the adapted mass matrix and the local posterior curvature. Where `"minerror2"` is optimal only in the small-step limit, `"adaptive2"` tracks the chain's realised step band; `"adaptive3"` spends a third gradient per step to hold a small error over a wider band. Both run a fixed placeholder of the same stage family during warmup, so the dual-averaged step size carries over. Step-adaptation applies to NUTS (the default sampler); the fixed-trajectory HMC path uses the placeholder.

`"mts"` is a multiple-time-stepping (RESPA) integrator. It splits the force into a stiff but cheap prior part – the Gaussian latent structure – and a smooth but expensive likelihood part. Each NUTS trajectory step takes `mts_substeps` inner leapfrog substeps against the prior force while evaluating

the likelihood gradient only once, so the outer step can be larger without the stiff prior forcing it small. It pays one full gradient per step (as leapfrog does) plus `mts_substeps` cheap prior gradients, and helps most when the latent field is stiff relative to a comparatively flat likelihood. Like the other schemes it applies to NUTS.

The choice is process-global (like the gradient mode): set it once before fitting. It is read on the main thread at the start of sampling. Because it is process-global, a caller that changes it owns restoring it – and an error between the two calls would leave the process on the other integrator. `with_tulpa_integrator()` does both, restoring on error as well as on success.

Value

If `name` is omitted, the current integrator name. If `name` is given, the previous name is returned invisibly.

Examples

```
tulpa_integrator()            # current integrator
old <- tulpa_integrator("yoshida4")
tulpa_integrator(old)         # restore
```

`tulpa_is_spatial_bar` *Recognize an inline varying-coefficient bar*

Description

Test whether a one-sided formula carries a single varying-coefficient grouping bar of the form `~ 1 + w || node` (independent fields, `||`) or `~ 1 + w | node` (correlated fields, `|`). This is the same grammar `spatial()` and the inline `temporal()` field constructor accept, and the grammar `tulpa_bar_field_specs()` expands. A downstream package can use it to branch on "is this term a spatial / temporal varying-coefficient bar?" before calling `tulpa_bar_field_specs()`.

Usage

```
tulpa_is_spatial_bar(x)
```

Arguments

`x`	A one-sided formula (e.g. `~ 1 + w || node`) or the bar language object itself (`quote(1 + w || node)`).

Value

A single logical: `TRUE` when `x` is (or wraps) a `|` / `||` bar, `FALSE` otherwise (a plain formula such as `~ 1 + w`, a non-bar term, or a non-formula / non-language input).

See Also

`tulpa_bar_field_specs()` for expanding the bar into per-column field specs.

Examples

```
tulpa_is_spatial_bar(~ 1 + w || cell)   # TRUE
tulpa_is_spatial_bar(~ 1 + w | cell)    # TRUE (correlated)
tulpa_is_spatial_bar(~ 1 + w)           # FALSE (no bar)
```

---

tulpa_joint_axis_specs_from_grid

Rebuild axis specs from an already-assembled outer grid

Description

Rebuilds each axis's declared spec (support, log/linear coordinate, prior) from a `theta_grid`'s columns, keyed off column names – the same metadata the multi-block joint driver recovers rather than carrying a second description of the axes it already assembled. A column holding one value across every cell is treated as a fixed setting rather than an axis (e.g. a fixed NB-size node) and dropped. Intended for a caller that needs an axis's declared span, coordinate or prior off a settled grid without restating tulpa's own hyperprior declarations alongside it.

Usage

```
tulpa_joint_axis_specs_from_grid(
  theta_grid,
  copy_slab = "exponential",
  folded_axes = NULL,
  logchol = .hp_logchol_designs(theta_grid)
)
```

Arguments

`theta_grid`	A named `[n_cells x n_axes]` matrix (see `tulpa_theta_matrix()`).
`copy_slab`	`"exponential"` or `"flat"`; the copy-scale axis's continuum measure (see `tulpa_hyper_check_copy_sl`
`folded_axes`	Optional names of axes folded onto `[0, Inf)`.
`logchol`	Free-covariance block design(s) as returned by `.hp_logchol_designs()`; defaults to the design `theta_grid` itself declares.

Value

A named list of per-axis spec lists, or `NULL` if `theta_grid` has no axis names.

See Also

`tulpa_theta_matrix()`, `tulpa_grid_log_quad()`, `tulpa_hyper_grid_supports()`

Examples

```
set.seed(1)
S <- 30L                                  # spatial units in a chain
nb <- lapply(seq_len(S), function(s) setdiff(c(s - 1L, s + 1L), c(0L, S + 1L)))
nn <- lengths(nb)
field <- as.numeric(scale(cumsum(rnorm(S, 0, 0.4))))
idx <- rep(seq_len(S), each = 6L); n <- length(idx); x <- rnorm(n)
y <- rbinom(n, 1L, plogis(-0.2 + 0.6 * x + field[idx]))
prior <- list(type = "icar", n_spatial_units = S, spatial_idx = idx,
              adj_row_ptr = c(0L, cumsum(nn)), adj_col_idx = unlist(nb) - 1L,
              n_neighbors = nn, tau_grid = c(0.5, 1, 2, 4, 8))
fit <- tulpa_nested_laplace(y, rep(1L, n), cbind(1, x), prior = prior,
                            family = "binomial",
                            control = list(progress = FALSE))
specs <- tulpa_joint_axis_specs_from_grid(tulpa_theta_matrix(fit))
specs[[1]]$log_scale
```

`tulpa_joint_grid_batch`

Fit multiple joint nested-Laplace models through one fused grid solve

---

Description

Fits `B` responses that share one design (arms, latent blocks, outer grid layout, solver settings) and differ only in their responses, their arm dispersions and the nodes of any dispersion axis, through one fused grid solve (`cpp_nested_laplace_joint_multi_batch`): one design pass per outer- grid cell for all `B` responses, `B` block-diagonal Newton solves, instead of `B` independent fits each repeating the design pass.

Usage

```
tulpa_joint_grid_batch(fits)
```

Arguments

`fits`	A non-empty list of functions of no arguments, each performing one ordinary joint nested-Laplace fit.

Details

Each element of `fits` is a function of no arguments that performs one ordinary fit (a `tulpa_nested_laplace_joint()` call, or a consumer front door that makes one). The fits run twice: the first run stops each fit at its main outer-grid solve and keeps the kernel request; the fused solve answers every request at once; the second run replays each fit with its main grid solve served from the fused result. Every other kernel call a fit makes (placement, refinement, diagnostics) runs as it always does, so each returned fit is the object its ordinary fit would have returned, built by the same code.

The fits replay one after another from the random number state the batch was called at, so they draw the same stream the same fits called in sequence would draw; each capture starts from that state too, and a fit whose grid request depended on a draw taken before it is refused at its replay rather than served. Warnings and messages a capture run raises are muffled; the replay raises them again.

Raises a `tulpa_grid_batch_ineligible` condition (catchable with `tryCatch(..., tulpa_grid_batch_ineligible = ...)`) when the fits do not share a design or a request uses a setting the fused driver does not carry, and says so on the first fit's capture when the setting is the request's own. The cheap-pass screen is one: it prunes each response's grid to its own kept set, which one fused grid walk cannot, and it is on by default, so fits meant for the fused solve pass `control$prune = FALSE`. The random number state is restored to where the batch was called before that condition leaves, so a caller that falls back to fitting the responses one at a time draws the stream it would have drawn without the batch.

Value

A list of fit results, one per element of `fits`, in order.

See Also

`tulpa_nested_laplace_joint()`

Examples

```
# The fused solve serves coupled-arm families (a model package's joint
# occupancy-cover fit, say); for designs it does not carry it signals
# `tulpa_grid_batch_ineligible`, and the caller fits one at a time.
set.seed(1)
n_s <- 20L; N <- 100L
s <- sample.int(n_s, N, replace = TRUE); x <- rnorm(N)
nb <- lapply(seq_len(n_s), function(i) setdiff(c(i - 1L, i + 1L), c(0L, n_s + 1L)))
prior <- list(type = "icar", n_spatial_units = n_s,
              adj_row_ptr = as.integer(c(0L, cumsum(lengths(nb)))),
              adj_col_idx = as.integer(unlist(nb)) - 1L,
              n_neighbors = lengths(nb), sigma_grid = c(0.5, 1))
fit_one <- function(y) function() tulpa_nested_laplace_joint(
  responses = list(occ = list(y = y, n_trials = rep(1L, N), X = cbind(1, x),
                              spatial_idx = s, family = "binomial")),
  prior = prior, control = list(progress = FALSE))
fits <- lapply(1:2, function(b) fit_one(rbinom(N, 1, plogis(0.3 * x))))
res <- tryCatch(tulpa_joint_grid_batch(fits),
                tulpa_grid_batch_ineligible = function(e) lapply(fits, function(f) f()))
length(res)
```

`tulpa_joint_inner_vcov_blocks`

Per-cell constrained inverse-precision sub-block for a joint fit

---

Description

Runs the same per-outer-grid-cell selected-inversion the joint nested-Laplace driver's own fixed-effect retention uses internally (`extract_inner_vcov_block_cell()`, see the note above `.joint_attach_grid_fixed()`), for a caller-chosen index set. A consumer package computing its own standard-error correction on a subset of a joint fit's latent coordinates – a different `idx` than whatever the fit itself retained on `$cov_block_per_grid` – reruns the extraction through this door instead of reimplementing the sparse selected inversion.

Usage

```
tulpa_joint_inner_vcov_blocks(
  Q_p_per_grid,
  Q_i_per_grid,
  Q_x_per_grid,
  n_x,
  idx,
  n_dense,
  A_cols_list,
  field_marginal = TRUE,
  n_threads = 1L
)
```

Arguments

`Q_p_per_grid`, `Q_i_per_grid`, `Q_x_per_grid`
: Per-outer-grid-cell CSC storage of the latent precision `Q` (`fit$Q_csc_p_per_grid` / `_i_per_grid` / `_x_per_grid`), one list element per cell.

`n_x`
: Integer, the latent dimension `Q` is built over (`fit$Q_csc_n`).

`idx`
: Integer vector (1-based) of latent coordinates to extract the sub-block for.

`n_dense`
: Integer, the number of leading `idx` entries that are dense fixed effects rather than field coordinates (see Details).

`A_cols_list`
: A list of integer vectors, one per sum-to-zero constraint group, each the (1-based) latent coordinates that group averages to zero.

`field_marginal`
: Logical; if `TRUE` (default) and `n_dense < length(idx)`, only the field coordinates' marginal variances are computed (a single Takahashi pass) rather than their full dense sub-block.

`n_threads`
: Integer, grid cells processed concurrently.

Details

When a field is present (`n_dense < length(idx)`), the extraction takes a cheap selected-inversion path: the dense (fixed-effect) block and the fixed-effect x field cross terms are exact; the field x field block is either its marginal diagonal (`field_marginal = TRUE`) or left at zero (`FALSE`) – never the full field x field covariance.

Value

A list of length `n_grid` (one per outer-grid cell), each element either `NULL` (the cell stored no `Q`, or its sparse Cholesky failed) or a dense `length(idx) x length(idx)` matrix: the constrained covariance sub-block for `idx`, with the sum-to-zero constraints in `A_cols_list` conditioned out by the same kriging correction the fit's own posterior draws use.

Examples

```
# Two outer-grid cells of a latent precision over two fixed effects and a
# ten-node random-walk field (sum-to-zero), each stored as the lower-triangle
# CSC a joint fit keeps per cell.
make_Q <- function(tau) {
  Q <- matrix(0, 12, 12)
  Q[1:2, 1:2] <- diag(2, 2) + 0.1
  Q[3:12, 3:12] <- tau * crossprod(diff(diag(10))) + diag(0.3, 10)
  Q[1, 3:12] <- Q[3:12, 1] <- 0.1
  L <- Matrix::tril(methods::as(methods::as(Q, "generalMatrix"),
                                "CsparseMatrix"))
  list(p = L@p, i = L@i, x = L@x)
}
cells <- list(make_Q(1), make_Q(4))
blocks <- tulpa_joint_inner_vcov_blocks(
  lapply(cells, `[[`, "p"), lapply(cells, `[[`, "i"), lapply(cells, `[[`, "x"),
  n_x = 12L, idx = 1:12, n_dense = 2L, A_cols_list = list(3:12))
blocks[[1]][1:2, 1:2]       # fixed-effect covariance in the first cell
diag(blocks[[2]])[3:12]     # field marginal variances in the second
```

---

tulpa_joint_inner_vcov_mixture

Outer-grid posterior covariance of a joint fit's latent coordinates

---

Description

The law-of-total-covariance mixture over the outer grid,

$$V = \sum_k w_k [C_k + (m_k - \bar{m})(m_k - \bar{m})^T],$$

with `C_k` the same per-cell constrained block `tulpa_joint_inner_vcov_blocks()` returns, accumulated inside the engine without holding one block per cell. Memory is one `length(idx)` square result whatever the grid size, so this is the door for an `idx` that carries a field.

Usage

```
tulpa_joint_inner_vcov_mixture(
  Q_p_per_grid,
  Q_i_per_grid,
  Q_x_per_grid,
  n_x,
```

```
  idx,
  n_dense,
  A_cols_list,
  weights,
  modes = NULL,
  field_marginal = TRUE,
  n_threads = 1L
)
```

Arguments

`Q_p_per_grid`, `Q_i_per_grid`, `Q_x_per_grid`
: Per-outer-grid-cell CSC storage of the latent precision `Q` (`fit$Q_csc_p_per_grid` / `_i_per_grid` / `_x_per_grid`), one list element per cell.

`n_x`
: Integer, the latent dimension `Q` is built over (`fit$Q_csc_n`).

`idx`
: Integer vector (1-based) of latent coordinates to extract the sub-block for.

`n_dense`
: Integer, the number of leading `idx` entries that are dense fixed effects rather than field coordinates (see Details).

`A_cols_list`
: A list of integer vectors, one per sum-to-zero constraint group, each the (1-based) latent coordinates that group averages to zero.

`weights`
: Numeric, one non-negative weight per outer-grid cell (normalized here). A cell of zero weight is never extracted.

`modes`
: Optional `n_grid x length(idx)` matrix of the cell modes on `idx`. `NULL` returns the within-cell term alone.

`field_marginal`
: Logical; if `TRUE` (default) and `n_dense < length(idx)`, only the field coordinates' marginal variances are computed (a single Takahashi pass) rather than their full dense sub-block.

`n_threads`
: Integer, grid cells processed concurrently.

Value

A list: `vcov`, the mixture covariance; `mean`, the weighted mean of `modes` (`NULL` without them); `dense_blocks`, one `n_dense x n_dense` block per cell (`NULL` where the cell was not extracted); `failed`, the cells of positive weight whose block is missing (no stored precision, or its Cholesky failed). Such a cell keeps its weight and contributes its between-cell term only. With `field_marginal = TRUE` the field x field part of the within term is its diagonal.

Examples

```
# Two outer-grid cells of a latent precision over two fixed effects and a
# ten-node random-walk field (sum-to-zero), each stored as the lower-triangle
# CSC a joint fit keeps per cell, with the cells' modes and weights.
make_Q <- function(tau) {
  Q <- matrix(0, 12, 12)
  Q[1:2, 1:2] <- diag(2, 2) + 0.1
  Q[3:12, 3:12] <- tau * crossprod(diff(diag(10))) + diag(0.3, 10)
  Q[1, 3:12] <- Q[3:12, 1] <- 0.1
  L <- Matrix::tril(methods::as(methods::as(Q, "generalMatrix"),
```

```
                                "CsparseMatrix"))
  list(p = L@p, i = L@i, x = L@x)
}
cells <- list(make_Q(1), make_Q(4))
set.seed(1)
modes <- matrix(rnorm(2 * 12, sd = 0.2), 2, 12)
mix <- tulpa_joint_inner_vcov_mixture(
  lapply(cells, `[[`, "p"), lapply(cells, `[[`, "i"), lapply(cells, `[[`, "x"),
  n_x = 12L, idx = 1:12, n_dense = 2L, A_cols_list = list(3:12),
  weights = c(0.7, 0.3), modes = modes)
mix$mean[1:2]
mix$vcov[1:2, 1:2]
```

`tulpa_kfold` *K-fold cross-validation for a tulpa fit*

Description

Splits the data into `K` folds, refits the model on each `K - 1` training partition (via the fit's stored `tulpa()` call), and accumulates the held-out fold's pointwise log predictive density $\log \frac{1}{S} \sum_s p(y_i \mid \eta_i^{(s)})$ over the training posterior draws. The summed `elpd_kfold` is directly comparable to the `elpd_loo` from `tulpa_criteria()` – the exact refit counterpart to the PSIS-LOO approximation, for when the Pareto k-hat gate flags LOO as unreliable.

Fixed-effect / GLMM fits only: subsetting the observations breaks a spatial or temporal field, so those fits are rejected (use PSIS-LOO via `tulpa_criteria()`). Held-out random-effect groups contribute at their prior mean (population-level held-out prediction), matching `stats::predict()`.

Usage

```
tulpa_kfold(object, data, K = 10L, folds = NULL, n_trials = NULL, seed = NULL)
```

Arguments

`object`	A `tulpa_fit` fitted through `tulpa()` (must carry `$call`).
`data`	The data frame the model was fit to.
`K`	Number of folds (default 10).
`folds`	Optional integer vector of fold ids, length `nrow(data)`; a random balanced partition is drawn when `NULL`.
`n_trials`	Optional binomial denominators (length `nrow(data)`); defaults to the trials stored on the fit, else 1 (Bernoulli). Each fold's refit receives the training rows' trials, and the held-out density is scored at the test rows' trials.
`seed`	Optional seed for the random partition.

Value

A list with `elpd_kfold` (summed held-out elpd), `se_elpd_kfold` (its standard error), `pointwise` (per-observation held-out elpd), `folds`, and `K`.

See Also

tulpa_criteria() for PSIS-LOO / WAIC on a single fit.

Examples

```
set.seed(1)
d <- data.frame(x = rnorm(120))
d$y <- rpois(120, exp(0.4 + 0.6 * d$x))
fit <- tulpa(y ~ x, data = d, family = "poisson", mode = "laplace")
cv  <- tulpa_kfold(fit, data = d, K = 5, seed = 1)
cv$elpd_kfold
```

---

tulpa_laplace *Fit a model via Laplace approximation*

Description

General-purpose Laplace approximation for latent Gaussian models. Finds the mode of the latent field (beta + random effects) and returns the Laplace-approximated marginal likelihood.

This is the public API for model packages (tulpaGlmm, tulpaObs, etc.) to call tulpa's Laplace engine. As the low-level engine entry point (not a front-door fitter), it keeps its numerical controls (max_iter, tol, n_threads, return_hessian) inline in the signature rather than in a control list, so callers assembling many solves pass them positionally.

Usage

```
tulpa_laplace(
  y,
  n_trials,
  X,
  re_list = list(),
  family = "binomial",
  phi = 1,
  phi2 = NULL,
  spatial = NULL,
  weights = NULL,
  offset = NULL,
  max_iter = 100L,
  tol = 1e-06,
  n_threads = 1L,
  return_hessian = TRUE,
  beta_prior = NULL,
  return_re_cov = FALSE,
  X_zi = NULL,
  zi_prior_sd = 2.5,
  return_joint_hessian = FALSE,
```

```
  compute_skew = FALSE,
  skew_idx = NULL,
  debias = NULL
)
```

Arguments

y	Response vector (integer for binomial/poisson/negbin, numeric for gaussian)
n_trials	Trial sizes (integer vector, used for binomial only)
X	Fixed-effects design matrix
re_list	List of RE specifications. Each element is a list with: • idx: integer vector of group indices (1-based) • n_groups: number of groups • n_coefs: coefficients per group (1 = intercept-only, >1 = random slopes) • sigma: per-coefficient RE standard deviation(s), a diagonal covariance (uncorrelated, lme4 (x || g)). A scalar is recycled to n_coefs. • Z: slope design matrix (n_obs x n_coefs) when n_coefs > 1; NULL means intercept-only. • L / cov: optional n_coefs x n_coefs covariance for a *correlated* term (lme4 (1 + x | g)) – supply either a lower-triangular Cholesky factor L (covariance = L L') or the covariance matrix cov. When present these take precedence over sigma; the off-diagonal enters both the joint Hessian (mode finding) and the marginal fixed-effect SE.
family	Character: "binomial", "poisson", "neg_binomial_2", "gaussian"
phi	Dispersion passed to the family, held fixed. One convention at every door: for gaussian / lognormal this is the residual VARIANCE (the SD is sqrt(phi)), for neg_binomial_2 the size, gamma the shape, beta the precision, t the scale; binomial and poisson ignore it. The compiled kernels parameterize the two variance families by the residual SD and are handed sqrt(phi) at the boundary. Defaulted, it conditions at 1 and says so with a warning, since for a family that reads a dispersion that is a modelling choice rather than a neutral value. fit$phi_estimated records whether the value on the fit was estimated or conditioned on.
phi2	Optional second dispersion: the Student-t degrees of freedom (family = "t"; default 4 when NULL). Non-spatial path only.
spatial	Optional spatial specification (tulpa_spatial object)
weights	Optional observation weights (numeric vector, length length(y)). Scales each observation's log-density, score and Fisher curvature by the same w_i, on the spatial route as well as the non-spatial one, so the mode and the marginal precision H_beta describe one weighted model. NULL (default) uses 1.
offset	Optional observation-level offset on the linear predictor (numeric vector, length length(y)). NULL (default) uses 0.
max_iter	Maximum Newton iterations (default 100)
tol	Convergence tolerance (default 1e-6)

`n_threads` Number of threads (default 1)

`return_hessian` Logical: return the fixed-effect Hessian block? (default TRUE)

`beta_prior` Optional Gaussian prior on the fixed effects. `NULL` (default) keeps the weak built-in prior `beta ~ N(0, 100^2)`. Otherwise a list with element `sd` (prior standard deviation, required) and optional `mean` (prior mean, default 0). Each may be a scalar (applied to every coefficient) or a length-`ncol(X)` vector. Adds `sum((beta - mean)^2 / (2 * sd^2))` to the negative log-posterior, so the mode is the penalized (MAP) estimate. A coefficient's `sd` may be `+Inf`, which sets its precision to 0 (no penalty on that coefficient). Not supported on the spatial path.

`return_re_cov` If `TRUE`, additionally return per-group marginal posterior covariance blocks `Cov(u_g | y, Sigma)` – one `n_coefs x n_coefs` matrix per (RE term, group), with the fixed effects and other groups marginalized out (each block is a diagonal block of the full inverse Hessian, not the inverse of a diagonal block). Used by the EM M-step for a full random-effect covariance. Non-spatial multi-RE path only.

`X_zi` Optional zero-inflation design matrix (`length(y)` rows). When supplied the latent fixed-effect block becomes `[beta_count | beta_zi]` and the family's compiled zero-inflated kernel is used. Non-spatial path only.

`zi_prior_sd` Prior SD on the zero-inflation coefficients, `beta_zi ~ N(0, zi_prior_sd^2)` (default 2.5, matching the samplers' `ModelData::zi_prior_sd`). It is what keeps the logit identified when a level contributes no zeros, where the likelihood alone drives `beta_zi` to `-Inf`. `+Inf` removes the penalty. Ignored when `X_zi` is `NULL`.

`return_joint_hessian`
If `TRUE`, additionally return `H_joint`: the full joint posterior precision of the latent field `[beta | random effects]` at the mode, as a symmetric sparse matrix. This is the matrix the Laplace approximation takes the determinant of, so it is what an exact derivative of the log-marginal has to differentiate through; `H_beta` is only its fixed-effect Schur complement. Costs one extra copy of the Hessian, so it is off by default. Non-spatial multi-RE path only. With `return_hessian = TRUE` it also returns `H_latent`, the same precision at the observed curvature (identical to `H_joint` except for the families whose Newton weight is the expected one); `H_beta` is its Schur complement, and `posterior_predict()` draws the random effects from it.

`compute_skew` If `TRUE`, additionally return the inner-Laplace reliability material at `skew_idx`: `inner_skew` (gamma_3, Rue Martino & Chopin 2009's cubic term), and the importance curve `inner_is_z` / `inner_is_log_joint` the inner Pareto-k-hat is fitted from. Costs one linear solve plus a fixed batch of objective evaluations per probed index. Non-spatial path only.

`skew_idx` 1-based latent indices to probe (in the `[beta | random effects]` layout of `mode`). `NULL` with `compute_skew = TRUE` probes every latent index.

`debias` Subspace debias: a list with `idx` (1-based latent indices to correct by Metropolis along the Gaussian-conditional-mean surface through the mode) and optional `n_iter` / `warmup` / `thin`. The result then carries `debias_draws` (`n_kept x length(idx)`, the sampled `x_S - mode_S`), `debias_sigma_ss` (the inner Laplace's own marginal covariance of `x_S`), `debias_accept` and `debias_idx`. `NULL` (default) or an empty `idx` leaves the solve bit-for-bit as it was and consumes no random number. Non-spatial path only.

Value

A list with:

- `mode`: full mode vector (beta, then RE values per term)
- `log_marginal`: Laplace-approximated log-marginal likelihood
- `n_iter`: number of Newton iterations
- `converged`: logical, whether the stopping rule was met
- `score_max`: largest absolute component of the joint penalized score at the returned mode – the residual the solve actually achieved, which is a different question from `converged` and the one anything differentiating through the mode depends on
- `log_det_Q`: log-determinant of the Hessian
- `H_beta`: fixed-effect block of the Hessian (if return_hessian = TRUE)
- `H_joint`, `H_latent`: the joint latent precision, at the kernel's and at the observed curvature (if return_joint_hessian = TRUE)
- `cov_blocks`: list of per-group posterior covariance matrices, one per (RE term, group) in term-major then group order (if return_re_cov = TRUE)

Examples

```
set.seed(1)
n <- 200L
X <- cbind(1, rnorm(n))
eta <- X %*% c(-0.3, 0.8)
y <- rbinom(n, 1, plogis(eta))
fit <- tulpa_laplace(y, rep(1L, n), X, family = "binomial")
fit$mode          # posterior mode of the fixed effects
```

`tulpa_laplace_beta` *Fit a beta-regression model via Laplace, estimating the precision*

Description

Thin wrapper around `tulpa_laplace()` for `family = "beta"`. The Laplace engine treats `phi` as fixed per fit (same contract as gamma and neg_binomial_2); this wrapper does an outer 1-D optimisation of the Laplace-approximated log-marginal over `phi`, then refits at the optimum to return betas and Hessian.

The mean-precision parameterisation is `y ~ Beta(mu * phi, (1 - mu) * phi)` with default logit link; `y` must be strictly in `(0, 1)`.

Usage

```
tulpa_laplace_beta(
  y,
  X,
  re_list = list(),
  spatial = NULL,
  weights = NULL,
  offset = NULL,
  max_iter = 100L,
  tol = 1e-06,
  n_threads = 1L,
  beta_prior = NULL,
  phi_init = NULL,
  phi_bounds = c(0.1, 10000),
  outer_tol = 1e-04,
  mode = c("laplace", "nuts"),
  control = list()
)
```

Arguments

`y` Response in `(0, 1)`.

`X` Fixed-effects design matrix.

`re_list`, `spatial`, `weights`, `offset`, `max_iter`, `tol`, `n_threads`, `beta_prior`
Passed to `tulpa_laplace()` verbatim. `beta_prior` places a Gaussian penalty on the fixed effects (a list with `sd`, optional `mean`; see `tulpa_laplace()`). It is included in the Laplace log-marginal that the precision `phi` is optimised against, so the penalised model is fit consistently across the outer `phi` search. Not supported with `spatial` (the spatial solver carries its own prior).

`phi_init` Optional starting value for the precision. If `NULL`, a method-of-moments warm start is used.

`phi_bounds` Numeric length-2 vector with lower/upper bounds on `phi` for the outer optimisation. Default `c(0.1, 1e4)`.

`outer_tol` Tolerance for the outer optimisation. Default 1e-4.

`mode` Inference method (the method is an argument, not a parallel verb): `"laplace"` (default) is the Laplace + Brent-over-`phi` point fit documented here; `"nuts"` delegates to `tulpa_nuts_beta()`, which samples `phi` jointly with the coefficients via NUTS. In `"nuts"` mode the Laplace-only arguments (`re_list`, `spatial`, `weights`, `offset`, `phi_init`, `phi_bounds`, `outer_tol`) are not used, and NUTS knobs are passed via `control` (see `tulpa_nuts_beta()`).

`control` Passed to `tulpa_nuts_beta()` when `mode = "nuts"` (ignored for `mode = "laplace"`).

Value

For `mode = "laplace"`, the list returned by `tulpa_laplace()` at the optimum, augmented with `phi` (the optimised precision), `phi_trace` (the optimisation trace: a matrix with one row per objective

evaluation and columns phi and log_marginal) and phi_converged (TRUE when the Brent search reported convergence). For mode = "nuts", the draws object returned by tulpa_nuts_beta().

Examples

```
set.seed(1)
n <- 200L
X <- cbind(1, rnorm(n))
mu <- plogis(X %*% c(0.2, 0.7)); phi <- 8
y <- rbeta(n, mu * phi, (1 - mu) * phi)
fit <- tulpa_laplace_beta(y, X)
fit$mode
```

---

tulpa_latent *Latent factor specification for unmeasured confounders*

Description

Specify latent factors to capture shared unmeasured confounders between model processes. Latent factors are particularly useful when you suspect that both processes are driven by common unmeasured variables.

Value

The constructor documented in this family (latent_factor()) returns a tulpa_latent specification object consumed by ratio / multi-arm model packages built on tulpa (e.g. tulpaRatio), where the shared factor enters two or more linear predictors and cancels in the derived ratio. It is a multi-arm construct: the single-response tulpa() front door does not read it (for an in-formula latent Gaussian block on a single response, use latent(tgmrf(...))).

---

tulpa_loglik *Streaming pointwise log-likelihood*

Description

Wrap a pointwise log-likelihood for tulpa_criteria() without materializing the whole [n_draws x n_obs] matrix. A plain matrix is wrapped directly; a block generator (a function of an integer column vector returning the [n_draws x length(cols)] submatrix) lets the criteria accumulators stream over observation blocks, so an EVA-scale [200 x 1.16M] log-likelihood is consumed a few thousand columns at a time.

Usage

```
tulpa_loglik(x, n_obs = NULL, n_draws = NULL)
```

Arguments

x	Either a numeric [n_draws x n_obs] matrix, an existing tulpa_loglik, or a function f(cols) returning the [n_draws x length(cols)] submatrix for the integer column indices cols.
n_obs, n_draws	Required when x is a generator function; the column and row counts of the implied matrix.

Value

A tulpa_loglik object: a list with get(cols), n_obs, n_draws, and materialized.

See Also

tulpa_criteria()

Examples

```
ll <- tulpa_loglik(matrix(rnorm(20 * 5, -1), nrow = 20))  # 20 draws x 5 obs
c(ll$n_draws, ll$n_obs)
```

---

tulpa_multinomial *Multinomial (nominal K-class) logistic regression via Laplace*

Description

Fits a baseline-category multinomial logit model: for a K-level unordered response, classes 1..K-1 each get their own linear predictor and class K is the baseline. The coupled multinomial likelihood is solved by a Newton step to the penalized mode (a Gaussian ridge prior on the coefficients) and summarized by a Laplace approximation, reusing the native multinomial kernel.

Usage

```
tulpa_multinomial(
  formula,
  data,
  beta_prior = .tulpa_default_beta_prior("multinomial"),
  control = list()
)

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

Arguments

formula	Model formula; the response must be a factor (or coercible to one) with >= 3 levels. The baseline is the last level.
data	A data frame.
beta_prior	Fixed-effect prior as `list(mean, sd)`: a mean-zero (`mean = 0`) Gaussian ridge on every coefficient with scalar SD `sd` (default the engine default, `prior_normal(0, 2.5)`). A finite SD keeps the mode finite under separation.
control	List of numerical knobs: `max_iter` (default 100), `tol` (default 1e-8), `n_draws` (posterior draws, default 2000), `seed`.
x	A `tulpa_multinomial` fit.
...	Ignored.

Value

A `tulpa_fit` (subclass `tulpa_multinomial`) with `means` (the posterior mode, named `class:term`) and `cov` of the Laplace Gaussian, which is the posterior `coef()`, `vcov()`, `summary.tulpa_fit()` and `confint()` report; `draws` sampled from that Gaussian; `log_marginal`, `classes`, `baseline`, and the standard generic-method support.

See Also

`tulpa()` for single-process GLMMs.

Examples

```
set.seed(1)
n <- 300L; x <- rnorm(n)
eta <- cbind(0.5 + 1.0 * x, -0.3 - 0.8 * x)          # classes 1, 2 vs baseline 3
P <- cbind(exp(eta), 1); P <- P / rowSums(P)
y <- factor(apply(P, 1, function(pr) sample.int(3L, 1L, prob = pr)))
fit <- tulpa_multinomial(y ~ x, data = data.frame(y = y, x = x))
coef(fit)
```

---

tulpa_normalise_weights_safe

Normalise outer-grid log-marginals to posterior cell weights

Description

The single weight normaliser for every outer hyperparameter grid: softmax of `lm` (optionally plus a per-cell log quadrature weight) into weights summing to 1. Non-finite nodes (an inner Newton diverging in a grid corner) are dropped from the max-shift and zeroed before renormalising over the finite cells; an all-non-finite grid returns all-`NA` with a warning rather than `NaN`. Intended for reconstructing a fit's outer-grid posterior weights (with `tulpa_theta_matrix()` and `tulpa_grid_log_quad()`) when the fit doesn't already carry `fit$weights`.

Usage

```
tulpa_normalise_weights_safe(lm, what = "grids / data", log_quad = NULL)
```

Arguments

lm	Numeric vector of per-cell log-marginal-likelihood values.
what	Label for the grid, used only in the degenerate-case warning.
log_quad	Optional per-cell log quadrature weight (prior mass) of the outer grid, e.g. from tulpa_grid_log_quad(); NULL normalises the likelihood alone.

Value

Numeric vector of posterior cell weights summing to 1, the same length as lm, or all-NA if no cell carries finite mass.

See Also

tulpa_theta_matrix(), tulpa_grid_log_quad()

Examples

```
tulpa_normalise_weights_safe(c(-1, 0, -2))
# With per-cell log quadrature weights (cell prior masses):
tulpa_normalise_weights_safe(c(-1, 0, -2), log_quad = log(c(0.2, 0.5, 0.3)))
```

---

tulpa_nuts_beta — *Fit a beta-regression model via NUTS (joint sampling of beta + log_phi)*

Description

Bayesian counterpart to tulpa_laplace_beta(). Routes the Beta GLM through tulpa's full NUTS backend via the generic LikelihoodSpec interface, sampling the precision phi jointly with the regression coefficients on the log scale. NUTS performs exact marginalisation over phi, replacing the Brent outer-opt in tulpa_laplace_beta() and the deterministic nested-Laplace grid that would otherwise integrate over the Beta dispersion.

Mean-precision parameterisation, default logit link: y_i ~ Beta(mu_i * phi, (1 - mu_i) * phi) with mu_i = 1 / (1 + exp(-eta_i)) and eta_i = X_i %*% beta.

Usage

```
tulpa_nuts_beta(
  y,
  X,
  beta_prior = .tulpa_default_beta_prior("beta_nuts"),
  log_phi_prior_sd = 3,
  log_phi_init = 0,
```

```
  control = list()
)
```

Arguments

`y`	Response vector, strictly in `(0, 1)`.
`X`	Fixed-effects design matrix.
`beta_prior`	Fixed-effect prior as `list(mean, sd)`: a mean-zero (`mean = 0`) Gaussian on each coefficient with SD `sd` (default the engine default, `prior_normal(0, 2.5)`).
`log_phi_prior_sd`	Prior SD on `log(phi)` (`log_phi ~ N(0, log_phi_prior_sd)`). Default `3` (very weak; covers `phi` from ~0.001 to ~1000 within +-2 SD).
`log_phi_init`	Starting value for `log(phi)`. Default `0` (i.e. `phi = 1`); a method-of-moments warm start can speed warmup in highly concentrated regimes.
`control`	A named list of numerical / sampler knobs (statistical arguments stay in the signature): `n_iter` (default 2000), `n_warmup` (default 1000), `max_treedepth` (default 10), `adapt_delta` (default 0.8), `seed` (`NULL` draws from the session RNG), `verbose` (default FALSE).

Value

A `tulpa_fit` object with:

- `draws` – `n_samples x (p + 1)` matrix of post-warmup draws, columns named from `colnames(X)` (falling back to `beta[1] ... beta[p]`) then `log_phi`.
- `means` – posterior means.
- `phi_summary` – posterior mean / median / quantiles of `phi = exp(log_phi)`.
- `accept_prob`, `divergent`, `treedepth`, `epsilon` – NUTS diagnostics from the underlying chain.

See Also

`tulpa_laplace_beta()` for the Laplace + Brent point estimate; `tulpa_laplace()` for the underlying Laplace engine.

Examples

```
set.seed(1)
n <- 150L
X <- cbind(1, rnorm(n))
mu <- plogis(X %*% c(0.2, 0.7)); phi <- 8
y <- rbeta(n, mu * phi, (1 - mu) * phi)

fit <- tulpa_nuts_beta(y, X, control = list(n_iter = 500L, n_warmup = 250L))
colMeans(fit$draws)
```

tulpa_nuts_spde *Sample an SPDE GLM via NUTS, optionally jointly over Matern hypers*

Description

Bayesian counterpart to `fit_spde()` / `laplace_spde_at()`. Routes the SPDE-augmented GLM through tulpa's full NUTS backend via the generic `LikelihoodSpec` interface. Two modes:

- **Legacy / fixed-hyper** (`joint = FALSE`, the default): conditions on the supplied `(range, sigma)`. Samples the latent block `(beta, w_mesh, log_phi)` jointly. Useful as the inner step of an outer CCD / nested-Laplace integration over hypers.
- **Joint hypers** (`joint = TRUE`): samples `(beta, z_mesh, log_kappa, log_tau, log_phi)` jointly via the non-centered transform `w = L(theta)^{-T} z`. A PC prior (Fuglstad et al. 2019 JASA) on `(range, sigma)` is taken from `prior_range`, `prior_sigma`. Returns additional `w_draws`, `range_draws`, `sigma_draws`, `kappa_draws`, `tau_draws` columns/vectors on top of the raw parameter draws.

Usage

```
tulpa_nuts_spde(
  y,
  X,
  spatial,
 family = c("gaussian", "poisson", "binomial", "gamma", "neg_binomial_2", "beta"),
  n_trials = NULL,
  joint = FALSE,
  range = NULL,
  sigma = NULL,
  prior_range = NULL,
  prior_sigma = NULL,
  log_kappa_init = NULL,
  log_tau_init = NULL,
  beta_prior = .tulpa_default_beta_prior("spde_nuts"),
  log_phi_prior_sd = 3,
  log_phi_init = 0,
  control = list()
)
```

Arguments

`y` Response vector. Family-specific:

- `gaussian`: any real
- `poisson`, `neg_binomial_2`: non-negative integers
- `binomial`: non-negative integers in `[0, n_trials]`
- `gamma`: strictly positive reals

- beta: strictly in (0, 1)

X — Fixed-effects design matrix.

spatial — A tulpa_spatial object from spatial_spde() / spatial_spde_custom() – supplies the FEM matrices (C0, G1) and projection (A) plus the smoothness nu.

family — One of "gaussian", "poisson", "binomial", "gamma", "neg_binomial_2", "beta".

n_trials — Integer vector for family = "binomial" (else ignored).

joint — Logical. FALSE (default) conditions on fixed (range, sigma). TRUE activates joint sampling of (log_kappa, log_tau) with the PC prior from prior_range, prior_sigma.

range, sigma — Fixed-hyper mode only. Matern range and marginal SD on the field. Default to the SPDE prior medians (spatial$prior_range[1], spatial$prior_sigma[1]).

prior_range, prior_sigma — Joint mode only. PC prior anchors as c(value, alpha) pairs:

- prior_range = c(r0, a_r) encodes P(range < r0) = a_r
- prior_sigma = c(s0, a_s) encodes P(sigma > s0) = a_s Default to the spec's anchors (spatial$prior_range, spatial$prior_sigma) when those are length-2 PC anchors.

log_kappa_init, log_tau_init — Joint mode only. Initial values for the hyper slots. Default to the value implied by the PC anchor's (r0, s0) pair via kappa = sqrt(8 nu) / r0, tau = 1 / (sqrt(4 pi) * kappa * s0).

beta_prior — Fixed-effect prior as list(mean, sd): a mean-zero (mean = 0) Gaussian on each coefficient with SD sd (default the engine default, prior_normal(0, 2.5)).

log_phi_prior_sd — Prior SD on log(phi). Role of phi is family-specific:

- gaussian: phi is the residual SD (sampled jointly)
- gamma: phi is the Gamma shape (sampled jointly)
- neg_binomial_2: phi is the NB size r (sampled jointly)
- beta: phi is the Beta precision (sampled jointly)
- poisson, binomial: log_phi is held tight and ignored downstream.

log_phi_init — Starting value for log(phi).

control — A named list of numerical / sampler knobs (statistical arguments stay in the signature): n_iter (default 2000), n_warmup (default 1000), max_treedepth (default 10), adapt_delta (default 0.8), seed (NULL draws from the session RNG), verbose (default FALSE), noncenter (fixed-hyper only, default TRUE: sample the mesh field in a non-centered v = L^{-T} z parameterisation – the same target density, a priori isotropised; ignored when joint = TRUE), and mass_matrix (NUTS metric: "auto" (default) picks a dense metric at <= 200 parameters and diagonal above, capturing the fixed-effect / field cross-curvature that corrects the intercept marginal; "diag", "dense", "block_diag" force the choice).

Value

A list with `draws` (matrix `n_samples x n_params`), `means`, `phi_summary` (where applicable), `accept_prob`, `divergent`, `treedepth`, `epsilon`, `joint_hypers`, plus the supplied `spatial` spec. In `joint = TRUE` mode additionally: `w_draws` (transformed `z -> w` per draw), `range_draws`, `sigma_draws`, `kappa_draws`, `tau_draws`, and `range_summary`, `sigma_summary` (mean/median/5%/95% quantiles).

See Also

`fit_spde()` for the Laplace counterpart and the nested-Laplace path over (range, sigma).

Examples

```
# Gaussian response with a smooth spatial trend, sampled at fixed
# (range, sigma) on a coarse mesh.
set.seed(1)
n <- 60L
coords <- cbind(runif(n), runif(n))
x <- runif(n, -1, 1)
y <- 0.5 - 0.8 * x + 0.4 * sin(3 * coords[, 1]) + rnorm(n, 0, 0.3)
spec <- spatial_spde(coords, max_edge = c(0.3, 0.6), cutoff = 0.1,
                     prior_range = c(0.5, 0.5), prior_sigma = c(1, 0.5))
fit <- tulpa_nuts_spde(y, cbind(1, x), spatial = spec, family = "gaussian",
                       range = 0.4, sigma = 0.5, log_phi_init = log(0.3),
                       control = list(n_iter = 300L, n_warmup = 150L,
                                      seed = 1L))
colMeans(fit$draws[, c("beta[1]", "beta[2]")])
fit$phi_summary
```

---

tulpa_ordinal *Ordinal (ordered K-class) cumulative-logit regression via Laplace*

Description

Fits a proportional-odds cumulative-logit model for an ordered factor response: `P(y <= j | x) = plogis(c_j - x'beta)` with ordered cutpoints `c_1 < ... < c_{K-1}`. The penalized mode (a ridge prior on the coefficients and cutpoints) is found by L-BFGS and summarized by a Laplace approximation. There is no separate intercept – the cutpoints carry the baseline levels.

Usage

```
tulpa_ordinal(
  formula,
  data,
  link = c("logit", "probit"),
  beta_prior = .tulpa_default_beta_prior("ordinal"),
  cut_prior_sd = 10,
```

```
  control = list()
)

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

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

Arguments

formula	Model formula; the response must be an ordered (or coercible) factor with >= 3 levels, each observed at least once. An intercept in formula is dropped; y ~ 1 fits the cutpoints alone. Incomplete rows are refused, not dropped.
data	A data frame.
link	Cumulative link: "logit" (proportional odds, default) or "probit".
beta_prior	Fixed-effect prior as list(mean, sd): a mean-zero (mean = 0) Gaussian ridge on every coefficient with scalar SD sd (default the engine default, prior_normal(0, 2.5)).
cut_prior_sd	SD of the mean-zero Gaussian ridge prior on the cutpoint parameters (default 10).
control	List of numerical knobs: max_iter (default 200), n_draws (default 2000), seed.
object	A tulpa_ordinal fit.
...	Ignored.
x	A tulpa_ordinal fit.

Value

A tulpa_fit (subclass tulpa_ordinal) with coefficients (covariate effects) and cutpoints at the posterior mode, draws (the Laplace Gaussian mapped to ordered cutpoints), log_marginal, levels. The draws are the posterior coef(), vcov(), summary.tulpa_fit() and confint() report; coef() omits the cutpoints, as MASS::polr does.

coef() returns the covariate effects without the cutpoints, as MASS::polr() omits its zeta; vcov() and summary() keep them.

See Also

tulpa_multinomial() for the nominal (unordered) case.

Examples

```
set.seed(1)
n <- 400L; x <- rnorm(n)
cuts <- c(-1, 0.5, 2); eta <- 0.8 * x
Fm <- plogis(outer(-eta, cuts, "+")); P <- cbind(Fm, 1) - cbind(0, Fm)
y <- ordered(apply(P, 1, function(pr) sample.int(4L, 1L, prob = pr)))
fit <- tulpa_ordinal(y ~ x, data = data.frame(y = y, x = x))
```

```
fit$coefficients; fit$cutpoints
```

`tulpa_outer_grid` *The outer hyperparameter grid of a nested-Laplace fit*

Description

A nested-Laplace fit integrates its hyperparameters on an outer grid: each cell is one hyperparameter value, carries an integration weight, and holds the inner Laplace solve at that value. This returns that grid, cell by cell, so the integration can be inspected or reused (a discrete hyperparameter posterior for `sbc_discrete()`, a grid-marginalized latent field).

The per-cell latent modes average to the grid-marginalized posterior mean of any latent coordinate: `colSums(g$weights * g$modes)`. Columns `seq_len(g$n_fixed)` are the fixed effects and the rest the latent field and random effects, in the order the fit's latent vector stores them.

Usage

```
tulpa_outer_grid(fit)
```

Arguments

`fit`	A nested-Laplace fit: `tulpa_nested_laplace()`, `tulpa_nested_laplace_joint()`, or a `tulpa()` fit routed to that tier.

Value

A list:

`nodes` numeric matrix `[n_cells x n_axes]`, the hyperparameter value of each cell, with the axis names as column names.

`weights` the cells' integration weights, normalized to sum to one.

`log_marginal` each cell's log marginal likelihood, hyperprior density included; with the cell's quadrature volume it gives the weight. `NULL` when the fit does not keep it.

`support` named list of natural-scale `c(lo, hi)` intervals, the span each axis's outer measure integrates (including levels a refinement pass added); `NULL` when the fit records none.

`modes` numeric matrix `[n_cells x n_latent]`, each cell's inner Laplace mode, or `NULL` when the fit keeps none.

`n_fixed` how many leading columns of `modes` are fixed effects.

See Also

`tulpa_hyper_draws()` for continuous draws of the hyperparameters, `tulpa_posterior_draws()` for draws of the latent field, and `summary(fit)$hyperparameters` for their marginal summaries.

Examples

```
set.seed(1)
S <- 30L
nb <- lapply(seq_len(S), function(s) setdiff(c(s - 1L, s + 1L), c(0L, S + 1L)))
nn <- lengths(nb)
field <- as.numeric(scale(cumsum(rnorm(S, 0, 0.4))))
idx <- rep(seq_len(S), each = 6L); n <- length(idx); x <- rnorm(n)
y <- rbinom(n, 1L, plogis(-0.2 + 0.6 * x + field[idx]))
prior <- list(type = "icar", n_spatial_units = S, spatial_idx = idx,
              adj_row_ptr = c(0L, cumsum(nn)), adj_col_idx = unlist(nb) - 1L,
              n_neighbors = nn, tau_grid = c(0.5, 1, 2, 4, 8))
fit <- tulpa_nested_laplace(y, rep(1L, n), cbind(1, x), prior = prior,
                            family = "binomial",
                            control = list(progress = FALSE))
g <- tulpa_outer_grid(fit)
cbind(g$nodes, weight = g$weights)
field_hat <- colSums(g$weights * g$modes)[-seq_len(g$n_fixed)]
```

---

tulpa_parse_formula *Parse a mixed-model formula*

Description

Decomposes a formula into fixed effects and random effects by walking the formula's abstract syntax tree. This is structural recursion with pattern matching on bar terms – no string manipulation.

Usage

```
tulpa_parse_formula(formula)

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

Arguments

formula	A formula object (e.g., y ~ x + (1 | group))
x	A tulpa_parsed_formula object.
...	Ignored.

Value

A list with:

- response: character, the response variable label (NULL if none)
- response_expr: language object, the LHS of ~ (NULL if none)
- fixed_formula: formula, the fixed-effects-only formula

- random_effects: list of parsed RE specifications
- n_re_terms: integer, number of RE terms
- latent_blocks: list of evaluated user-defined latent block objects (typically tgmrf S3 objects), one per latent(...) term in the original formula. Empty list when no latent(...) terms are present.
- n_latent_blocks: integer, number of latent(...) terms
- original: the original formula

Examples

```
pf <- tulpa_parse_formula(y ~ x1 + x2 + (1 | group) + (x1 || site))
pf$response        # "y"
pf$fixed_formula   # y ~ x1 + x2
```

---

tulpa_pit *Probability integral transform from a predictive CDF*

Description

The generic, family-agnostic half of a PIT residual check: the model package supplies the posterior-predictive CDF evaluated at each observation (a [n_draws x n_obs] matrix, or a draw-averaged [n_obs] vector), and this returns the PIT value per observation. For a discrete or mixed response (a hurdle has a point mass at zero) supply the left limit cdf_lower (P(Y < y)); the randomized PIT then draws one uniform per observation and interpolates F(y^-) + U (F(y) - F(y^-)), which is uniform under a correct model. With cdf_lower = NULL the response is treated as continuous and the PIT is the draw-averaged CDF.

Usage

```
tulpa_pit(
  cdf,
  cdf_lower = NULL,
  jitter = TRUE,
  log_lik = NULL,
  tail_points = NULL,
  n_threads = 1L
)
```

Arguments

cdf	Posterior-predictive CDF at the observed value, P(Y <= y). A [n_draws x n_obs] matrix (averaged over draws here) or an [n_obs] vector.
cdf_lower	Optional left-limit CDF P(Y < y), same shape as cdf, for the randomized PIT of a discrete / mixed response.

`jitter`	If `TRUE` (default) and `cdf_lower` is `NULL`, add a tiny uniform jitter to break ties from a discretized CDF; ignored when `cdf_lower` is supplied (the interpolation already randomizes).
`log_lik`	Optional `[n_draws x n_obs]` matrix of the pointwise log-likelihood at each draw. When supplied, the PIT is the **leave-one-out** PIT: `cdf` / `cdf_lower` are reweighted by that observation's PSIS leave-one-out weights instead of being column-averaged.
`tail_points`	Optional override for the PSIS tail size used by the leave-one-out weighting (see `tulpa_psis()`); `NULL` uses the automatic rule. Ignored unless `log_lik` is supplied.
`n_threads`	Number of threads for the leave-one-out weighting (one observation per thread). Ignored unless `log_lik` is supplied.

Details

Supplying `log_lik` switches to the **leave-one-out** PIT (as in INLA's `cpo$pit` or `loo::psis_loo()`'s LOO-PIT): each observation's CDF limits are averaged over draws with PSIS leave-one-out weights (from that observation's own pointwise log-likelihood) instead of equal weights, so the PIT does not use the observation to predict itself. A column whose importance ratio is not all finite falls back to the equal-weight average for that observation.

Value

Numeric vector of length `n_obs` of PIT values in `[0, 1]`.

See Also

`tulpa_criteria()`, `tulpa_psis()`

Examples

```
set.seed(1)
Fl <- matrix(runif(200 * 10, 0, 0.4), 200, 10)   # per-draw CDF limits
Fu <- Fl + 0.3
tulpa_pit(Fu, cdf_lower = Fl)
```

`tulpa_posterior_draws` *Posterior draws from a nested-Laplace fit*

Description

Draw from the outer-grid mixture posterior of a nested-Laplace fit – the engine analogue of `inla.posterior.sample()`. Each draw picks an outer-grid cell `k ~ Categorical(weights)` and then samples that cell's inner Gaussian, so the draws are i.i.d. samples from `sum_k w_k N(m_k, V_k)`.

Sampling the mixture is the faithful primitive for marginalizing nonlinear derived quantities (e.g. `plogis(eta_2) - plogis(eta_1)`, expected-cover products `p * mu`): compute the derived quantity per draw, then summarize. Collapsing the grid to a single moment-matched Gaussian biases skewed or multimodal-over-grid posteriors.

Usage

```
tulpa_posterior_draws(fit, idx = NULL, n = 1000, ...)

## Default S3 method:
tulpa_posterior_draws(fit, idx = NULL, n = 1000, ...)

## S3 method for class 'tulpa_nested_laplace'
tulpa_posterior_draws(fit, idx = NULL, n = 1000, ...)
```

Arguments

`fit`	A nested-Laplace fit (`tulpa_nested_laplace()` or `tulpa_nested_laplace_joint()`).
`idx`	Optional integer vector of 1-based indices into whatever a draw covers for this backend (see above); `NULL` (default) returns all of them.
`n`	Number of posterior draws (default 1000).
`...`	Unused; for S3 compatibility.

Value

A numeric matrix `[n x length(idx)]`, one row per draw. Carries `attr(., "draws_kind") = "iid"` (consistent with the draws-provenance gate), `attr(., "cells")` – the outer-grid cell index each row was drawn from – and `attr(., "scope")`, which of the two representations above the columns are. `attr(., "theta")` is the HYPERPARAMETER half of the same rows, one column per outer-grid axis, continuized within each row's own cell by `tulpa_hyper_draws()`; reading `fit$theta_grid[attr(., "cells"), ]` instead returns the bare node coordinate, which is an atom rather than a marginal.

What a draw covers

It depends on which representation the backend retained, and the returned matrix says so in its `scope` attribute.

`tulpa_nested_laplace_joint` the FULL latent vector – per-arm fixed effects, per-arm random effects, then the latent field(s) – because the joint fit retains each cell's sparse precision over that vector (`control$store_Q = TRUE`). `scope` is `"latent"`.

`tulpa_nested_laplace` **(single-block)** the FIXED-EFFECT block. This backend inverts each cell's precision into the marginal fixed-effect block and releases the precision itself, so the latent field is not part of the retained per-cell Gaussian and cannot be sampled from the fit. `scope` is `"fixed"`.

See Also

`tulpa_hyper_draws()`, `tulpa_nested_laplace()`, `tulpa_nested_laplace_joint()`, `posterior_sample()`

Examples

```
set.seed(1)
S <- 30L                                  # spatial units in a chain
nb <- lapply(seq_len(S), function(s) setdiff(c(s - 1L, s + 1L), c(0L, S + 1L)))
nn <- lengths(nb)
field <- as.numeric(scale(cumsum(rnorm(S, 0, 0.4))))
idx <- rep(seq_len(S), each = 6L); n <- length(idx); x <- rnorm(n)
y <- rbinom(n, 1L, plogis(-0.2 + 0.6 * x + field[idx]))
prior <- list(type = "icar", n_spatial_units = S, spatial_idx = idx,
              adj_row_ptr = c(0L, cumsum(nn)), adj_col_idx = unlist(nb) - 1L,
              n_neighbors = nn, tau_grid = c(0.5, 1, 2, 4, 8))
fit <- tulpa_nested_laplace(y, rep(1L, n), cbind(1, x), prior = prior,
                            family = "binomial",
                            control = list(progress = FALSE))
dr <- tulpa_posterior_draws(fit, n = 200)
colMeans(dr)                # fixed effects, mixed over the outer grid
summary(attr(dr, "theta"))  # the hyperparameter half of the same draws
```

---

tulpa_posterior_draws.tulpa_nested_laplace_joint

Posterior draws from a joint nested-Laplace fit

Description

The `tulpa_posterior_draws()` method for a joint nested-Laplace fit (`tulpa_nested_laplace_joint()`). Each draw picks an outer-grid cell `k ~ Categorical(weights)` and then samples the inner latent vector from the constrained Gaussian `N(m_k, V_k)` at that cell, where `m_k` is the cell's inner mode and `V_k` is the inner-Laplace covariance with the ICAR / BYM2 sum-to-zero field constraint imposed (conditioning by kriging). The draws are i.i.d. samples from `sum_k weights_k * N(m_k, V_k)`.

Usage

```
## S3 method for class 'tulpa_nested_laplace_joint'
tulpa_posterior_draws(fit, idx = NULL, n = 1000, ...)
```

Arguments

`fit`	A `tulpa_nested_laplace_joint` fit (single-block or multi-block). The fit must have been produced with `control$store_Q = TRUE` so the per-grid sparse precision `Q_csc_*_per_grid` is available.
`idx`	Optional integer vector of 1-based latent indices to return. `NULL` (default) returns the full latent vector. The latent vector stacks per-arm fixed effects, per-arm random effects, then the latent field(s); use `fit$arm_layout` (`beta_start`, `re_start`, `phi_start` / `theta_start` / `field_starts`, all 0-based) to map a sub-block to indices.
`n`	Number of posterior draws (default 1000).
`...`	Unused; for S3 compatibility.

Details

The constrained draw at cell k uses the sparse Cholesky of the stored precision Q_k and the conditioning-by-kriging correction

$$z_c = z - Q_k^{-1} A^\top (A Q_k^{-1} A^\top)^{-1} A z,$$

where z ~ N(0, Q_k^{-1}) and A stacks the field sum-to-zero rows (one for an ICAR / CAR field; two – structured and unstructured – for a BYM2 field; one per spatial block in a multi-block fit). The returned draw is m_k + z_c, restricted to idx. Because m_k already satisfies the constraint (the inner solve centres the field), the mean is left unchanged and only the covariance is constrained, so the per-cell marginal matches the inner-Laplace constrained covariance exactly.

Cells with zero outer-grid weight (e.g. pruned cells) or no stored Q are dropped and the remaining weights renormalized. A degenerate single-cell grid (quadrature ESS 1) returns draws from that cell's N(m_1, V_1).

Value

A numeric matrix [n x length(idx)] of latent draws, one row per draw, columns named x<idx>. Carries attr(., "draws_kind") = "iid" (consistent with the draws-provenance gate), attr(., "cells") – the outer-grid cell index each row was drawn from – attr(., "theta"), the hyperparameter half of the same rows continuized within each row's own cell (tulpa_hyper_draws()) – and attr(., "scope") = "latent".

See Also

tulpa_posterior_draws(), tulpa_nested_laplace_joint(), posterior_sample()

---

tulpa_powerscale_sensitivity

Power-scaling prior / likelihood sensitivity

Description

Local power-scaling sensitivity (Kallioinen et al. 2024): how much each fixed-effect posterior moves when the prior or the likelihood is raised to a power alpha near 1. The existing draws are importance-reweighted by exp((alpha - 1) * log_component) (PSIS-smoothed via tulpa_psis(); no refits), and the sensitivity is the gradient of the cumulative Jensen-Shannon distance between the base and power-scaled posteriors with respect to log2(alpha).

Values above threshold flag sensitivity. High on **both** the prior and likelihood components indicates potential prior-data conflict; high prior with low likelihood indicates a strong prior / weak likelihood.

When the fit recorded per-draw hyperparameter log-prior values at draw-synthesis time ($hyper_log_prior_draws, stored by the nested-Laplace mixture paths such as tulpa_re_cov_nested() and the tulpa() random-slope redirect), a hyperparameter column reports the power-scaling sensitivity of the hyperparameter prior by the same reweighting; NA otherwise.

Usage

```
tulpa_powerscale_sensitivity(
  fit,
  prior = NULL,
  lower_alpha = 0.99,
  upper_alpha = 1.01,
  threshold = 0.05
)
```

Arguments

`fit`	A `tulpa_fit` fitted through `tulpa()` (fixed-effect / GLMM; spatial / temporal-field fits and nested-Laplace outer-grid mixture fits, e.g. a `latent()` block or a random-slope redirect, are rejected).
`prior`	The Gaussian fixed-effect prior the fit used, as `list(mean =, sd =)` (scalars recycled). Required for the prior component; omit to compute the likelihood component only.
`lower_alpha, upper_alpha`	Power-scaling grid endpoints for the gradient (defaults 0.99 / 1.01, as in priorsense).
`threshold`	Sensitivity flag threshold (default 0.05).

Value

A data frame with one row per fixed-effect parameter and columns `variable`, `prior`, `hyperparameter`, `likelihood`, `diagnosis`.

References

Kallioinen, Paananen, Buerkner & Vehtari (2024). Detecting and diagnosing prior and likelihood sensitivity with power-scaling. *Statistics and Computing* 34:57. Nguyen & Vreeken (2015). Nonparametric Jensen-Shannon divergence. ECML PKDD.

See Also

`tulpa_psis()`, `tulpa_criteria()`.

Examples

```
set.seed(1)
d <- data.frame(x = rnorm(150))
d$y <- rpois(150, exp(0.5 + 0.7 * d$x))
fit <- tulpa(y ~ x, data = d, family = "poisson", mode = "laplace",
             beta_prior = list(mean = 0, sd = 5))
tulpa_powerscale_sensitivity(fit, prior = list(mean = 0, sd = 5))
```

tulpa_priors *Prior specification for tulpa models*

Description

Specify priors for model parameters. Supports both PC (penalized complexity) priors for variance components and standard distributions for other parameters.

A `tulpa_priors` object is what the simulation tools draw parameters from: `prior_predict()` and `tulpa_simulate()` with `theta = NULL`. The model fitters do not read one. `tulpa()` carries its own defaults, set per backend through its `beta_prior`, `re_prior` and `hyperprior` arguments:

- Fixed effects: `prior_normal(0, 2.5)` on every backend – the same default as `beta` here.
- Random-effect SDs: a PC prior with `P(sigma > 3) = 0.01` (plus an LKJ(2) prior on a correlated block's correlations) on the routes that integrate or maximize over them (nested Laplace, `re_cov_nested`, `mode = "eb"`); half-Cauchy(2.5) on the samplers (`re_prior$sigma_re_scale`, and `re_prior$prior_sigma_scale` for `mode = "gibbs"`); an inverse-Wishart under `control$re_cov = "gibbs"`.

The `sigma`, `phi`, `rho_temporal` and `rho_spatial` defaults below are therefore the simulation defaults, not the priors a fit ran under.

Usage

```
tulpa_priors(
  beta = NULL,
  sigma = NULL,
  phi = NULL,
  rho_temporal = NULL,
  rho_spatial = NULL
)

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

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

## S3 method for class 'tulpa_prior_normal'
format(x, indent = "", ...)

## S3 method for class 'tulpa_prior_half_normal'
format(x, indent = "", ...)

## S3 method for class 'tulpa_prior_half_cauchy'
format(x, indent = "", ...)

## S3 method for class 'tulpa_prior_gamma'
```

```
format(x, indent = "", ...)

## S3 method for class 'tulpa_prior_exponential'
format(x, indent = "", ...)

## S3 method for class 'tulpa_prior_beta'
format(x, indent = "", ...)

## S3 method for class 'tulpa_prior_pc'
format(x, indent = "", ...)

## S3 method for class 'tulpa_prior'
format(x, indent = "", ...)
```

Arguments

`beta`	Prior for fixed effects. Default: `prior_normal(0, 2.5)`.
`sigma`	Prior for random effect SDs. Default: PC prior with P(sigma > 1) = 0.01.
`phi`	Prior for overdispersion parameter. Default: PC prior with P(phi > 10) = 0.01.
`rho_temporal`	Prior for temporal autocorrelation. Default: `prior_beta(2, 2)` centered at 0.5.
`rho_spatial`	Prior for spatial proportion (BYM2). Default: `prior_beta(1, 1)` (uniform).
`x`	A `tulpa_priors` specification, or a single prior built by a `prior_*()` constructor.
`...`	Ignored.
`indent`	String prefixed to each formatted line.

Details

PC priors (Simpson et al., 2017) provide principled regularization that:

- Favors simpler models (smaller variance components)
- Has interpretable parameters (tail probabilities)
- Prevents overfitting with sparse data

For other parameters, standard distributions are available via helper functions: `prior_normal()`, `prior_half_normal()`, `prior_half_cauchy()`, `prior_gamma()`, `prior_beta()`, `prior_exponential()`.

Value

A `tulpa_priors` object

`format()` returns the prior as a character string.

References

Simpson, D., Rue, H., Riebler, A., Martins, T. G., & Sorbye, S. H. (2017). Penalising model component complexity: A principled, practical approach to constructing priors. Statistical Science, 32(1), 1-28.

Examples

```
# Default priors
tulpa_priors()

# Custom fixed effect prior
tulpa_priors(beta = prior_normal(0, 1))

# Tighter random effect prior
tulpa_priors(sigma = prior_pc(U = 0.5, alpha = 0.01))

# Half-Cauchy for random effect SD
tulpa_priors(sigma = prior_half_cauchy(2.5))

# Informative prior for temporal correlation
tulpa_priors(rho_temporal = prior_beta(5, 2))  # Prior mode at ~0.8
```

---

tulpa_profile *Profile a fit by solver phase*

Description

Times a fit one solver phase at a time – the inner Newton solve's eta, scatter (the Hessian and gradient assembly), factorize (Cholesky), line search and final pass, the nested-Laplace outer grid's cell solves, and the NUTS sampler's iterations and gradient evaluations – and returns the breakdown as a data frame. The accumulator aggregates across the parallel outer-grid worker threads and across concurrently run chains, so the reported times cover the whole fit rather than only the calling thread (and can therefore exceed its wall time).

Usage

```
tulpa_profile(expr, sort = TRUE)
```

Arguments

`expr`	An expression that runs a fit (for example a call to `tulpa_laplace()` or `tulpa()`). Evaluated once, after the profile counters are reset and timing is switched on.
`sort`	Logical; order rows by descending time. Default `TRUE`.

Details

Use it to settle where a fit spends its time, e.g. whether a slow joint `occu_cover()` fit is bound by the assembly scatter or the Cholesky factorize:

```
p <- tulpa_profile(
  tulpa_nested_laplace_joint(..., control = list(integration = "ccd"))
)
print(p)            # rows ordered by time; scatter vs factorize at top
fit <- attr(p, "value")
```

Timing is off outside `tulpa_profile()`: an ordinary fit pays one flag read per phase scope and nothing else.

Value

A data frame with one row per phase and columns `phase`, `seconds`, `calls`, `ms_per_call` (mean wall time per phase call), and `share` (fraction of the total timed seconds of the leaf phases; `NA` for an enclosing phase). The fit result is attached as the `"value"` attribute.

Which fits are timed

- The single-response Laplace solve behind `tulpa_laplace()`, `tulpa(mode = "laplace")` and every per-cell inner solve of `tulpa_nested_laplace()` / `tulpa(mode = "nested_laplace")`: phases `eta`, `scatter`, `factorize`, `line_search`, `log_det`, `log_lik_prior`, `hessian_extract` and `inner_diagnostics`.
- The joint solver behind `tulpa_nested_laplace_joint()`, on both its dense and its sparse path (the sparse one adds `pattern_build` and `prep`).
- The nested-Laplace outer grid, single-response and joint: `outer_grid_cell`, one call per solved cell.
- The samplers: `nuts_warmup` and `nuts_sampling`, one call per NUTS iteration per chain, and `gradient`, one call per log-density gradient evaluation (NUTS leapfrog steps, static HMC, the step-size search and the other samplers that evaluate the gradient through the engine).

An expression that reaches none of these – a Gibbs or quadrature backend that runs neither an engine Newton solve nor an engine gradient, or no fit at all – records nothing, and `tulpa_profile()` then warns rather than returning an all-zero table as if it were a measurement.

Enclosing phases

`outer_grid_cell`, `nuts_warmup` and `nuts_sampling` ENCLOSE the leaf phases timed inside them (a cell's inner Newton phases, an iteration's gradient evaluations), so their seconds overlap the leaves' rather than adding to them. Their `share` is `NA`; the shares of the leaf phases sum to one.

Examples

```
set.seed(1)
n <- 200L; X <- cbind(1, rnorm(n))
y <- rbinom(n, 1, plogis(X %*% c(0, 0.5)))
tulpa_profile(tulpa_laplace(y, rep(1L, n), X, family = "binomial"))
```

`tulpa_psis` *Pareto-smoothed importance sampling*

Description

Smooths a set of importance ratios by replacing the largest weights with the order statistics of a generalized-Pareto fit to their upper tail, and returns the Pareto shape diagnostic `pareto_k` together with the smoothed (normalized) log weights and their importance-sampling effective sample size. `pareto_k` estimates the number of finite moments of the raw weight distribution: `< 0.5` is good (finite variance). The usable upper boundary is sample-size dependent, `min(1 - 1/log10(S), 0.7)` for `S` draws (Vehtari et al. 2024): about 0.565 at `S = 200`, reaching the 0.7 cap only past `S` ~ 2154. Above it the proposal cannot be reliably corrected to the target.

Usage

```
tulpa_psis(log_ratios, tail_points = NULL)
```

Arguments

`log_ratios`	Numeric vector of (unnormalized) log importance ratios `log p_target(x) - log q_proposal(x)` evaluated at draws `x ~ q`. `-Inf` marks a draw of zero weight (it keeps its place and counts towards `S`); `NA` / `NaN` is an error. A `+Inf` ratio is the worst possible one and returns `pareto_k = Inf` with all the weight on the `+Inf` draws.
`tail_points`	Number of upper-tail order statistics for the generalized-Pareto fit, or `NULL` (default) for the automatic PSIS rule `ceil(min(0.2 * S, 3 * sqrt(S)))`. An explicit value is an expert tail-threshold control, capped at `floor(0.2 * S)` so the fit stays an extreme tail; it is NOT a precision knob (raise the draw count for a tighter shape estimate).

Value

A list with `pareto_k` (the tail shape, `NA` if the sample is too small to fit), `is_ess` (importance-sampling effective sample size, `1 / sum(w^2)` on the normalized smoothed weights), `log_weights` (the normalized smoothed log weights, one per element of `log_ratios` and in the same order; with fewer than 5 draws the raw ratios, normalized), `tail_len` (the tail size used, `0` when no tail was fitted), and `tail_smoothed` (`FALSE` when the tail kept its raw log ratios because the generalized-Pareto fit was not attempted or returned a shape / scale the quantile function is undefined at; `pareto_k` then reports the attempted fit and does not describe the returned weights).

References

Vehtari, Simpson, Gelman, Yao & Gabry (2024). Pareto smoothed importance sampling. *JMLR* 25(72):1-58.

See Also

`diagnostics()` for the fit-level diagnostic front door.

Examples

```
set.seed(1)
# Well-behaved importance ratios: k-hat is small.
ps <- tulpa_psis(rnorm(2000))
```

```
ps$pareto_k
ps$is_ess
```

---

tulpa_reloo *Selective refit of high-Pareto-k observations (reloo)*

Description

PSIS-LOO with exact refits where the importance sampling is unreliable: observations whose Pareto k-hat exceeds `k_threshold` are re-scored by refitting the model without that observation (through the fit's stored `tulpa()` call, as in `tulpa_kfold()`) and evaluating the exact held-out log predictive density. All other observations keep their PSIS-LOO value, so the cost is one refit per flagged observation rather than per fold.

The same restrictions as `tulpa_kfold()` apply: fixed-effect / GLMM fits only (subsetting breaks a spatial / temporal field), and held-out random-effect groups contribute at their prior mean.

Usage

```
tulpa_reloo(
  object,
  data,
  k_threshold = .nl_diag("k_usable"),
  n_trials = NULL,
  ndraws = NULL
)
```

Arguments

`object`	A `tulpa_fit` fitted through `tulpa()` (must carry `$call`).
`data`	The data frame the model was fit to.
`k_threshold`	Pareto k-hat above which an observation is refit (default 0.7, the standard PSIS reliability gate).
`n_trials`	Optional binomial denominators (length `nrow(data)`); defaults to the trials stored on the fit, else 1.
`ndraws`	Number of posterior draws used for the PSIS-LOO baseline (defaults to all stored draws, or 400 on the draw-free Laplace tier).

Value

A list with `elpd_loo` (corrected), `se_elpd_loo`, `looic`, `pointwise` (per-observation elpd, exact at the refit observations), `reloo_idx` (indices refit), `pareto_k` (the original k-hat values), and `k_threshold`.

See Also

`tulpa_kfold()` for the full refit-CV; `tulpa_criteria()` for PSIS-LOO / WAIC without refits.

Examples

```
set.seed(1)
d <- data.frame(x = rnorm(120))
d$y <- rpois(120, exp(0.4 + 0.6 * d$x))
fit <- tulpa(y ~ x, data = d, family = "poisson", mode = "laplace")
rl  <- tulpa_reloo(fit, data = d)
rl$elpd_loo
```

`tulpa_re_aghq` *Adaptive Gauss-Hermite refinement of a grouped random-effect covariance*

Description

Refines a generalized linear mixed model's fixed effects and random-effect covariance by replacing the per-group Laplace integral with `n_quad`-point adaptive Gauss-Hermite quadrature (AGHQ). At `n_quad = 1` this is the joint Laplace (glmer `nAGQ = 1`); higher `n_quad` reduces the small-cluster attenuation of the variance components for binary / count data. Unlike `agq_fit()` (intercept-only RE, built-in `binomial`/`poisson`/`gaussian` likelihoods), this engine is **callback-driven**: the caller supplies the per-group conditional likelihood, so a custom marginal (e.g. a latent-state-integrated occupancy / detection likelihood, or the latent-abundance-integrated N-mixture marginal) refines through the same quadrature.

This is an engine block, not a front door: model packages call it programmatically, so its tuning knobs (`max_iter`, `n_quad`, `keep`) sit in the signature rather than in a `control` list.

The engine is **structure-agnostic**. It integrates the per-group marginal

$$M_g = \int \exp\{\ell_g(b_g)\}\, N(b_g; 0, \Sigma)\, db_g,$$

where b_g is the group's random-effect vector (dimension $\sum_m c_m$ over the RE terms) and $\ell_g(b_g)$ is the group's conditional log-likelihood when its linear predictors are perturbed by b_g. How b_g enters the likelihood – through one linear predictor, or through several coupled arms at different observation granularities (e.g. a per-site abundance arm and a per-visit detection arm sharing a species grouping) – lives entirely in the callback. The engine only needs, per group, the value / gradient / Hessian of ℓ_g in b_g (for the mode) and ℓ_g at the quadrature nodes (for the sum). The fixed parameters `theta` and the log-Cholesky coordinates of Σ are optimized jointly on $\sum_g \log M_g$; standard errors come from the exact-marginal Hessian.

Two callback forms select the structure (supply exactly one):

- `make_site` – the common **single-arm, per-row-separable** case ($\ell_g(b_g) = \sum_{i \in g} \log f_i(\eta_i + Z_i b_g)$ for one linear predictor η). The engine builds the oracle from the per-row marginal and the RE design `Z` itself.
- `make_group` – the **general / multi-arm** case. The caller supplies the per-group b-space oracle directly, so non-separable units (e.g. the visits of an N-mixture site coupled through the shared latent count) and random effects on several arms at once are handled with no engine change.

Scope: one shared grouping factor across all RE terms (the per-group integral factorizes). The total RE dimension per group should be small (the quadrature grid is `n_quad^dim`).

Usage

```
tulpa_re_aghq(
  theta0,
  re_terms,
  Sigma0,
  make_site = NULL,
  make_group = NULL,
  oracle = NULL,
  n_obs = NULL,
  keep = NULL,
  n_quad = 9L,
  lkj_eta = 1,
  theta_prior_sd = Inf,
  sigma_prior = NULL,
  gradient = c("fd", "analytic"),
  max_iter = 200L
)
```

Arguments

`theta0` Initial fixed-parameter vector. The engine optimizes these jointly with the RE covariance; the callback interprets them.

`re_terms` A list of RE term specs (or one spec), each defining a covariance block: `n_coefs` (block dimension c_m), optional `correlated` (default `TRUE` for `c_m > 1`; `FALSE` gives a diagonal block), and `n_groups` (shared across terms). For the `make_site` path each term also carries `idx` (1-based group index, length `n_obs`) and, for a slope block, `Z` (the `n_obs x n_coefs` design). For the `make_group` path the per-observation `idx` / `Z` are optional – the callback owns them – and the term needs only `n_coefs` / `correlated` / `n_groups`.

`Sigma0` List of initial per-term covariance matrices (the EM estimate).

`make_site` `function(theta)` for the single-arm separable case, returning a list with: `eta_re` (length `n_obs`, the RE-arm fixed predictor), `deriv = function(rows, eta)` returning `list(logL, d1, d2)` (per-row marginal log-likelihood and its first/second derivatives w.r.t. the RE-arm predictor `eta`, used for the per-group mode), and `lmat = function(rows, ETA)` returning a `length(rows) x ncol(ETA)` matrix of per-observation log-likelihoods over the quadrature node columns. Supply this or `make_group`, not both.

`make_group` `function(theta)` for the general / multi-arm case, returning a list with two per-group closures (let `d = sum(n_coefs)` be the group RE dimension):

- `grad_hess(g, b)` – for group `g` at RE value `b` (length `d`), the list `list(logL, grad, negH)`: the group conditional log-likelihood $\ell_g(b)$, its gradient $\partial\ell_g/\partial b$ (length `d`), and the **data-only** observed information $-\partial^2\ell_g/\partial b^2$ (`d x d`; the engine adds the Σ^{-1} prior curvature).
- `node_ll(g, B)` – for group `g`, a numeric vector of length `nrow(B)` giving ℓ_g at each quadrature node (rows of the `nrow x d` matrix `B` are candidate `b` vectors). The callback owns all arm / design / clamping bookkeeping. Supply this or `make_site`, not both.

`oracle` Optional prebuilt native (compiled) oracle, an external pointer to a `REGroupOracle` (constructed in a consumer package's src/ via `LinkingTo: tulpa` against `<tulpa/aghq_oracle.h>`). When supplied the engine drives it directly, with no per-group / per-node round trip into R, and neither `make_site` nor `make_group` is needed; `re_terms`, `theta0` and `Sigma0` must still describe the same layout the oracle exposes. The integration core is identical to the R-closure path.

`n_obs` Number of observations (length of each term's `idx`). Required for the `make_site` path; ignored for `make_group`.

`keep` Optional logical/integer mask of observations to include (default all; `make_site` path only). Rows outside `keep` are dropped from every group.

`n_quad` Quadrature nodes per RE dimension. Either a single integer (default 9; `1` = Laplace) broadcast to every covariance block, or an integer vector of length `length(re_terms)` giving a per-block node count. The tensor grid then uses `n_quad[b]` nodes along every dimension of block `b`, for `prod_b n_quad[b]^(dim_b)` total nodes; a scalar reproduces the uniform grid exactly. Per-block orders let a heterogeneous stack spend fewer nodes on cheap scalar nuisance blocks than on the correlated coefficient blocks (e.g. `c(3, 3, 2, 2)` on blocks of dimension `2, 2, 1, 1` gives `3^2 * 3^2 * 2 * 2 = 324` nodes rather than `3^6 = 729`).

`lkj_eta` LKJ shape for an optional correlation penalty on each *correlated* block (log-density `(eta - 1) log det R`, maximized at independence). `1` disables it; `> 1` regularizes a weakly-identified correlation off the boundary without touching the marginal SDs. The marginal SDs are otherwise unpenalized (pure ML), so the refinement debiases them rather than shrinking them.

`theta_prior_sd` Optional Gaussian ridge SD on the fixed parameters `theta` (a mean-zero `N(0, theta_prior_sd^2)` prior, added to the optimized objective and hence the marginal Hessian). `Inf` (default) is pure ML on `theta`; a large finite value (e.g. 100) is a weak ridge that stabilizes a weakly-identified fixed effect without materially shifting the estimate.

`sigma_prior` Optional Penalized-Complexity prior on the marginal standard deviations of one or more RE covariance blocks, added to the objective (and hence the marginal Hessian). `NULL` (default) is pure ML on the covariances – the refinement debiases the SDs rather than shrinking them. Otherwise a `c(U, alpha)` pair (`P(sigma_i > U) = alpha`, the same convention as `re_cov_pc_lkj_prior()`) applied to every block, or a list `list(blocks = <integer indices>, prior_sigma = c(U, alpha))` applied to the named blocks only. Reuses the exact PC log-prior + Jacobian of `re_cov_pc_lkj_prior()`. A weakly-identified variance component (e.g. a scalar dispersion / zero-inflation random effect at few groups) can drift to the boundary and flatten the marginal Hessian; a weak PC prior adds curvature there (the `+ log sigma` Jacobian repels `sigma -> 0`, the `- lambda sigma` term caps inflation), keeping the joint optimum non-singular without materially shifting an identified fit.

`gradient` How `stats::optim` gets the gradient of the AGHQ objective. `"fd"` (default) lets `optim` finite-difference the objective – correct at every `n_quad` and the only option for the R-closure (`make_site` / `make_group`) paths. `"analytic"` supplies the Fisher-identity gradient (posterior-weighted theta-score plus the `Sigma` moment-matching residual), which avoids the per-coordinate objective re-solve and so is far cheaper for the quadrature debias. It requires a prebuilt native

oracle (the only one exposing the theta-score) and n_quad > 1: being the gradient of the true marginal it omits the node-placement terms (0 of the AGHQ truncation), so it agrees with the objective only as n_quad grows.

max_iter Optimizer iteration cap (default 200).

Value

A list with: theta (refined fixed parameters), Sigma_list (refined per-term covariance), blup / blup_var (per-term n_groups x n_coefs posterior mean / variance of the RE), group_ok (logical, length n_groups: FALSE where that group's mode search or precision factorization failed, which is what the NA rows of blup, blup_var, blup_cov_g and blup_cross_g mean – a caller conditions its per-group reads on this rather than on trapping the accompanying warning), blup_cross (per-term n_groups x n_theta x n_coefs array: the mode/theta cross-Hessian block Bf, -d^2 ell_g / d theta db at each group's mode, in the same negative-Hessian sign convention as the posterior precision underlying blup_var – so a joint draw of theta and a group's RE b_g uses b_g | theta ~ N(blup_g - Cinv_g %*% t(Bf_g) %*% (theta_draw - theta), Cinv_g) with Cinv_g the group's n_coefs x n_coefs posterior covariance block. NA throughout when blup_cross_available is FALSE: the cross-Hessian needs the oracle's analytic theta_score, which the R-closure bridge (make_site / make_group) does not supply – only a prebuilt native oracle carries it), blup_cov_g / blup_cross_g (per-group lists, length n_groups, of the FULL joint posterior covariance (d x d, d = every RE term's width combined) and mode/theta cross-Hessian (n_theta x d) across ALL RE terms sharing that group – the superset blup_var/blup_cross reduce to a per-term diagonal block of when a group carries more than one term, since a group's terms are found jointly and can carry real posterior covariance BETWEEN terms (e.g. an abundance-arm and a detection-arm term sharing one grouping factor); same NA-when-unavailable rule as blup_cross), theta_cov / theta_se (fixed-parameter covariance / SE from the marginal Hessian), re_par / re_par_cov / re_par_se (the RE-covariance coordinates the optimizer carried – log-Cholesky for a full block, log-SD for a diagonal one – with their block of the same inverse Hessian, so SE(log sigma) is available for a boundary test on a weakly-identified variance component), re_par_layout (per block: label, nc, full, the index range into re_par and the coord names, so a caller does not reconstruct the packing), joint_cov (the whole (n_theta + n_chol) inverse Hessian), log_marginal (the AGHQ marginal log-likelihood at the optimum, excluding any ridge), n_quad, lkj_eta, converged, and counts (stats::optim's own function / gradient evaluation counts, so a caller reporting how much work the fit took has a number to report rather than NA). RE terms that do not share one grouping factor are an input error and stop. Three conditions warn and return NULL (caller keeps its prior fit): a singular / non-finite optimum, an objective that is already undefined at the starting parameters (some group's solve fails there, so there is nothing to descend), and an optimum whose objective is the failure sentinel rather than an attained marginal likelihood – the last two report which groups failed.

References

Pinheiro & Bates (1995). Approximations to the log-likelihood function in the nonlinear mixed-effects model. *Journal of Computational and Graphical Statistics* 4(1):12-35. Lewandowski, Kurowicka & Joe (2009). Generating random correlation matrices based on vines and extended onion method. *Journal of Multivariate Analysis* 100(9):1989-2001.

Examples

```
# A per-row-separable binomial GLMM marginal supplied through `make_site`.
```

```
l1pe <- function(x) ifelse(x > 0, x + log1p(exp(-x)), log1p(exp(x)))
make_binom_site <- function(X, y, nt) function(theta) {
  eta_fixed <- as.numeric(X %*% theta)
  list(eta_re = eta_fixed,
       deriv = function(rows, eta) {
         p <- plogis(eta)
         list(logL = y[rows] * eta - nt[rows] * l1pe(eta),
              d1 = y[rows] - nt[rows] * p, d2 = -nt[rows] * p * (1 - p))
       },
       lmat = function(rows, ETA) y[rows] * ETA - nt[rows] * l1pe(ETA))
}
set.seed(1)
ng <- 30L; npg <- 8L; n <- ng * npg
g <- rep(seq_len(ng), each = npg); x <- rnorm(n)
X <- cbind(1, x); nt <- rep(3L, n); u <- rnorm(ng, 0, 0.9)
y <- rbinom(n, nt, plogis(0.3 + 0.7 * x + u[g]))
fit <- tulpa_re_aghq(theta0 = c(0, 0),
                     re_terms = list(list(idx = g, n_groups = ng, n_coefs = 1L)),
                     Sigma0 = list(matrix(0.25, 1, 1)),
                     make_site = make_binom_site(X, y, nt), n_obs = n, n_quad = 5L)
sqrt(fit$Sigma_list[[1]][1, 1])     # adaptive-GHQ RE standard deviation
```

`tulpa_re_cov_gibbs` *Gibbs estimation of random-effect covariances (exact-target debias)*

Description

For one or more random-effects terms (e.g. `(1 + x | g)`, `(1 + x || g)`, or several terms together), estimate the random-effect covariances `Sigma` by sampling the exact joint posterior `p(beta, {b}, {Sigma} | y)` rather than fixing each `Sigma` at the Laplace mode. This removes the Laplace / PQL "approximation" bias that shrinks variance components low for binary and low-count responses with small groups.

Usage

```
tulpa_re_cov_gibbs(
  y,
  n_trials = NULL,
  X,
  re_terms,
  family = "binomial",
  phi = 1,
  prior_df = NULL,
  prior_scale = NULL,
  beta_prior = .tulpa_default_beta_prior("re_cov_gibbs"),
  control = list()
)
```

Arguments

`y`, `n_trials`, `X`, `family`, `phi`
: Passed to the likelihood and to `tulpa_laplace()` for the pilot solve. `n_trials = NULL` defaults to 1.

`re_terms`
: Either a single random-effect term or a list of them. Each term is a list with `idx` (1-based group index per observation), `n_groups`, `n_coefs` (`c`), `Z` (the `n_obs x c` RE design; only required when `c > 1`), and `correlated` (`TRUE` for a full `Sigma`, `FALSE` for a diagonal one; defaults to `TRUE`). An optional `label` / `group_var` names the block. Any supplied `L` / `cov` / `sigma` is ignored – `Sigma` is what this function samples.

`prior_df`
: Inverse-Wishart prior degrees of freedom. Applied to every correlated block (default `n_coefs + 1`, the minimal proper choice) and as the scalar inverse-gamma shape for every diagonal block (default 2). Must leave each block's prior proper – unlike `tulpa_re_cov_nested()` / `tulpa_eb()` (`hyperprior = "flat"` by default), the `Sigma_m | b_m` conjugate draw here needs a proper Inverse-Wishart to sample from, so an improper flat prior is not an option; the minimal-`df` default is the closest analogue this sampler can offer.

`prior_scale`
: Inverse-Wishart prior scale matrix. Used for a block when its dimension matches (default `diag(n_coefs)`); otherwise the per-block default is used.

`beta_prior`
: Gaussian fixed-effect prior as `list(mean, sd)` (default the engine default, `prior_normal(0, 2.5)`). Scalar `mean` / `sd` are recycled to `ncol(X)`; a length-`ncol(X)` vector sets a per-coefficient prior.

`control`
: A named list of numerical / tuning knobs (statistical arguments stay in the signature above). Recognized entries:

- `n_iter`: recorded post-warmup sweeps (default 2000).
- `warmup`: warmup (burn-in) sweeps, used for proposal-scale adaptation (default 1000).
- `thin`: keep every `thin`-th recorded sweep (default 1).
- `seed`: optional integer seed for reproducibility.
- `max_iter`, `tol`, `n_threads`: pilot-solve controls (see `tulpa_laplace()`).

Details

Metropolis-within-Gibbs targeting `p(beta, {b_m}, {Sigma_m} | y)`:

- `b_{m,g} | beta, Sigma, y` – random-walk Metropolis per (term, group) (groups are conditionally independent given `beta` and the covariances). The proposal *shape* is the Laplace per-group posterior covariance block (`return_re_cov` from `tulpa_laplace()`); the Metropolis acceptance is computed against the exact group log-likelihood plus the Gaussian RE log-prior, which corrects the non-Gaussianity the Laplace approximation misses. The linear predictor holds every other term's contribution fixed.
- `beta | b, Sigma, y` – random-walk Metropolis, proposal shape from the Laplace fixed-effect Hessian block `H_beta`.
- `Sigma_m | b_m` – exact conjugate draw. A correlated block draws the full matrix from `IW(prior_df + G_m, prior_scal` an uncorrelated (diagonal) block draws each variance from its scalar conjugate (inverse-gamma). No linearization enters this step.

A single Laplace solve provides the starting values (beta, b) and the proposal shapes; the random-walk scales for the beta block and the per-term b blocks are adapted toward their target acceptance during burn-in (Robbins-Monro), then held fixed for the recorded sweeps. The covariance summary marginalizes the derived scale / correlation parameters over the posterior draws via the same machinery as tulpa_re_cov_nested().

For family = "gaussian" the response is already conditionally Gaussian, so there is no Laplace bias to remove; phi (the residual variance) is treated as known. The sampler still runs and is useful as a reference / for Sigma uncertainty.

Value

A list with:

- posterior: data frame with one row per parameter (sigma_i, rho_ij, Sigma_ij; prefixed by block label when there are several terms; diagonal blocks report no rho) and columns mean, sd, median, ci_lo, ci_hi.
- Sigma_mean: the posterior mean of Sigma (a matrix for one block, a named list of matrices for several).
- Sigma_draws: list of recorded draws; each element is the per-block list of Sigma matrices for that sweep.
- beta_draws / draws: matrix of recorded beta draws; draws (with means, param_names, process_info) drives the generic tulpa_fit methods.
- re: matrix of recorded random-effect draws, n_kept rows by one column per (block, group, coefficient) in that order – the exact per-group posterior the sweep samples. Row-aligned with beta_draws, so a draw is a joint (beta, b) state. This is what ranef() summarizes and what posterior_predict() adds to the linear predictor.
- accept: list with beta and b acceptance rates over recorded sweeps.
- n_kept, n_coefs (vector of per-block c), prior: bookkeeping.

References

Lewandowski, Kurowicka & Joe (2009). Generating random correlation matrices based on vines and extended onion method. *Journal of Multivariate Analysis* 100(9):1989-2001.

See Also

tulpa_re_cov_nested() for the grid-integration (summary-bias) fix; tulpa_laplace() for the pilot solve and per-group covariance blocks.

Examples

```
set.seed(1)
G <- 20L; per <- 12L; n <- G * per
grp <- rep(seq_len(G), each = per); x <- rnorm(n)
b <- cbind(rnorm(G, 0, 0.7), rnorm(G, 0, 0.5))     # random intercept + slope
eta <- -0.2 + 0.5 * x + b[grp, 1] + b[grp, 2] * x
y <- rbinom(n, 1L, plogis(eta))
re_term <- list(idx = grp, n_groups = G, n_coefs = 2L, Z = cbind(1, x),
```

```
                 correlated = TRUE)
fit <- tulpa_re_cov_gibbs(y, rep(1L, n), cbind(1, x), re_term,
                          family = "binomial",
                          control = list(n_iter = 300L, warmup = 150L))
fit$Sigma_mean        # exact-debias RE covariance posterior mean
```

`tulpa_re_cov_nested` *Nested-Laplace integration over random-effect covariances*

Description

For one or more random-effects terms (e.g. `(1 + x | g)`, `(1 + x || g)`, or several terms together), integrate the Laplace marginal likelihood over the random-effect covariances `Sigma` instead of fixing them at point estimates. Reports weighted posterior summaries (mean, SD, median, 2.5\ `Sigma` and its derived scale (`sigma_i`) and correlation (`rho_ij`) parameters, marginalizing the joint posterior over a `Sigma`-grid.

This corrects the plug-in-MAP ("summary") bias: the mode of a skewed variance-component marginal is biased low relative to its median, so the headline summary should be the marginalized median, not the mode.

Usage

```
tulpa_re_cov_nested(
  y,
  n_trials = NULL,
  X,
  re_terms,
  family = "binomial",
  phi = 1,
  phi2 = NULL,
  prior_sigma = NULL,
  eta = NULL,
  hyperprior = c("proper", "flat"),
  log_prior_theta = NULL,
  beta_prior = NULL,
  offset = NULL,
  weights = NULL,
  n_quad = 1L,
  X_zi = NULL,
  zi_prior_sd = 2.5,
  control = list()
)
```

Arguments

`y`, `n_trials`, `X`, `family`, `phi`

Passed to `tulpa_laplace()` for the inner solve. `n_trials = NULL` defaults to 1 (binary / single-trial).

`re_terms`

Either a single random-effect term or a list of them. Each term is a list with `idx` (1-based group index per observation), `n_groups`, `n_coefs` (`c`), `Z` (the `n_obs x c` RE design, e.g. `cbind(1, x)` for `(1 + x | g)`; only required when `c > 1`), and `correlated` (`TRUE` for a full `Sigma`, `FALSE` for a diagonal one; defaults to `TRUE`). An optional `label` / `group_var` names the block in the output. Any `L` / `cov` / `sigma` field is ignored – `Sigma` is what this function integrates over.

`phi2`

Optional second dispersion, threaded into every inner `tulpa_laplace()` solve: the Student-t degrees of freedom (`family = "t"`, default 4 when `NULL`) or the Tweedie variance power (`family = "tweedie"`, required – a defaulted power would be a statistical decision the caller never made). A `phi2` supplied for any other family errors rather than being ignored. It is conditioned on: the integration is over the random-effect covariances, not over `phi2`.

`prior_sigma`, `eta`

Hyperparameters of the PC + LKJ prior used when `hyperprior = "proper"` (see `re_cov_pc_lkj_prior()`): `prior_sigma = c(U, alpha)` with `P(sigma_i > U) = alpha` (`NULL`, the default, is `c(3, 0.01)`) and LKJ shape `eta` (`NULL` is 2). Ignored when `hyperprior = "flat"` or `log_prior_theta` is supplied.

`hyperprior`

`"proper"` (default) or `"flat"`. `"proper"` builds the PC + LKJ prior from `prior_sigma` / `eta`, the proper prior every other scale axis of the engine carries by default. `"flat"` integrates with `log_prior_theta` the zero function (flat in log(theta)); that prior is improper, so the fit then reports no evidence. Ignored when `log_prior_theta` is supplied.

`log_prior_theta`

Optional `function(theta)` returning a scalar log prior density on the full stacked parameter vector, overriding `hyperprior` entirely. Default `NULL`, which defers to `hyperprior`.

`beta_prior`

Optional Gaussian prior on the fixed effects, threaded into every inner `tulpa_laplace()` solve (`list(mean, sd)`). `NULL` (default) keeps the weak built-in prior.

`offset`

Optional observation-level offset on the linear predictor (length `length(y)`), e.g. `log(exposure)` for a rate model. Not supported with `n_quad > 1`, which errors rather than dropping it.

`weights`

Optional non-negative observation weights (length `length(y)`), each scaling its row's log-likelihood in every inner `tulpa_laplace()` solve, so the covariance is integrated under the weighted likelihood. Not supported with `n_quad > 1`, which errors rather than dropping them.

`n_quad`

Quadrature order for the inner marginal. `1` (default) uses the joint-field Laplace inner solve (`tulpa_laplace()`). `> 1` refines the inner marginal with `n_quad`-point adaptive Gauss-Hermite quadrature (the `tulpa_re_aghq()` debias applied inside the `Sigma` integration), reducing the small-cluster variance attenuation for binary / low-count data. AGHQ requires a single shared grouping factor (the per-group integral must factorize); with crossed RE terms `n_quad > 1` errors.

When AGHQ is used the fixed effects are integrated, so the reported fixed-effect posterior is the marginal (ML-II) one rather than the joint-mode (PQL) estimate.

`X_zi` Optional zero-inflation design matrix (`length(y)` rows), making the model a two-process mixture: each observation is a structural zero with probability `plogis(X_zi beta_zi)` and otherwise follows `family`. Paired with a zero-truncated family it is the hurdle model. The random effects enter the count predictor only, and the integration runs over the same covariance coordinates – the mixture changes the inner solve, not the parameters being integrated over. The ZI coefficients are reported alongside the count ones in `coef()` / `vcov()`, so the fixed block is `ncol(X)` + `ncol(X_zi)` wide. Needs `n_quad = 1`: the adaptive Gauss-Hermite inner marginal runs through a single-predictor oracle.

`zi_prior_sd` Prior SD on `beta_zi`, keeping the logit identified where a level carries no zeros (the likelihood alone would send it to `-Inf`). Ignored when `X_zi` is `NULL`.

`control` A named list of numerical / tuning knobs (statistical arguments stay in the signature above). Recognized entries:

- `integration`: node layout, `"ccd"` (default, central-composite design, scales to larger total parameter count) or `"grid"` (full tensor product).
- `n_per_axis`: points per parameter axis in the tensor grid (default 5); used only when `integration = "grid"`.
- `span`: half-width of the tensor grid in posterior standard deviations per whitened axis (default 3); grid only.
- `n_draws`: posterior draws of the fixed effects synthesized from the node mixture (default 2000), exposed as `draws` for the generic `tulpa_fit` methods. The `Sigma` posterior is summarized directly from the integration nodes in `posterior`, independent of `n_draws`.
- `seed`: optional integer seed for the fixed-effect draw synthesis.
- `diagnose_k`: if `TRUE` (default), compute the outer Pareto k-hat accuracy diagnostic for the Gaussian proposal over the hyperparameters, returned as `pareto_k`. Several proposal candidates are scored and the best is kept: `pareto_k_proposal_source` names which one produced the reported number and `pareto_k_first_pass` is the k-hat of the proposal exactly as the mode-find placed it, before refinement. A large gap between the two says the placement is poor even where the verdict is fine – on a small-group binary fit the first pass runs 15 to 49 where the reported k-hat is 0.3 to 0.8.
- `k_samples`: importance draws for the `diagnose_k` estimate (default 500). It is a precision knob: the GPD tail size is held at the fraction that default budget implies, so raising it supplies more tail ratios for the SAME estimand rather than moving the fit to a deeper quantile of the weight distribution (gcol33/tulpa#631). Each draw is one inner Laplace solve, so this sets the diagnostic's cost; the draws are solved in one compiled batch, each started from the mode at the proposal centre.
- `k_threads`: thread width for those draws (`NULL`, the default, follows `n_threads`; `"auto"` takes the physical performance cores; an integer sets it, `1L` forcing serial). Each draw is solved single-threaded and from the same start, so the k-hat does not depend on the width.

- k_tail_points: expert override for that tail size, in upper-tail order statistics. Silently capped at 20% of the draws, beyond which body ratios enter the tail and bias the shape.
- max_iter, tol, n_threads: inner-solve controls (see tulpa_laplace()).
- outer_maxit: iteration budget for the mode-finding step that centres the integration grid (default 500). Applies to the Nelder-Mead simplex used from two parameters up; the one-parameter case is bracketed by Brent. Exhausting the budget warns, since the nodes are then centred on wherever the optimizer stopped.
- checkpoint: node checkpoint/resume spec list(path = , resume =). Each completed CCD / grid node (one inner Laplace solve) is cached to path; a resume = TRUE run loads the finished nodes and re-solves only the rest. resume = FALSE starts fresh. A file written for different data, layout, or grid is rejected (fingerprint mismatch). Default NULL (off).
- subspace_debias: subspace debias, FALSE by default. TRUE takes every default; a list overrides band (the inner-reliability floor a coordinate is selected at, default "ok"), idx (pin the corrected set explicitly, skipping the selector), probe (the latent indices scored, default the fixed effects), closure (FALSE, TRUE, or a partial-correlation threshold: grow the set by the precision-graph neighbours it is strongly coupled to), closure_max, and the sampler budget n_iter / warmup / thin. When the selected set is non-empty, each integration node reports the selected fixed-effect coordinates from a Metropolis sample of the exact conditional along the Gaussian-conditional-mean surface, and the rest from the Gaussian conditional given them; an EMPTY set leaves the fit bit-for-bit identical to the plain path. What was selected is recorded in subspace_debias on the returned fit.

Details

Each term is one covariance **block**. A correlated block ((1 + x | g)) is a full Sigma = L L' parameterized by its lower Cholesky factor in log-Cholesky coordinates (the log-diagonal and the strictly-lower entries of L, c(c+1)/2 values for a c-coefficient block), which keeps Sigma positive definite for every coordinate. An uncorrelated block ((1 + x || g)) is a diagonal Sigma parameterized by its c log standard deviations. A scalar (1 | g) term is the degenerate c = 1 block. Several blocks stack their parameters into one integration vector; a single-term model is the length-1 case.

Integration nodes live in the whitened stacked-parameter space, centred at the joint marginal-likelihood mode and rotated/scaled by the Cholesky of the mode's posterior covariance (solve(Hessian)), so points track the posterior ridge. Two node layouts are available via integration:

- "ccd" (default): a central-composite design (ccd_grid()) of 1 + 2k + 2^(k-q) points for the total k = sum_blocks parameters, with the corrected R-INLA design weights (ccd_weights()). Scales polynomially in k, where the tensor grid is exponential.
- "grid": the full n_per_axis^k tensor product with uniform cell weights – denser and more robust to a non-Gaussian whitened posterior, but only tractable for small k.

Each node k contributes integration weight proportional to Delta_k * exp(log_marginal(Sigma_k) + log_prior_theta(theta_k)), following the INLA convention int ~ sum_k Delta_k pi(theta_k).

The two layouts also decide how the reported median and 2.5\ of each derived quantity are read off the nodes. A tensor grid's uniform cells discretize the posterior density, so the cumulative node weights are a CDF and the summary is the weighted quantile. A CCD is a moment rule: its nodes sit where they reproduce the integrand's first two moments and carry no probability mass of their own, so the summary is moment-matched instead – the first two weighted moments on each quantity's own coordinate (`log` for a scale or a variance, `atanh` for a correlation, the identity for a covariance) define a Gaussian there whose quantiles are mapped back. Scale intervals are therefore positive and asymmetric, and correlation intervals stay inside `(-1, 1)`. The `mean` and `sd` columns are the weighted moments under either layout.

By default (`hyperprior = "proper"`) `log_prior_theta` is the weakly-informative PC + LKJ hyperprior, built per block by `re_cov_pc_lkj_prior()` and summed over blocks (PC prior on each marginal SD via `prior_sigma`, LKJ prior on each correlated block's correlation matrix via `eta`), expressed in the same parameterization with the exact change-of-variables Jacobian. `hyperprior = "flat"` makes it the zero function, flat in log(theta). The likelihood it multiplies is the inner Laplace marginal, which integrates the fixed effects out together with the random effects rather than profiling them, so under `"flat"` the node centre `theta_hat` is a restricted (REML-type) estimate – REML for a gaussian response, agreeing with `glmmTMB(..., REML = TRUE)` rather than with the ML fit of `lme4::glmer()` otherwise (see `tulpa_eb()`). Supply a custom `log_prior_theta` function to override either default (then `prior_sigma` / `eta` / `hyperprior` are ignored); it must act on the full stacked parameter vector. `tulpa_eb()` shares this same objective and the same default, so `tulpa_eb()$theta_hat` and `tulpa_re_cov_nested()$theta_hat` stay the same estimate on the same data under either setting.

Value

A list with:

- `posterior`: data frame with one row per parameter and columns `mean`, `sd`, `median`, `ci_lo`, `ci_hi`. Parameter names are `sigma_i`, `rho_ij`, `Sigma_ij` for a single block, prefixed by the block label (`g.sigma_1`, ...) when there are several blocks. Diagonal blocks report no `rho`.
- `map`: the plug-in-mode summary at `theta_hat` (a single `list(Sigma, sigma, rho)` for one block, or a named list of them).
- `Sigma_mean`: the weighted posterior mean of `Sigma` (a matrix for one block, or a named list of matrices).
- `beta`, `draws`, `means`, `param_names`, `process_info`: the fixed-effect posterior from the node mixture (drives `coef`/`confint`/`vcov`/`summary`).
- `re_nodes`, `re_var_nodes`: the per-group random-effect posterior at each integration node – conditional mean and marginal variance of every (block, group, coefficient), one row per node. `ranef()` reports the `weights`-mixture of them. `NULL` at `n_quad > 1`, whose inner marginal integrates each group out instead of conditioning on it; that fit carries `ranef_unavailable` (the reason) in their place.
- `re_debias_draws`, `re_debias_idx`: present when the subspace debias selected a random-effect coordinate. The sampled draws of those coordinates on the node mixture the fixed-effect draws use, and their positions within the random-effect block. `ranef()` reports those rows empirically and the rest from the Gaussian mixture, recording which in its `source` column.
- `theta_hat`, `theta_grid`, `weights`, `log_marginal`, `n_grid`, `layout`, `n_blocks`, `n_coefs` (vector of per-block `c`).

- subspace_debias: present only when control$subspace_debias was set. idx are the corrected latent coordinates, bands the per-probed-index reliability table they were read from, closure_added what the coupling closure added, and accept the per-node Metropolis acceptance rate.

References

Rue, Martino & Chopin (2009). Approximate Bayesian inference for latent Gaussian models by using integrated nested Laplace approximations. *JRSS-B* 71(2):319-392. Lewandowski, Kurowicka & Joe (2009). Generating random correlation matrices based on vines and extended onion method. *Journal of Multivariate Analysis* 100(9):1989-2001.

See Also

tulpa_laplace() for the inner solve; tulpa_nested_laplace() for the analogous outer integration over spatial / temporal prior hyperparameters.

Examples

```
set.seed(1)
G <- 20L; per <- 12L; n <- G * per
grp <- rep(seq_len(G), each = per); x <- rnorm(n)
b <- cbind(rnorm(G, 0, 0.7), rnorm(G, 0, 0.5))     # random intercept + slope
eta <- -0.2 + 0.5 * x + b[grp, 1] + b[grp, 2] * x
y <- rbinom(n, 1L, plogis(eta))
re_term <- list(idx = grp, n_groups = G, n_coefs = 2L, Z = cbind(1, x),
                correlated = TRUE)
fit <- tulpa_re_cov_nested(y, rep(1L, n), cbind(1, x), re_term,
                           family = "binomial")
fit$Sigma_mean        # marginalized RE covariance
```

---

tulpa_rpg *Draw Polya-Gamma random variates*

Description

Vectorized PG(b, z) draws via the Polson, Scott & Windle (2013) sampler (tulpa::rpg_vec()), the kernel every Polya-Gamma Gibbs fitter in this package draws its auxiliary weights from. Exposed as a door for consumer packages fitting their own Polya-Gamma Gibbs models, which cannot reach an RNG through LinkingTo the way a likelihood kernel would.

Usage

```
tulpa_rpg(b, z)
```

Arguments

b	Integer vector of PG shape parameters (trial counts); one draw per element.
z	Numeric vector of tilting parameters, the same length as b.

Value

A numeric vector of `PG(b, z)` draws, the same length as `b`.

Examples

```
tulpa_rpg(rep(1L, 5), rnorm(5))
```

---

tulpa_simulate *Simulate data from a tulpa model*

Description

Generic simulator that dispatches through a `tulpa_family`'s `simulate_fn`. Given a model spec (formula + family + data) and a parameter vector or a fitted model, generate one or more synthetic response datasets.

Used internally by `prior_predict()` and exposed for posterior predictive checks, simulation-based calibration, and what-if analyses with fixed parameters.

Usage

```
tulpa_simulate(
  formula,
  family,
  data,
  theta = NULL,
  n_sims = 1L,
  priors = NULL,
  seed = NULL,
  ...
)

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

Arguments

`formula`	A model formula, or list of formulas keyed by process name.
`family`	A `tulpa_family` object (see `tulpa_family()`).
`data`	Data frame with covariates and grouping factors. The response column may be absent or NA; it is not read.
`theta`	One of: • A named list with `beta` (numeric or list per process), `u` (list of RE blocks, one per RE term per process, each an `n_groups x n_coefs` matrix; a vector of length `n_groups` is accepted for a single-coefficient term), `extras` (named list of family-specific extras like `phi`, `sigma_y`).

- A `tulpa_fit` object: each simulation takes one posterior draw of the fixed effects and, paired with it, of the FITTED random effects (read through the same accessor as `ranef()` and `posterior_predict()`). The data are therefore simulated conditional on the fitted groups: every level of a grouping variable in `data` must be one the fit estimated (matched by level), and a fit that carries no random-effect posterior is an error rather than a set of zero effects. To simulate NEW groups, pass `theta` as a list with `u` drawn for them.
- `NULL`: equivalent to a single prior draw (shortcut).

`n_sims` Number of simulated datasets. When `theta` is a fit, draws are subsampled (or recycled) to `n_sims`. Default 1.

`priors` Used only if `theta = NULL`; default `tulpa_priors()`.

`seed` Optional integer seed.

`...` Passed to `family$simulate_fn`.

`x` A `tulpa_simulate` object.

Value

A `tulpa_simulate` object: list with `y` (length-`n_sims` list of simulated responses), `theta` (parameters used per sim), `linpred`, `family`, `n_sims`, `n_obs`.

Examples

```
fam <- tulpa_family(
  name = "gaussian",
  simulate_fn = function(eta, params, n_obs, ...) {
    rnorm(n_obs, eta[[1]], params$sigma_y)
  },
  extra_params = list(sigma_y = prior_half_normal(1))
)
df <- data.frame(y = rep(0, 20), x = rnorm(20))
theta <- list(
  beta = list(y = c(0.5, 1.0)),
  u = list(y = list()),
  extras = list(sigma_y = 0.5)
)
sim <- tulpa_simulate(y ~ x, fam, df, theta = theta, n_sims = 3, seed = 1)
length(sim$y)  # 3
```

`tulpa_spatial` *Spatial structure specifications for tulpa*

Description

Functions to specify spatial random effects for tulpa models. Spatial effects are shared between processes by default, which helps prevent bias from spatially-structured unmeasured confounders.

Value

The spatial constructors documented in this family (spatial_car(), spatial_bym2(), spatial_gp(), and the others) each return a tulpa_spatial (or related) specification object to pass to the spatial argument of tulpa().

---

tulpa_spde_log_hyperprior

SPDE field hyperprior log density on (range, sigma)

Description

The SPDE field's hyperprior on (range, sigma), evaluated as a density on (log range, log sigma) – the coordinates every SPDE outer integrator works in (grid, CCD and k-hat alike, gcol33/tulpa#731). Intended for a caller running its own outer-grid loop over (range, sigma) around a coupled model tulpa's own SPDE front doors do not fit directly.

Usage

```
tulpa_spde_log_hyperprior(range, sigma, sp, hyperprior = "proper")
```

Arguments

range, sigma	Numeric vectors of range / marginal-SD nodes (natural scale, recycled against each other).
sp	A spatial_spde() / spatial_spde_custom() spec, carrying the declared or defaulted prior_range / prior_sigma anchors.
hyperprior	"proper" (default) or "flat"; see ?spatial_spde.

Value

Numeric vector of log-density values, one per (range, sigma) pair.

See Also

tulpa_spde_precision_Q(), spatial_spde()

Examples

```
n <- 6L
G <- Matrix::bandSparse(n, k = c(-1, 0, 1),
                        diagonals = list(rep(-1, n - 1), c(1, rep(2, n - 2), 1),
                                         rep(-1, n - 1)))
sp <- spatial_spde_custom(Matrix::Diagonal(n), G, Matrix::Diagonal(n),
                          prior_range = c(1, 0.5), prior_sigma = c(1, 0.01))
tulpa_spde_log_hyperprior(range = c(0.5, 1, 2), sigma = 1, sp = sp)
```

tulpa_spde_precision_Q

Rebuild the SPDE/Matern field precision matrix

Description

Rebuilds the sparse precision matrix `Q(kappa, tau)` of an SPDE/Matern field from its mesh geometry and a `(kappa, tau)` hyperparameter pair, mirroring the compiled builder (`src/spde_qbuilder.h`) that `tulpa_nuts_spde()` and `fit_spde()` use internally. Intended for a caller running its own outer-grid loop over `(range, sigma)` around a coupled model tulpa's own SPDE front doors (`fit_spde()`, `spatial_spde()`) do not fit directly – e.g. a per-species community layer sharing one spatial field.

Usage

```
tulpa_spde_precision_Q(spatial, kappa, tau_spde)
```

Arguments

`spatial`	A `spatial_spde()` / `spatial_spde_custom()` spec, carrying `n_mesh`, `C0_diag`, `G` and `nu`.
`kappa`	Spatial frequency `kappa = sqrt(8*nu) / range`.
`tau_spde`	SPDE precision scale (see `tulpa_spde_log_hyperprior()`'s `(range, sigma) -> (kappa, tau)` mapping via `.spde_kappa_tau()`).

Value

A sparse `n_mesh x n_mesh` precision matrix.

See Also

`tulpa_spde_log_hyperprior()`, `fit_spde()`, `spatial_spde()`

Examples

```
n <- 6L
G <- Matrix::bandSparse(n, k = c(-1, 0, 1),
                        diagonals = list(rep(-1, n - 1), c(1, rep(2, n - 2), 1),
                                         rep(-1, n - 1)))
sp <- spatial_spde_custom(Matrix::Diagonal(n), G, Matrix::Diagonal(n),
                          prior_range = c(1, 0.5), prior_sigma = c(1, 0.01))
Q <- tulpa_spde_precision_Q(sp, kappa = sqrt(8) / 1, tau_spde = 1)
dim(Q)
```

tulpa_temporal *Temporal structure specifications for tulpa*

Description

Functions to specify temporal random effects for tulpa models. Temporal effects are shared between processes by default, which helps prevent bias from temporally-structured unmeasured confounders.

Value

The temporal constructors documented in this family (`temporal_rw1()`, `temporal_rw2()`, `temporal_ar1()`, and the others) each return a `tulpa_temporal` specification object to pass to the `temporal` argument of `tulpa()`.

tulpa_tgmrf *Fit a custom tgmrf latent block*

Description

One front door for inference over a user-defined `tgmrf()` latent block's hyperparameter vector `theta`. `mode` selects the inference engine, all of which share the same Laplace body for `(beta, z) | theta`:

- `"imh"` (default) – Tier-1 exact MCMC: the Laplace body plus an independence-Metropolis bias correction over `theta` (the "Laplace + MH debias" composition).
- `"nuts"` – Tier-1 exact MCMC: NUTS over the marginal `theta` posterior.
- `"vi"` – Tier-2 structured: a single-path Pathfinder Gaussian fit of the `theta` posterior (no bias correction).
- `"nuts_joint"` – Tier-1 exact MCMC sampling the FULL joint `(beta, z, theta)` rather than the Laplace-marginalized `theta` (requires a C++-backend block, see `tgmrf_cpp()`).

The inference method is an argument, not a parallel verb.

Usage

```
tulpa_tgmrf(
  y,
  n_trials,
  X,
  block,
  family = "binomial",
  phi = 1,
  re_idx = NULL,
  n_re_groups = 0L,
  sigma_re = 1,
  mode = c("imh", "nuts", "vi", "nuts_joint"),
  control = list(),
  ...
)
```

Arguments

y, n_trials, X	Response, binomial trial counts (or NULL), and the fixed-effect design matrix.
block	A tgmrf() latent block.
family, phi	Observation family and its dispersion.
re_idx, n_re_groups, sigma_re	Optional scalar random-intercept structure.
mode	Inference engine: "imh", "nuts", "vi", or "nuts_joint".
control	A list of numerical / tuning knobs for the chosen mode (e.g. n_iter, warmup, thin, scale for "imh"; epsilon, max_depth, target_accept for the NUTS modes; n_draws, max_lbfgs, lbfgs_tol for "vi"; plus the shared pilot_axis_points, max_iter, tol, n_threads, verbose). An unknown knob for the chosen mode errors rather than being silently ignored.
...	Individual control knobs may also be passed directly by name; they are merged into control (a named argument here wins over the same name inside control).

Value

A tulpa_fit object; $backend and $mode record the engine used.

See Also

tgmrf() for the block, tgmrf_cpp() for the compiled-block form.

Examples

```
# An iid random intercept written as a tgmrf block over 15 groups; the
# inference method is an argument, not a parallel verb. See the
# custom-process article at https://gillescolling.com/tulpa/articles/tgmrf.html.
set.seed(3)
J <- 15L; g <- rep(seq_len(J), each = 6L); n <- length(g); x <- rnorm(n)
y <- rpois(n, exp(0.3 + 0.5 * x + rnorm(J, 0, 0.6)[g]))
blk <- tgmrf(
  Q = function(theta) Matrix::Diagonal(J, exp(-2 * theta[1])),
  prior = function(theta) stats::dnorm(theta[1], 0, 1, log = TRUE),
  init = c(log_sigma = 0), obs_idx = g)
fit <- tulpa_tgmrf(y, rep(1L, n), cbind(1, x), block = blk,
                   family = "poisson", mode = "imh",
                   control = list(n_iter = 300L, warmup = 150L))
mean(exp(fit$draws[, "log_sigma"]))   # posterior mean group SD
```

tulpa_theta_matrix *A fit's outer grid as a named matrix*

Description

Coerces a nested-Laplace fit's `theta_grid` to matrix form with named axis columns. A single-axis grid is stored on the fit as a bare vector named by `theta_names`; every downstream reader that keys an axis by name needs the named matrix form, so this is the one place that coercion happens. Intended for reconstructing a fit's outer-grid posterior cell weights (see `tulpa_grid_log_quad()`, `tulpa_normalise_weights_safe()`) for a fit or path whose driver doesn't already carry `fit$log_quad`.

Usage

```
tulpa_theta_matrix(res)
```

Arguments

`res`	A `tulpa_fit` object (or any list carrying `theta_grid` and `theta_names`).

Value

`res$theta_grid` as an `[n_cells x n_axes]` matrix with axis names as column names, or `NULL` if the fit has no grid.

See Also

`tulpa_grid_log_quad()`, `tulpa_normalise_weights_safe()`

Examples

```
set.seed(1)
S <- 30L                                  # spatial units in a chain
nb <- lapply(seq_len(S), function(s) setdiff(c(s - 1L, s + 1L), c(0L, S + 1L)))
nn <- lengths(nb)
field <- as.numeric(scale(cumsum(rnorm(S, 0, 0.4))))
idx <- rep(seq_len(S), each = 6L); n <- length(idx); x <- rnorm(n)
y <- rbinom(n, 1L, plogis(-0.2 + 0.6 * x + field[idx]))
prior <- list(type = "icar", n_spatial_units = S, spatial_idx = idx,
              adj_row_ptr = c(0L, cumsum(nn)), adj_col_idx = unlist(nb) - 1L,
              n_neighbors = nn, tau_grid = c(0.5, 1, 2, 4, 8))
fit <- tulpa_nested_laplace(y, rep(1L, n), cbind(1, x), prior = prior,
                            family = "binomial",
                            control = list(progress = FALSE))
tulpa_theta_matrix(fit)
```

tulpa_validate *Posterior predictive checks for tulpa models*

Description

Visual and numerical checks comparing observed data to posterior predictive distributions. Essential for assessing model fit.

Value

The functions documented in this family return a ggplot object (pp_check()) or a tulpa_prior_predict object (prior_predict()); see each function's own help page.

tulpa_variogram *Empirical semivariogram of residuals*

Description

Computes the empirical semivariogram in distance bins for visual assessment of remaining spatial structure in residuals.

Usage

```
tulpa_variogram(
  object,
  coords,
  n_bins = 15L,
  max_dist = NULL,
  resid_type = "pearson"
)

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

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

Arguments

object	A fitted model, or a numeric vector of residuals
coords	N x 2 coordinate matrix (required)
n_bins	Number of distance bins (default 15)
max_dist	Maximum distance (default: half the maximum pairwise distance)
resid_type	Residual type if extracting from model (default "pearson")
x	A tulpa_variogram table.
...	Passed to graphics::plot() by plot() and to print() by print().

Value

A `tulpa_variogram` data.frame with columns `dist`, `gamma`, `n_pairs`

`plot()` draws the semivariance against distance, point size scaled by the pair count, and returns `x` invisibly.

Examples

```
set.seed(1)
coords <- cbind(runif(60), runif(60))
tulpa_variogram(rnorm(60), coords, n_bins = 5)
```

---

tvc — *Extract temporally-varying coefficients from a fitted model*

Description

Extract posterior distributions of temporally-varying coefficients (TVCs) from a fitted tulpa model with TVC specification.

Usage

```
tvc(object, terms = NULL, summary = FALSE, probs = c(0.025, 0.5, 0.975), ...)

## S3 method for class 'tulpa_fit'
tvc(object, terms = NULL, summary = FALSE, probs = c(0.025, 0.5, 0.975), ...)

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

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

## S3 method for class 'tulpa_tvc_posterior'
summary(object, probs = c(0.025, 0.5, 0.975), ...)
```

Arguments

`object`	A `tulpa_fit` object fitted with `tvc` argument
`terms`	Which TVC terms to extract. If NULL (default), extracts all.
`summary`	Logical; if TRUE, return summary statistics instead of full posterior draws.
`probs`	Quantiles to compute if `summary = TRUE`.
`...`	Ignored
`x`	A `tulpa_tvc_posterior` or `tulpa_tvc_summary` object.

Value

A `tulpa_tvc_posterior` object containing:

- `draws`: Array of posterior draws (draws x times x terms)
- `time_levels`: Time point labels
- `term_names`: Names of TVC terms

`summary()` on a `tulpa_tvc_posterior` returns a `tulpa_tvc_summary` data frame with one row per time point and term, holding the posterior mean, SD and requested quantiles of each temporally varying coefficient.

See Also

`temporal_tvc()`, `plot.tulpa_tvc_posterior()`

Examples

```
set.seed(160)
n_t <- 10L; reps <- 5L
walk <- cumsum(rnorm(n_t, 0, 0.35)); walk <- walk - mean(walk)
year <- rep(seq_len(n_t), each = reps)
df <- data.frame(year = year, x = rnorm(length(year)))
df$count <- rpois(nrow(df), exp(0.3 + (0.5 + walk[year]) * df$x))

# The slope on `x` walks in time; TVC is exact-mode only.
fit <- tulpa(
  count ~ x,
  data = df,
  family = "poisson",
  temporal = temporal_tvc("year", terms = ~ x - 1, structure = "rw1"),
  mode = "exact",
  control = list(n_iter = 200L, n_warmup = 100L, seed = 1L)
)

tvc_post <- tvc(fit)
summary(tvc_post)
plot(tvc_post, "x")
```

---

validate_gp — *Validate a GP spatial specification against data*

Description

Fills in the neighbor structure (`spatial_gp()`) or structures (`spatial_multiscale()`) that `tulpa_gibbs()` and `tulpa_laplace()` need from a `spatial =` spec, since those design-matrix doors take no `data` argument to validate one themselves. `tulpa()` calls this internally, so it only needs to be called directly when fitting through a design-matrix door.

Usage

```
validate_gp(gp, data)
```

Arguments

gp	A spatial_gp() or spatial_multiscale() object.
data	Data frame containing the coordinate columns named in gp.

Value

Updated spatial object with computed neighbor structure.

See Also

validate_temporal_multiscale(), spatial_gp(), spatial_multiscale()

Examples

```
set.seed(1)
d <- data.frame(lon = runif(30), lat = runif(30))
gp <- validate_gp(spatial_gp(~ lon + lat, nn = 5), d)
gp$n_spatial
```

---

validate_mode *Validate that a fit used the expected mode*

Description

Ensures the fit object was created with the expected inference mode. Useful for enforcing mode requirements in downstream analysis.

A fit matches a mode when the mode names either the backend that ran (fit$backend, e.g. "laplace", "mala", "re_cov_gibbs") or the tier it ran in (fit$inference_mode: "exact", "structured", "optimized"). So a fit made with mode = "laplace" validates against both "laplace" and "structured", and a fit made with mode = "auto" validates against the backend and the tier auto chose. A request that tulpa() overrode (see fit$mode_overridden) validates against what ran, not what was asked for.

Usage

```
validate_mode(fit, expected_mode, error = TRUE)
```

Arguments

fit	A tulpa_fit object
expected_mode	Mode(s) that should have been used: a character vector of backend and / or tier names, any one of which is accepted.
error	If TRUE (default), error on mismatch. If FALSE, return logical.

Value

If error = FALSE, returns TRUE/FALSE. Otherwise errors on mismatch.

Examples

```
set.seed(1)
d <- data.frame(x = rnorm(60))
d$y <- rpois(60, exp(0.5 + 0.3 * d$x))
fit <- tulpa(y ~ x, data = d, family = "poisson", mode = "hmc",
             control = list(n_iter = 400L, warmup = 200L, n_chains = 2L,
                            seed = 1L))
validate_mode(fit, "exact")                      # the tier hmc runs in
validate_mode(fit, "structured", error = FALSE)  # FALSE
```

---

validate_temporal_multiscale

Validate a multi-scale temporal specification against data

Description

Fills in the time index and point count that `tulpa_gibbs()` needs from a `temporal =` spec built with `temporal_multiscale()`, since that design-matrix door takes no `data` argument to validate one itself. `tulpa()` calls this internally, so it only needs to be called directly when fitting through a design-matrix door.

Usage

```
validate_temporal_multiscale(temporal, data)
```

Arguments

`temporal`	A `temporal_multiscale()` object (or a single-component temporal spec, dispatched to `validate_temporal()`).
`data`	Data frame containing the time column named in `temporal`.

Value

Updated object with indices computed.

See Also

`validate_gp()`, `temporal_multiscale()`

Examples

```
tm <- temporal_multiscale("t", trend = "rw1", seasonal = 12)
tm <- validate_temporal_multiscale(tm, data.frame(t = 1:36))
class(tm)
```

VarCorr *Random-effect variances and correlations*

Description

Summarize the random-effect covariance of a fit: the standard deviation of each random-effect coefficient, the correlations between them within a term, and the covariance matrix itself.

Whether the covariance was estimated, sampled, or merely conditioned on is reported alongside it, because the three are different claims. A fit from `mode = "laplace"` conditions on `sigma_re`, so its values are the ones supplied; `mode = "eb"` estimates them; a sampler tier integrates them.

Usage

```
VarCorr(x, sigma = 1, ...)

## S3 method for class 'tulpa_fit'
VarCorr(x, sigma = 1, ...)
```

Arguments

`x`	A `tulpa_fit`.
`sigma`	Ignored, for compatibility with the `VarCorr` generic.
`...`	Ignored.

Value

A data frame with one row per random-effect coefficient: `term`, `coef`, `sd`, and `source` (one of `"estimated"`, `"sampled"`, `"conditioned"`). A gaussian fit adds a final `term = "Residual"` row (`coef NA`) with the residual SD `sqrt(phi)`, labelled `"estimated"` under `estimate_phi = TRUE` and `"conditioned"` otherwise; it has no entry in `"cov"`. Correlated terms additionally carry the covariance matrices in the `"cov"` attribute, one per term, each with a `"correlation"` attribute. Returns an empty data frame when the fit has no random effects.

See Also

`ranef()` for the per-level deviations, `tulpa_eb()` to estimate the covariance rather than condition on it.

Examples

```
set.seed(1)
G <- 30L; per <- 10L; n <- G * per
grp <- rep(seq_len(G), each = per); x <- rnorm(n)
b <- rnorm(G, 0, 0.8)
d <- data.frame(y = rpois(n, exp(0.3 + 0.5 * x + b[grp])), x = x,
                g = factor(grp))
fit <- tulpa(y ~ x + (1 | g), data = d, family = "poisson", mode = "eb")
```

```
VarCorr(fit)
```

`vcov.tulpa_fit` *Variance-covariance matrix of the fixed effects*

Description

Variance-covariance matrix of the fixed effects

Usage

```
## S3 method for class 'tulpa_fit'
vcov(object, ...)
```

Arguments

`object`	A `tulpa_fit` object.
`...`	Ignored.

Value

Fixed-effect variance-covariance matrix (empirical for sampler tiers, `H_beta^-1` for the Laplace tier, the stated covariance on a fit whose reported posterior is a closed-form Gaussian).

`wbic` *Widely applicable Bayesian information criterion (WBIC)*

Description

Watanabe's (2013) WBIC: the expected negative log-likelihood under the posterior tempered at inverse temperature `beta = 1 / log(n)`,

$$\mathrm{WBIC} = -E_\beta\left[\sum_{i=1}^{n}\log p(y_i \mid \theta)\right], \qquad p_\beta(\theta) \propto p(\theta)\prod_{i=1}^{n} p(y_i \mid \theta)^\beta.$$

It estimates the Bayes free energy `-log p(y)`, the negative log evidence, and shares its asymptotic expansion up to `O_p(1)` terms in singular models as well as regular ones – mixtures, reduced-rank and latent-factor structure, a variance component at zero – where BIC's `(d / 2) log n` penalty does not describe the evidence. Lower is better.

Usage

```
wbic(object, ...)

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

## S3 method for class 'tulpa_fit'
wbic(object, ...)

## S3 method for class 'tulpa_wbic'
print(x, digits = 2, ...)
```

Arguments

`object`	A `tulpa_fit` fitted with `mode = "smc"` and `control = list(wbic = TRUE)`, or an `[n_draws x n_obs]` pointwise log-likelihood matrix (or `tulpa_loglik()`) at draws from the posterior tempered at `1 / log(n_obs)`.
`...`	Unused.
`x`	A `tulpa_wbic` result.
`digits`	Decimal places printed.

Details

No draw from the posterior itself can supply the expectation: it is taken under a different distribution, and importance-reweighting posterior draws down to `beta` degenerates as `n` grows. A `tulpa_fit` therefore carries the tempered draws only when its sampler visited that distribution, which the SMC path does when asked: fit with `mode = "smc"` and `control = list(wbic = TRUE)`. The population is then carried from the reference through `p(theta) L(theta)^beta`, recorded there, and tempered on to the posterior, so the fit's own `$draws` are posterior draws as usual. A fit without the tempered draws is refused.

The log-likelihood is the normalized pointwise density `pointwise_loglik()` returns, the input of `loo::waic()` and `loo::loo()` too, so WBIC is on the scale of `-log p(y)` and comparable across fits of the same response. The parameter vector tempered is the one the sampler samples – fixed effects, latent effects and hyperparameters – with the likelihood conditional on it, so the free energy estimated is that of the marginal likelihood integrating all of them.

The default method takes an `[n_draws x n_obs]` pointwise log-likelihood whose rows are draws from the posterior tempered at `1 / log(n_obs)`. A model package with its own likelihood draws them through the `tulpa_smc_fit` C entry with `bridge_end = 1 / log(n)` and evaluates its density there.

Value

A `tulpa_wbic` object: a list with `wbic`, the inverse temperature `beta`, `n_obs` and `n_draws`.

References

Watanabe, S. (2013). A widely applicable Bayesian information criterion. *Journal of Machine Learning Research* 14:867-897.

See Also

loo::waic(), dic() and compare_models() for the other criteria; tulpa_criteria().

Examples

```
set.seed(1)
d <- data.frame(x = rnorm(80))
d$y <- rpois(80, exp(0.3 + 0.5 * d$x))
fit <- tulpa(y ~ x, data = d, family = "poisson", mode = "smc",
             control = list(wbic = TRUE, n_particles = 500L, seed = 1L))
wbic(fit)
```

---

with_tulpa_integrator *Run an expression under a chosen symplectic integrator*

Description

Evaluate expr with the integrator set to name, then restore whatever was selected before – on an error as well as on success. The integrator selection is process-global, so a bare tulpa_integrator() call leaves every later fit in the session on the new scheme, and an error between setting and restoring leaves it there permanently.

Usage

```
with_tulpa_integrator(name, expr, mts_substeps = 4L)
```

Arguments

name	Integrator name, as for tulpa_integrator().
expr	Expression to evaluate. Evaluated in the caller's environment.
mts_substeps	Inner prior-force substeps for "mts" (default 4).

Value

The value of expr.

See Also

tulpa_integrator()

Examples

```
with_tulpa_integrator("yoshida4", tulpa_integrator())
tulpa_integrator()   # unchanged
```

Index

∗ **models**
spatiotemporal, 165

adjacency, 6
adjacency(), *21, 22*, *65*
agq_fit, 9
agq_fit(), *12*, *59*, *274*
AIC.tulpa_fit, 12
AIC.tulpa_fit(), *59*
as_draws, 13
as_draws_array *(*as_draws*)*, 13
as_draws_df *(*as_draws*)*, 13
as_draws_matrix *(*as_draws*)*, 13
as_draws_rvars *(*as_draws*)*, 13
auto_grid, 14
auto_grid(), *16*, *46*, *53, 54*, *94*, *103*, *105, 106*, *109*, *119*
auto_grid_place, 16
auto_grid_place(), *15*, *53, 54*

bayes_R2, 17
BIC.tulpa_fit *(*AIC.tulpa_fit*)*, 12
BIC.tulpa_fit(), *59*
bridge_sampling, 18
bridge_sampling(), *52*, *68*

categorical_accessors, 19
ccd_grid(), *284*
ccd_weights(), *284*
check_adjacency, 21
check_adjacency(), *9*
check_diagnostics, 22
check_diagnostics(), *34*, *36*, *49*, *66*, *76*
check_model, 23
coef(), *48*, *89, 90*, *220*, *253*, *259*
coef.summary.tulpa_fit *(*print.tulpa_fit*)*, 89
coef.tulpa_fit, 24
coef.tulpa_fit(), *48*, *92*
coef.tulpa_ordinal *(*tulpa_ordinal*)*, 258
compare_models, 24
compare_models(), *27, 28*, *63*, *207*, *302*
confint(), *90*, *220*, *253*, *259*
confint.tulpa_fit, 25
confint.tulpa_fit(), *91, 92*, *109*
cpo *(*criteria_doors*)*, 26
cpo(), *20*
criteria_doors, 26

diagnostic_summary, 35
diagnostic_summary(), *22, 23*, *34*, *76, 77*, *82*, *91*
diagnostics, 28
diagnostics(), *23*, *36*, *49*, *61, 62*, *74*, *77*, *81, 82*, *85*, *91, 92*, *95*, *115*, *124*, *143*, *208*, *272*
dic *(*criteria_doors*)*, 26
dic(), *20*, *302*
durbin_watson, 36

energy_bfmi, 37

family_names, 38
family_names(), *45*, *127*, *194*, *220*
field_effects, 38
field_effects(), *136*, *146*
fit_spde, 40
fit_spde(), *39*, *45*, *95*, *107*, *130*, *195*, *197*, *256*, *258*, *290*
fit_st_nested, 44
fit_st_nested(), *15*, *98*, *165*, *169*
fitted(), *47*, *63*, *89*, *136*
fitted.tulpa_categorical *(*categorical_accessors*)*, 19
fitted.tulpa_fit, 40
fixef, 48
format.tulpa_prior *(*tulpa_priors*)*, 268
format.tulpa_prior_beta *(*tulpa_priors*)*, 268

format.tulpa_prior_exponential (tulpa_priors), 268
format.tulpa_prior_gamma (tulpa_priors), 268
format.tulpa_prior_half_cauchy (tulpa_priors), 268
format.tulpa_prior_half_normal (tulpa_priors), 268
format.tulpa_prior_normal (tulpa_priors), 268
format.tulpa_prior_pc (tulpa_priors), 268

geweke_test, 49
glance.tulpa_fit, 50
graphics::plot(), *294*

hyper_axis_spec(), *232*, *234*

imh_laplace, 51
imh_laplace(), *67, 68*
inference_mode_info, 53
inference_mode_info(), *193*, *199*
is_auto_grid, 53
is_auto_grid(), *15, 16*

latent, 54
latent(), *190*, *192*
latent_factor, 55
latent_factor(), *251*
latent_factors, 57
logLik(), *25*
logLik.tulpa_fit, 58
logLik.tulpa_fit(), *12*
loo.tulpa_fit (criteria_doors), 26
loo::loo(), *301*
loo::loo_compare(), *28*
loo::waic(), *301, 302*

mala, 59
mala(), *199*
mcmc_draws, 61
mcmc_draws(), *85*
model.matrix(), *201*
model_average, 62
model_average(), *25*, *27*
moran_i, 63
moran_i(), *37*

n_divergent, 66
n_divergent(), *23*, *75*
nobs.tulpa_fit, 64
node_index, 65
node_index(), *7–9*, *22*

pathfinder, 67
pathfinder(), *199*, *221*
pit_residuals, 69
pit_residuals(), *143*
plot.sbc (sbc), 139
plot.tulpa_fit, 70
plot.tulpa_prior_predict, 70
plot.tulpa_st_summary, 71
plot.tulpa_svc_posterior, 72
plot.tulpa_svc_posterior(), *171*
plot.tulpa_temporal_posterior, 72
plot.tulpa_tvc_posterior, 73
plot.tulpa_tvc_posterior(), *296*
plot.tulpa_variogram (tulpa_variogram), 294
plot_acf, 73
plot_diagnostics(), *36*
plot_divergences, 74
plot_divergences(), *66*, *81*
plot_energy, 75
plot_ess, 76
plot_ess(), *34*, *74*, *82*
plot_map, 77
plot_map_panel, 79
plot_pairs, 81
plot_pairs(), *75*
plot_rhat, 82
plot_rhat(), *34*, *77*
pointwise_loglik (criteria_doors), 26
pointwise_loglik(), *301*
post_hoc_lm, 86
posterior_predict, 83
posterior_predict(), *17*, *19*, *47*, *88*, *109*, *118*, *145*, *248*, *280*, *288*
posterior_predict.tulpa_categorical (categorical_accessors), 19
posterior_sample, 85
posterior_sample(), *14*, *61, 62*, *208*, *264*, *266*
pp_check, 87
pp_check(), *84*, *294*
pp_check.tulpa_categorical (categorical_accessors), 19

predict.tulpa_categorical (categorical_accessors), 19
predict.tulpa_fit, 88
predict.tulpa_fit(), *83*
print.laplace_diagnostics (diagnostics), 28
print.post_hoc_lm (post_hoc_lm), 86
print.sbc (sbc), 139
print.sbc_summary (sbc), 139
print.summary.tulpa_fit (print.tulpa_fit), 89
print.tgmrf (tgmrf), 188
print.tulpa_adjacency (adjacency), 6
print.tulpa_adjacency_check (check_adjacency), 21
print.tulpa_criteria (tulpa_criteria), 205
print.tulpa_diagnostic_summary (diagnostic_summary), 35
print.tulpa_family (tulpa_family), 221
print.tulpa_fit, 89
print.tulpa_geweke (geweke_test), 49
print.tulpa_gp (spatial_gp), 152
print.tulpa_hsgp (spatial_gp), 152
print.tulpa_latent (latent_factor), 55
print.tulpa_multinomial (tulpa_multinomial), 252
print.tulpa_multiscale (spatial_multiscale), 154
print.tulpa_nested_laplace, 92
print.tulpa_nested_laplace_joint, 101
print.tulpa_nested_laplace_joint_multi (print.tulpa_nested_laplace_joint), 101
print.tulpa_ordinal (tulpa_ordinal), 258
print.tulpa_parsed_formula (tulpa_parse_formula), 261
print.tulpa_prior (tulpa_priors), 268
print.tulpa_prior_predict (prior_predict), 134
print.tulpa_priors (tulpa_priors), 268
print.tulpa_rsr (spatial_rsr), 158
print.tulpa_simulate (tulpa_simulate), 287
print.tulpa_spatial_field (spatial), 147
print.tulpa_spatial_field_fit (spatial), 147
print.tulpa_st_nested (fit_st_nested), 44
print.tulpa_st_summary (spatiotemporal_effects), 167
print.tulpa_svc (spatial_svc), 163
print.tulpa_svc_posterior (svc), 170
print.tulpa_svc_summary (svc), 170
print.tulpa_temporal_field (print.tulpa_temporal_field_fit), 125
print.tulpa_temporal_field_fit, 125
print.tulpa_temporal_gp (temporal_gp), 177
print.tulpa_temporal_latent_block (temporal_ar), 172
print.tulpa_temporal_multiscale (temporal_multiscale), 179
print.tulpa_temporal_posterior (print.tulpa_temporal_field_fit), 125
print.tulpa_temporal_summary (print.tulpa_temporal_field_fit), 125
print.tulpa_tvc (temporal_tvc), 183
print.tulpa_tvc_posterior (tvc), 295
print.tulpa_tvc_summary (tvc), 295
print.tulpa_variogram (tulpa_variogram), 294
print.tulpa_wbic (wbic), 300
prior_beta, 128
prior_exponential, 129
prior_from_spec, 130
prior_from_spec(), *94*
prior_gamma, 131
prior_half_cauchy, 131
prior_half_normal, 132
prior_normal, 132
prior_pc, 133
prior_pc(), *56*
prior_predict, 134
prior_predict(), *127*, *221*, *268*, *287*, *294*
priors_default, 127

ranef, 135
ranef(), *24*, *39*, *40*, *48*, *88*, *146*, *280*, *285*, *288*, *299*
Rcpp::sourceCpp(), *191*
re_cov_pc_lkj_prior, 137
re_cov_pc_lkj_prior(), *210*, *212*, *276*, *282*, *285*

residuals(), *47*
residuals.tulpa_categorical (categorical_accessors), 19
residuals.tulpa_fit, 136
rubins_pool, 138
rubins_pool(), *95*, *215–218*

sbc, 139
sbc(), *29, 30*, *34*, *143, 144*
sbc_discrete (sbc_predictive), 143
sbc_discrete(), *260*
sbc_draws (sbc_predictive), 143
sbc_mixture (sbc_predictive), 143
sbc_mixture(), *142, 143*
sbc_normal (sbc_predictive), 143
sbc_predictive, 143
sbc_rank (sbc_predictive), 143
select_main_params, 145
simulate(), *47*
simulate.tulpa_fit, 145
simulate.tulpa_fit(), *20*, *84*
smooth_effects, 146
smooth_effects(), *39*, *91*, *136*
spatial, 147
spatial(), *6*, *9*, *21*, *126*, *200, 201*, *238*
spatial_bym2, 149
spatial_bym2(), *94*, *151, 152*, *154*, *165*, *289*
spatial_car, 150
spatial_car(), *6*, *9*, *21*, *94*, *148*, *152*, *154*, *160*, *165*, *167*, *289*
spatial_car_proper, 151
spatial_car_proper(), *150, 151*
spatial_gp, 152
spatial_gp(), *46*, *155–157*, *160*, *164, 165*, *167*, *178*, *225*, *289*, *296, 297*
spatial_multiscale, 154
spatial_multiscale(), *225*, *296, 297*
spatial_range, 157
spatial_range(), *154*
spatial_rsr, 158
spatial_rsr(), *104*, *180, 181*
spatial_spde, 161
spatial_spde(), *41, 42*, *163*, *257*, *289, 290*
spatial_spde_custom, 162
spatial_spde_custom(), *41*, *257*
spatial_svc, 163
spatial_svc(), *148*, *157*, *171*
spatiotemporal, 165
spatiotemporal(), *168*
spatiotemporal_effects, 167
spatiotemporal_gp, 169
stats::AIC(), *12*
stats::BIC(), *12*
stats::confint.default(), *26*
stats::fitted(), *19*
stats::lm(), *194*
stats::model.matrix(), *147*, *200*
stats::optim, *67*
stats::predict(), *19*, *245*
stats::residuals(), *19*
stats::simulate(), *146*
summary.sbc (sbc), 139
summary.tgmrf (tgmrf), 188
summary.tulpa_fit (print.tulpa_fit), 89
summary.tulpa_fit(), *26*, *109*, *220*, *222, 223*, *253*, *259*
summary.tulpa_svc_posterior (svc), 170
summary.tulpa_temporal_posterior (print.tulpa_temporal_field_fit), 125
summary.tulpa_tvc_posterior (tvc), 295
svc, 170
svc(), *164*

temporal (print.tulpa_temporal_field_fit), 125
temporal(), *39*
temporal_ar, 172
temporal_ar1, 173
temporal_ar1(), *94*, *165*, *167*, *173*, *175*, *178*, *180–182*, *185*, *197*, *291*
temporal_ar2, 175
temporal_ar2(), *172, 173*
temporal_corr, 176
temporal_corr(), *178*
temporal_gp, 177
temporal_gp(), *165*, *183–185*, *197*
temporal_multiscale, 179
temporal_multiscale(), *126*, *156*, *197*, *225*, *298*
temporal_rtr, 180
temporal_rw1, 181
temporal_rw1(), *94*, *126*, *165*, *167*, *178*, *180, 181*, *185*, *197*, *291*
temporal_rw2, 182
temporal_rw2(), *94*, *165*, *180–182*, *185*, *197*, *291*

temporal_tvc, 183
temporal_tvc(), *296*
test_dispersion, 185
test_outliers, 186
test_uniformity, 187
test_zero_inflation, 188
tgmrf, 188
tgmrf(), *45*, *54*, *56*, *95*, *107*, *173*, *175*, *191*, *192*, *291*, *292*
tgmrf_cpp, 191
tgmrf_cpp(), *203*, *204*, *291*, *292*
tidy.tulpa_fit, 192
tulpa, 193
tulpa(), *17*, *45*, *47*, *56*, *84*, *95*, *107*, *127*, *146*, *147*, *221*, *245*, *251*, *253*, *260*, *267*, *268*, *273*, *289*, *291*, *296*, *298*
tulpa_bar_field_replicate, 199
tulpa_bar_field_specs, 200
tulpa_bar_field_specs(), *199*, *200*, *238*
tulpa_batched_pareto_k, 202
tulpa_cache_clear, 203
tulpa_cache_clear(), *204*
tulpa_cache_dir, 204
tulpa_cache_dir(), *203*
tulpa_check_control, 205
tulpa_criteria, 205
tulpa_criteria(), *24–28*, *245*, *246*, *251*, *252*, *263*, *267*, *273*, *302*
tulpa_diagnostics, 207
tulpa_draws_array, 208
tulpa_draws_array(), *14*, *34*
tulpa_eb, 209
tulpa_eb(), *45*, *95*, *107*, *194–197*, *279*, *285*, *299*
tulpa_em_laplace, 213
tulpa_em_laplace(), *217*, *218*
tulpa_em_mc, 217
tulpa_ep, 219
tulpa_family, 221
tulpa_family(), *127*, *134*, *287*
tulpa_fit_summary, 222
tulpa_formula, 223
tulpa_gaussian, 223
tulpa_gibbs, 224
tulpa_gibbs(), *296*, *298*
tulpa_grid_axis, 226
tulpa_grid_log_quad, 227
tulpa_grid_log_quad(), *239*, *253*, *254*, *293*
tulpa_hyper_check_copy_slab, 229
tulpa_hyper_check_copy_slab(), *230*, *239*
tulpa_hyper_copy_slab_density, 229
tulpa_hyper_copy_slab_density(), *229*
tulpa_hyper_draws, 230
tulpa_hyper_draws(), *260*, *264*, *266*
tulpa_hyper_grid, 232
tulpa_hyper_grid_supports, 235
tulpa_hyper_grid_supports(), *236*, *239*
tulpa_hyper_slice_home, 236
tulpa_hyper_slice_home(), *228*, *235*
tulpa_integrator, 237
tulpa_integrator(), *302*
tulpa_is_spatial_bar, 238
tulpa_is_spatial_bar(), *201*
tulpa_joint_axis_specs_from_grid, 239
tulpa_joint_axis_specs_from_grid(), *228*, *230*, *235*, *236*
tulpa_joint_grid_batch, 240
tulpa_joint_inner_vcov_blocks, 241
tulpa_joint_inner_vcov_blocks(), *243*
tulpa_joint_inner_vcov_mixture, 243
tulpa_kfold, 245
tulpa_kfold(), *273*
tulpa_laplace, 246
tulpa_laplace(), *10*, *11*, *52*, *68*, *197*, *199*, *209–211*, *213–215*, *217*, *249*, *250*, *255*, *279*, *280*, *282*, *284*, *286*, *296*
tulpa_laplace_beta, 249
tulpa_laplace_beta(), *254*, *255*
tulpa_latent, 251
tulpa_loglik, 251
tulpa_loglik(), *205*, *206*, *301*
tulpa_multinomial, 252
tulpa_multinomial(), *19*, *21*, *145*, *194*, *259*
tulpa_nested_laplace (print.tulpa_nested_laplace), 92
tulpa_nested_laplace(), *46*, *47*, *115*, *116*, *124*, *130*, *195*, *197*, *214*, *231*, *260*, *264*, *286*
tulpa_nested_laplace_joint (print.tulpa_nested_laplace_joint), 101
tulpa_nested_laplace_joint(), *14*, *15*, *47*, *94*, *98–100*, *198*, *231*, *234*, *240*, *241*, *260*, *264–266*
tulpa_normalise_weights_safe, 253

tulpa_normalise_weights_safe(), *227, 228*, *293*
tulpa_nuts_beta, 254
tulpa_nuts_beta(), *250, 251*
tulpa_nuts_spde, 256
tulpa_nuts_spde(), *42*
tulpa_ordinal, 258
tulpa_ordinal(), *19*, *21*, *145*, *194*
tulpa_outer_grid, 260
tulpa_outer_grid(), *92*
tulpa_parse_formula, 261
tulpa_pit, 262
tulpa_pit(), *207*
tulpa_posterior_draws, 263
tulpa_posterior_draws(), *31, 32*, *34*, *85*, *208*, *230, 231*, *260*, *265, 266*
tulpa_posterior_draws.tulpa_nested_laplace_joint, 265
tulpa_powerscale_sensitivity, 266
tulpa_priors, 268
tulpa_priors(), *127, 128*, *134*
tulpa_profile, 270
tulpa_psis, 271
tulpa_psis(), *24, 25*, *30*, *34*, *42*, *62*, *95*, *202, 203*, *205*, *207*, *263*, *266, 267*
tulpa_re_aghq, 274
tulpa_re_aghq(), *282*
tulpa_re_cov_gibbs, 278
tulpa_re_cov_gibbs(), *135*, *198*
tulpa_re_cov_nested, 281
tulpa_re_cov_nested(), *32*, *45*, *95*, *107*, *136–138*, *196*, *198*, *209, 210*, *212, 213*, *266*, *279, 280*
tulpa_reloo, 273
tulpa_rpg, 286
tulpa_simulate, 287
tulpa_simulate(), *127*, *221*, *268*
tulpa_spatial, 288
tulpa_spde_log_hyperprior, 289
tulpa_spde_log_hyperprior(), *290*
tulpa_spde_precision_Q, 290
tulpa_spde_precision_Q(), *289*
tulpa_temporal, 291
tulpa_tgmrf, 291
tulpa_theta_matrix, 293
tulpa_theta_matrix(), *227, 228*, *235*, *239*, *253, 254*
tulpa_validate, 294
tulpa_variogram, 294
tvc, 295

validate_gp, 296
validate_gp(), *298*
validate_mode, 297
validate_temporal_multiscale, 298
validate_temporal_multiscale(), *297*
VarCorr, 299
VarCorr(), *91, 92*
vcov(), *88*, *90*, *220*, *253*, *259*
vcov.tulpa_fit, 300
vcov.tulpa_fit(), *109*

waic.tulpa_fit *(*criteria_doors*)*, 26
wbic, 300
wbic(), *24*
with_tulpa_integrator, 302
with_tulpa_integrator(), *238*